

Max5 API

5.1.7

Generated by Doxygen 1.6.3

Tue Jan 25 14:35:38 2011

Contents

1 Objects in C: A Roadmap	1
1.1 Max Objects	1
1.2 MSP Objects	1
1.3 Jitter Objects	2
2 Development System Information	3
2.1 Building	4
2.2 Mac	4
2.2.1 XCode Project Setup	4
2.2.2 Linking and Frameworks	4
2.3 Windows	5
2.3.1 Compiling with Cygwin	5
2.3.1.1 Requirements	5
2.3.1.2 Build Steps	6
2.3.1.3 Additional Notes	6
2.4 Important Project Settings	7
2.4.1 Mac	7
2.4.2 Windows	7
2.5 Platform-specificity	7
3 Anatomy of a Max Object	9
3.1 Include Files	10
3.2 The Object Declaration	10
3.3 Initialization Routine	10
3.4 New Instance Routine	11
3.5 Message Handlers	12
4 Inlets and Outlets	13
4.1 Creating and Using Inlets	14

4.2	Creating and Using Outlets	15
4.3	Creating and Using Proxies	16
4.4	Example	16
5	Atoms and Messages	19
5.1	Argument Type Specifiers	21
5.2	Writing A_GIMME Functions	21
5.3	Writing "Anything" Methods	22
6	The Scheduler	23
6.1	Creating and Using Clocks	24
6.2	Creating and Using Qelems	25
6.3	Defer	26
6.3.1	Defer Variants	26
6.4	Schedule	27
7	Memory Allocation	29
8	Anatomy of a MSP Object	31
8.1	Additional Header Files	32
8.2	C Structure Declaration	32
8.3	Initialization Routine	32
8.4	New Instance Routine	32
8.5	The DSP Method and Perform Routine	33
8.6	Free Function	34
9	Advanced Signal Object Topics	35
9.1	Saving Internal State	36
9.2	Observing Patcher Muting	36
9.3	Using Connection Information	37
10	Sending Messages, Calling Methods	39
10.1	Attributes	40
10.1.1	Attribute Basics	40
10.1.2	Defining Attributes	41
10.1.3	Attributes With Custom Getters and Setters	41
10.2	Receiving Notifications	42
11	Anatomy of a UI Object	45

11.1 Required Headers	46
11.2 UI Object Data Structure	47
11.3 Initialization Routine for UI Objects	47
11.4 UI Object Methods	48
11.5 Defining Attributes	49
11.5.1 Standard Color Attribute	51
11.5.2 Setting a Default Size	51
11.6 New Instance Routine	51
11.7 Dynamic Updating	53
11.8 The Paint Method	53
11.9 Handling Mouse Gestures	55
11.10 Freeing a UI Object	57
12 File Handling	59
12.1 Reading Text Files	61
12.2 Reading Data Files	61
12.3 Writing Files	62
13 Scripting the Patcher	65
13.1 Knowing the Patcher	66
13.1.1 Patcher Name and File Path	66
13.1.2 Patcher Hierarchy	66
13.1.3 Getting Objects in a Patcher	67
13.1.4 Iteration Using Callbacks	67
13.2 Creating Objects	68
13.2.1 Connecting Objects	69
13.3 Deleting Objects	69
13.4 Obtaining and Changing Patcher and Object Attributes	69
13.4.1 Patcher Attributes	71
13.4.2 Box Attributes	73
14 Enhancements to Objects	75
14.1 Preset Support	76
14.2 Pattr Support	76
14.3 Assistance	76
14.4 Hot and Cold Inlets	77
14.5 Showing a Text Editor	77
14.6 Accessing Data in table Objects	79

15 Data Structures	81
15.1 Linked Lists	82
15.2 Hash Tables	83
16 Threading	85
16.1 Max Threading Operation	86
16.2 Thread Protection	87
16.2.1 When Messages Arrive	87
16.2.2 Critical Section Example	87
17 Drag'n'Drop	89
17.1 Discussion	90
18 ITM	93
18.1 Scheduling Temporary Events	94
18.2 Permanent Events	96
18.3 Cleaning Up	97
19 Jitter Object Model	99
19.1 Jitter Object Model Basics	100
19.2 Defining a Jitter Class	100
19.3 Object Struct	101
19.4 Constructor/Destructor	102
19.5 Methods	102
19.6 Attributes	103
19.7 Array Attributes	105
19.8 Attribute Notification	105
20 Jitter Max Wrappers	107
20.1 Max Wrapper Classes	108
20.2 Object Struct	109
20.3 Defining Your Max Class	110
20.4 Constructor	110
20.5 Destructor	110
20.6 Dumpout	111
20.7 Additional inlets/outlets	111
20.8 Max Wrapper Attributes	111
21 Matrix Operator QuickStart	113

21.1 Defining the MOP Jitter Class	114
21.2 The Jitter Class Constructor/Destructor	115
21.3 The Matrix Calculation Method	115
21.4 Processing N-Dimensional Matrices	116
21.5 Defining the MOP Max Wrapper Class	118
21.6 The Max Class Constructor/Destructor	119
22 Matrix Operator Details	121
22.1 Defining the MOP Jitter Class	122
22.2 The jit_mop_io Object	122
22.3 Restricting Input/Output Attributes	123
22.4 The ioproc Function	123
22.5 Variable Inputs/Outputs	124
22.6 Adding jit_mop as a Class Adornment	124
22.7 The Matrix Calculation Method	124
22.8 Accessing the Input and Output Lists	124
22.9 Locking and Unlocking Matrices	125
22.10 Retrieving Matrix Information	125
22.11 Retrieving the Data Pointer	126
22.12 Processing the Data	126
22.13 Processing N-Dimensional Matrices	127
22.14 Defining the MOP Max Wrapper Class	127
22.15 Overriding the jit_matrix Method	128
22.16 Overriding the bang and outputmatrix Methods	128
22.17 Overriding the name, type, dim, and planecount Attributes	129
22.18 Overriding the clear and notify Methods	129
22.19 Overriding the adapt and outputmode Attributes	129
22.20 Defining an mproc Method	130
22.21 The Max Class Constructor/Destructor	130
22.21.1 Variable Inputs/Outputs	130
22.21.2 Matrix Arguments	131
23 OB3D QuickStart	133
23.1 Defining the OB3D Jitter Class	134
23.2 The Jitter Class Constructor/Destructor	135
23.3 The OB3D draw Method	136
23.4 Defining the OB3D Max Wrapper Class	136

23.5 The Max Class Constructor/Destructor	137
24 OB3D Details	139
24.1 Defining the OB3D Jitter Class	140
24.2 Declaring a Draw Method	140
24.3 Declaring Destination and Geometry Related Methods	140
24.4 Declaring a Register Method	140
24.5 Overriding Rotation and Scale Related Attributes	141
24.6 Overriding Color Related Attributes	141
24.7 Overriding Texture Related Attributes	141
24.8 Overriding Lighting and Material Related Attributes	141
24.9 Overriding Fog Related Attributes	141
24.10 Overriding Polygon Variable Related Attributes	142
24.11 Overriding Blending Related Attributes	142
24.12 Overriding Depth Buffer and Antialiasing Related Attributes	142
24.13 Overriding Matrixoutput and Automatic Attributes	142
24.14 Declaring a User Interface Object	142
24.15 The Jitter Class Constructor and Destructor	143
24.16 The OB3D Draw Method	143
24.17 The t_jit_glchunk Structure and Matrix Output	143
24.18 OB3D OpenGL Caveats	144
24.19 Getting Information About the OB3D Attributes	144
24.20 Getting Information About the Context	144
24.21 Playing Well with Others	145
24.22 Defining the OB3D Max Wrapper Class	145
24.23 Matrix Input	145
25 Scheduler and Low Priority Queue Issues	147
25.1 Defer and Usurp	148
25.2 Using Defer and Usurp in Jitter Object Methods	148
25.3 Using Defer and Usurp in Jitter Object Attributes	149
25.4 Using Defer and Usurp in the Max Wrapper Object	150
25.5 When Not to Use the Usurp Mechanism	150
25.6 Overriding Defer and Usurp	151
26 Jitter Object Registration and Notification	153
26.1 Registering Named Objects	154
26.2 Looking Up an Object by Name	154

26.3 Attaching to Named Objects	155
26.4 Notifying Clients	155
27 Using Jitter Objects in C	157
27.1 Example 1: the t_jit_qt_movie object	158
28 JXF File Specification	161
28.1 The Binary JXF API	162
28.2 Specification of the JXF Format	163
29 Jitter Networking Specification	167
30 Appendix: Messages sent to Objects	171
30.1 Messages for All Objects	172
30.2 Messages for Non-UI Objects	172
30.3 Messages for User Interface Objects	173
30.4 Message for Audio Objects	173
30.5 Messages for Objects Containing Text Fields	174
30.6 Messages for Objects with Text Editor Windows	174
30.7 Messages for Dataview Client Objects	174
31 Appendix: Providing Icons for UI Objects	175
31.1 Object SVG Icon	176
31.2 Object Palette Definition	176
32 Appendix: Additional Resources	179
33 Module Documentation	181
33.1 Attributes	181
33.1.1 Detailed Description	194
33.1.2 Setting and Getting Attribute Values	194
33.1.2.1 Writing a custom Attribute Getter	195
33.1.2.2 Writing a custom Attribute Getter	195
33.1.3 Attribute Notificaton	195
33.1.4 Define Documentation	195
33.1.4.1 CLASS_ATTR_ACCESSORS	195
33.1.4.2 CLASS_ATTR_ADD_FLAGS	196
33.1.4.3 CLASS_ATTR_ALIAS	196
33.1.4.4 CLASS_ATTR_ATOM	197

33.1.4.5 CLASS_ATTR_ATOM_ARRAY	197
33.1.4.6 CLASS_ATTR_ATOM_VARSIZE	197
33.1.4.7 CLASS_ATTR_CATEGORY	198
33.1.4.8 CLASS_ATTR_CHAR	198
33.1.4.9 CLASS_ATTR_CHAR_ARRAY	198
33.1.4.10 CLASS_ATTR_CHAR_VARSIZE	199
33.1.4.11 CLASS_ATTR_DEFAULT	199
33.1.4.12 CLASS_ATTR_DEFAULT_PAINT	199
33.1.4.13 CLASS_ATTR_DEFAULT_SAVE	200
33.1.4.14 CLASS_ATTR_DEFAULT_SAVE_PAINT	200
33.1.4.15 CLASS_ATTR_DEFAULTNAME	200
33.1.4.16 CLASS_ATTR_DEFAULTNAME_PAINT	201
33.1.4.17 CLASS_ATTR_DEFAULTNAME_SAVE	201
33.1.4.18 CLASS_ATTR_DEFAULTNAME_SAVE_PAINT	201
33.1.4.19 CLASS_ATTR_DOUBLE	202
33.1.4.20 CLASS_ATTR_DOUBLE_ARRAY	202
33.1.4.21 CLASS_ATTR_DOUBLE_VARSIZE	203
33.1.4.22 CLASS_ATTR_ENUM	203
33.1.4.23 CLASS_ATTR_ENUMINDEX	203
33.1.4.24 CLASS_ATTR_FILTER_CLIP	204
33.1.4.25 CLASS_ATTR_FILTER_MAX	204
33.1.4.26 CLASS_ATTR_FILTER_MIN	205
33.1.4.27 CLASS_ATTR_FLOAT	205
33.1.4.28 CLASS_ATTR_FLOAT_ARRAY	205
33.1.4.29 CLASS_ATTR_FLOAT_VARSIZE	206
33.1.4.30 CLASS_ATTR_INVISIBLE	206
33.1.4.31 CLASS_ATTR_LABEL	206
33.1.4.32 CLASS_ATTR_LONG	207
33.1.4.33 CLASS_ATTR_LONG_ARRAY	207
33.1.4.34 CLASS_ATTR_LONG_VARSIZE	207
33.1.4.35 CLASS_ATTR_MAX	208
33.1.4.36 CLASS_ATTR_MIN	208
33.1.4.37 CLASS_ATTR_OBJ	208
33.1.4.38 CLASS_ATTR_OBJ_ARRAY	209
33.1.4.39 CLASS_ATTR_OBJ_VARSIZE	209
33.1.4.40 CLASS_ATTR_ORDER	209

33.1.4.41 CLASS_ATTR_PAINT	210
33.1.4.42 CLASS_ATTR_REMOVE_FLAGS	210
33.1.4.43 CLASS_ATTR_RGBA	210
33.1.4.44 CLASS_ATTR_SAVE	211
33.1.4.45 CLASS_ATTR_STYLE	211
33.1.4.46 CLASS_ATTR_STYLE_LABEL	212
33.1.4.47 CLASS_ATTR_SYM	212
33.1.4.48 CLASS_ATTR_SYM_ARRAY	212
33.1.4.49 CLASS_ATTR_SYM_VARSIZE	213
33.1.4.50 CLASS_METHOD_ATTR_PARSE	213
33.1.4.51 CLASS_STICKY_ATTR	214
33.1.4.52 CLASS_STICKY_ATTR_CLEAR	214
33.1.4.53 CLASS_STICKY_METHOD	214
33.1.4.54 CLASS_STICKY_METHOD_CLEAR	215
33.1.4.55 OBJ_ATTR_ATOM	215
33.1.4.56 OBJ_ATTR_ATOM_ARRAY	216
33.1.4.57 OBJ_ATTR_CHAR	216
33.1.4.58 OBJ_ATTR_CHAR_ARRAY	216
33.1.4.59 OBJ_ATTR_DEFAULT	216
33.1.4.60 OBJ_ATTR_DEFAULT_SAVE	217
33.1.4.61 OBJ_ATTR_DOUBLE	217
33.1.4.62 OBJ_ATTR_DOUBLE_ARRAY	217
33.1.4.63 OBJ_ATTR_FLOAT	218
33.1.4.64 OBJ_ATTR_FLOAT_ARRAY	218
33.1.4.65 OBJ_ATTR_LONG	218
33.1.4.66 OBJ_ATTR_LONG_ARRAY	218
33.1.4.67 OBJ_ATTR_OBJ	219
33.1.4.68 OBJ_ATTR_OBJ_ARRAY	219
33.1.4.69 OBJ_ATTR_SAVE	219
33.1.4.70 OBJ_ATTR_SYM	219
33.1.4.71 OBJ_ATTR_SYM_ARRAY	220
33.1.4.72 STATIC_ATTR_ATOM	220
33.1.4.73 STATIC_ATTR_ATOM_ARRAY	220
33.1.4.74 STATIC_ATTR_CHAR	220
33.1.4.75 STATIC_ATTR_CHAR_ARRAY	221
33.1.4.76 STATIC_ATTR_DOUBLE	221

33.1.4.77 STATIC_ATTR_DOUBLE_ARRAY	221
33.1.4.78 STATIC_ATTR_FLOAT	221
33.1.4.79 STATIC_ATTR_FLOAT_ARRAY	222
33.1.4.80 STATIC_ATTR_LONG	222
33.1.4.81 STATIC_ATTR_LONG_ARRAY	222
33.1.4.82 STATIC_ATTR_OBJ	222
33.1.4.83 STATIC_ATTR_OBJ_ARRAY	223
33.1.4.84 STATIC_ATTR_SYM	223
33.1.4.85 STATIC_ATTR_SYM_ARRAY	223
33.1.4.86 STRUCT_ATTR_ATOM	223
33.1.4.87 STRUCT_ATTR_ATOM_ARRAY	224
33.1.4.88 STRUCT_ATTR_ATOM_VARSIZE	224
33.1.4.89 STRUCT_ATTR_CHAR	224
33.1.4.90 STRUCT_ATTR_CHAR_ARRAY	225
33.1.4.91 STRUCT_ATTR_CHAR_VARSIZE	225
33.1.4.92 STRUCT_ATTR_DOUBLE	226
33.1.4.93 STRUCT_ATTR_DOUBLE_ARRAY	226
33.1.4.94 STRUCT_ATTR_DOUBLE_VARSIZE	226
33.1.4.95 STRUCT_ATTR_FLOAT	227
33.1.4.96 STRUCT_ATTR_FLOAT_ARRAY	227
33.1.4.97 STRUCT_ATTR_FLOAT_VARSIZE	227
33.1.4.98 STRUCT_ATTR_LONG	228
33.1.4.99 STRUCT_ATTR_LONG_ARRAY	228
33.1.4.100STRUCT_ATTR_LONG_VARSIZE	228
33.1.4.101STRUCT_ATTR_OBJ	229
33.1.4.102STRUCT_ATTR_OBJ_ARRAY	229
33.1.4.103STRUCT_ATTR_OBJ_VARSIZE	229
33.1.4.104STRUCT_ATTR_SYM	230
33.1.4.105STRUCT_ATTR_SYM_ARRAY	230
33.1.4.106STRUCT_ATTR_SYM_VARSIZE	230
33.1.5 Enumeration Type Documentation	231
33.1.5.1 e_max_attrflags	231
33.1.6 Function Documentation	231
33.1.6.1 attr_addfilter_clip	231
33.1.6.2 attr_addfilter_clip_scale	232
33.1.6.3 attr_addfilterget_clip	232

33.1.6.4 attr_addfilterget_clip_scale	232
33.1.6.5 attr_addfilterget_proc	233
33.1.6.6 attr_addfilterset_clip	233
33.1.6.7 attr_addfilterset_clip_scale	234
33.1.6.8 attr_addfilterset_proc	234
33.1.6.9 attr_args_dictionary	235
33.1.6.10 attr_args_offset	235
33.1.6.11 attr_args_process	236
33.1.6.12 attr_dictionary_process	236
33.1.6.13 attr_offset_array_new	237
33.1.6.14 attr_offset_new	237
33.1.6.15 attribute_new	238
33.1.6.16 object_addattr	239
33.1.6.17 object_attr_get	240
33.1.6.18 object_attr_get_rect	240
33.1.6.19 object_attr_getchar_array	240
33.1.6.20 object_attr_getcolor	241
33.1.6.21 object_attr_getdouble_array	241
33.1.6.22 object_attr_getdump	241
33.1.6.23 object_attr_getfloat	242
33.1.6.24 object_attr_getfloat_array	242
33.1.6.25 object_attr_getjrgba	242
33.1.6.26 object_attr_getlong	243
33.1.6.27 object_attr_getlong_array	243
33.1.6.28 object_attr_getpt	243
33.1.6.29 object_attr_getsize	244
33.1.6.30 object_attr_getsym	244
33.1.6.31 object_attr_getsym_array	244
33.1.6.32 object_attr_method	245
33.1.6.33 object_attr_set_rect	245
33.1.6.34 object_attr_setchar_array	245
33.1.6.35 object_attr_setcolor	246
33.1.6.36 object_attr_setdouble_array	246
33.1.6.37 object_attr_setfloat	246
33.1.6.38 object_attr_setfloat_array	247
33.1.6.39 object_attr_setjrgba	247

33.1.6.40 object_attr_setlong	247
33.1.6.41 object_attr_setlong_array	248
33.1.6.42 object_attr_setparse	248
33.1.6.43 object_attr_setpt	248
33.1.6.44 object_attr_setsize	249
33.1.6.45 object_attr_setsym	249
33.1.6.46 object_attr_setsym_array	249
33.1.6.47 object_attr_setvalueof	250
33.1.6.48 object_attr_usercanget	250
33.1.6.49 object_attr_usercanset	250
33.1.6.50 object_chuckattr	250
33.1.6.51 object_deleteattr	251
33.1.6.52 object_new_parse	251
33.2 Classes	252
33.2.1 Detailed Description	254
33.2.2 Define Documentation	254
33.2.2.1 CLASS_BOX	254
33.2.3 Enumeration Type Documentation	254
33.2.3.1 e_max_class_flags	254
33.2.4 Function Documentation	255
33.2.4.1 class_addattr	255
33.2.4.2 class_addmethod	255
33.2.4.3 class_alias	255
33.2.4.4 class_dumpout_wrap	256
33.2.4.5 class_findbyname	256
33.2.4.6 class_findbyname_casefree	256
33.2.4.7 class_free	256
33.2.4.8 class_is_ui	257
33.2.4.9 class_nameget	257
33.2.4.10 class_new	257
33.2.4.11 class_obexoffset_get	258
33.2.4.12 class_obexoffset_set	258
33.2.4.13 class_register	258
33.3 Old-Style Classes	259
33.3.1 Function Documentation	260
33.3.1.1 addbang	260

33.3.1.2 addfloat	260
33.3.1.3 addftx	260
33.3.1.4 addint	261
33.3.1.5 addinx	261
33.3.1.6 addmess	261
33.3.1.7 alias	261
33.3.1.8 class_setname	262
33.3.1.9 egetfn	262
33.3.1.10 freeobject	262
33.3.1.11 getfn	262
33.3.1.12 newinstance	263
33.3.1.13 newobject	263
33.3.1.14 setup	264
33.3.1.15 typedmess	264
33.3.1.16 zgetfn	265
33.4 Inlets and Outlets	266
33.4.1 Detailed Description	267
33.4.2 Function Documentation	267
33.4.2.1 bangout	267
33.4.2.2 floatin	267
33.4.2.3 floatout	267
33.4.2.4 inlet_new	268
33.4.2.5 intin	268
33.4.2.6 intout	269
33.4.2.7 listout	269
33.4.2.8 outlet_anything	269
33.4.2.9 outlet_bang	270
33.4.2.10 outlet_float	270
33.4.2.11 outlet_int	270
33.4.2.12 outlet_list	271
33.4.2.13 outlet_new	271
33.4.2.14 proxy_getinlet	272
33.4.2.15 proxy_new	272
33.5 Data Storage	273
33.5.1 Detailed Description	274
33.5.2 Typedef Documentation	274

33.5.2.1 <code>t_cmpfn</code>	274
33.5.3 Enumeration Type Documentation	275
33.5.3.1 <code>e_max_datastore_flags</code>	275
33.6 Atom Array	276
33.6.1 Detailed Description	277
33.6.2 Define Documentation	277
33.6.2.1 <code>ATOMARRAY_FLAG_FREECHILDREN</code>	277
33.6.3 Function Documentation	277
33.6.3.1 <code>atomarray_appendatom</code>	277
33.6.3.2 <code>atomarray_appendatoms</code>	278
33.6.3.3 <code>atomarray_chuckindex</code>	278
33.6.3.4 <code>atomarray_clear</code>	278
33.6.3.5 <code>atomarray_copyatoms</code>	278
33.6.3.6 <code>atomarray_duplicate</code>	279
33.6.3.7 <code>atomarray_flags</code>	279
33.6.3.8 <code>atomarray_funall</code>	279
33.6.3.9 <code>atomarray_getatoms</code>	280
33.6.3.10 <code>atomarray_getflags</code>	280
33.6.3.11 <code>atomarray_getindex</code>	281
33.6.3.12 <code>atomarray_getsize</code>	281
33.6.3.13 <code>atomarray_new</code>	281
33.6.3.14 <code>atomarray_setatoms</code>	282
33.7 Database	283
33.7.1 Detailed Description	285
33.7.2 Typedef Documentation	285
33.7.2.1 <code>t_database</code>	285
33.7.2.2 <code>t_db_result</code>	285
33.7.2.3 <code>t_db_view</code>	285
33.7.3 Function Documentation	285
33.7.3.1 <code>db_close</code>	285
33.7.3.2 <code>db_open</code>	286
33.7.3.3 <code>db_query</code>	286
33.7.3.4 <code>db_query_getlastinsertid</code>	286
33.7.3.5 <code>db_query_silent</code>	287
33.7.3.6 <code>db_query_table_addcolumn</code>	287
33.7.3.7 <code>db_query_table_new</code>	287

33.7.3.8 db_result_clear	288
33.7.3.9 db_result_datetimeinseconds	288
33.7.3.10 db_result_fieldname	288
33.7.3.11 db_result_float	288
33.7.3.12 db_result_long	289
33.7.3.13 db_result_nextrecord	289
33.7.3.14 db_result_numfields	289
33.7.3.15 db_result_numrecords	289
33.7.3.16 db_result_reset	290
33.7.3.17 db_result_string	290
33.7.3.18 db_transaction_end	290
33.7.3.19 db_transaction_flush	290
33.7.3.20 db_transaction_start	291
33.7.3.21 db_util_datetostring	291
33.7.3.22 db_util_stringtodate	291
33.7.3.23 db_view_create	291
33.7.3.24 db_view_getresult	292
33.7.3.25 db_view_remove	292
33.7.3.26 db_view_setquery	292
33.8 Dictionary	293
33.8.1 Detailed Description	296
33.8.2 Using Dictionaries	296
33.8.2.1 Understanding Dictionaries	296
33.8.2.2 When to Free a Dictionary	297
33.8.2.3 Some Common Uses of Dictionaries	298
33.8.3 Function Documentation	298
33.8.3.1 dictionary_appendatom	298
33.8.3.2 dictionary_appendatomarray	298
33.8.3.3 dictionary_appendatoms	299
33.8.3.4 dictionary_appenddictionary	299
33.8.3.5 dictionary_appendfloat	300
33.8.3.6 dictionary_appendlong	300
33.8.3.7 dictionary_appendobject	300
33.8.3.8 dictionary_appendstring	301
33.8.3.9 dictionary_appendsym	301
33.8.3.10 dictionary_chuckentry	301

33.8.3.11 dictionary_clear	302
33.8.3.12 dictionary_copyatoms	302
33.8.3.13 dictionary_copydefatoms	302
33.8.3.14 dictionary_copyentries	303
33.8.3.15 dictionary_copyunique	303
33.8.3.16 dictionary_deleteentry	304
33.8.3.17 dictionary_dump	304
33.8.3.18 dictionary_entry_getkey	304
33.8.3.19 dictionary_entry_getvalue	305
33.8.3.20 dictionary_entryisatomarray	305
33.8.3.21 dictionary_entryisdictionary	305
33.8.3.22 dictionary_entryisstring	305
33.8.3.23 dictionary_freekeys	306
33.8.3.24 dictionary_funall	306
33.8.3.25 dictionary_getatom	306
33.8.3.26 dictionary_getatomarray	307
33.8.3.27 dictionary_getatoms	307
33.8.3.28 dictionary_getdefatom	307
33.8.3.29 dictionary_getdefatoms	308
33.8.3.30 dictionary_getdeffloat	308
33.8.3.31 dictionary_getdeflong	309
33.8.3.32 dictionary_getdefstring	309
33.8.3.33 dictionary_getdefsym	310
33.8.3.34 dictionary_getdictionary	310
33.8.3.35 dictionary_getentrycount	310
33.8.3.36 dictionary_getfloat	311
33.8.3.37 dictionary_getkeys	311
33.8.3.38 dictionary_getlong	312
33.8.3.39 dictionary_getobject	312
33.8.3.40 dictionary_getstring	312
33.8.3.41 dictionary_getsym	312
33.8.3.42 dictionary_hasentry	313
33.8.3.43 dictionary_new	313
33.8.3.44 dictionary_read	313
33.8.3.45 dictionary_sprintf	314
33.8.3.46 dictionary_write	314

33.8.3.47 postdictionary	315
33.9 Hash Table	316
33.9.1 Detailed Description	317
33.9.2 Define Documentation	318
33.9.2.1 HASH_DEFSLOTS	318
33.9.3 Function Documentation	318
33.9.3.1 hashtab_chuck	318
33.9.3.2 hashtab_chuckkey	318
33.9.3.3 hashtab_clear	319
33.9.3.4 hashtab_delete	319
33.9.3.5 hashtab_findfirst	319
33.9.3.6 hashtab_flags	320
33.9.3.7 hashtab_funall	320
33.9.3.8 hashtab_getflags	321
33.9.3.9 hashtab_getkeyflags	321
33.9.3.10 hashtab_getkeys	321
33.9.3.11 hashtab_getsize	322
33.9.3.12 hashtab_keyflags	322
33.9.3.13 hashtab_lookup	322
33.9.3.14 hashtab_lookupflags	323
33.9.3.15 hashtab_methodall	323
33.9.3.16 hashtab_new	323
33.9.3.17 hashtab_print	324
33.9.3.18 hashtab_READONLY	324
33.9.3.19 hashtab_STORE	324
33.9.3.20 hashtab_STORE_SAFE	325
33.9.3.21 hashtab_STOREFLAGS	325
33.10 Index Map	326
33.10.1 Detailed Description	327
33.10.2 Function Documentation	327
33.10.2.1 indexmap_append	327
33.10.2.2 indexmap_clear	327
33.10.2.3 indexmap_datafromindex	327
33.10.2.4 indexmap_delete	328
33.10.2.5 indexmap_delete_index	328
33.10.2.6 indexmap_delete_index_multi	328

33.10.2.7 <code>indexmap_delete_multi</code>	328
33.10.2.8 <code>indexmap_getsize</code>	329
33.10.2.9 <code>indexmap_indexfromdata</code>	329
33.10.2.10 <code>indexmap_move</code>	329
33.10.2.11 <code>indexmap_new</code>	329
33.10.2.12 <code>indexmap_sort</code>	330
33.11 Linked List	331
33.11.1 Detailed Description	333
33.11.2 Function Documentation	333
33.11.2.1 <code>linklist_append</code>	333
33.11.2.2 <code>linklist_chuck</code>	334
33.11.2.3 <code>linklist_chuckindex</code>	334
33.11.2.4 <code>linklist_chuckobject</code>	334
33.11.2.5 <code>linklist_clear</code>	335
33.11.2.6 <code>linklist_deleteindex</code>	335
33.11.2.7 <code>linklist_findall</code>	335
33.11.2.8 <code>linklist_findfirst</code>	336
33.11.2.9 <code>linklist_flags</code>	337
33.11.2.10 <code>linklist_funall</code>	337
33.11.2.11 <code>linklist_funall_break</code>	337
33.11.2.12 <code>linklist_funindex</code>	338
33.11.2.13 <code>linklist_getflags</code>	338
33.11.2.14 <code>linklist_getindex</code>	338
33.11.2.15 <code>linklist_getsize</code>	339
33.11.2.16 <code>linklist_insert_sorted</code>	339
33.11.2.17 <code>linklist_insertafterobjptr</code>	339
33.11.2.18 <code>linklist_insertbeforeobjptr</code>	340
33.11.2.19 <code>linklist_insertindex</code>	340
33.11.2.20 <code>linklist_last</code>	340
33.11.2.21 <code>linklist_makearray</code>	340
33.11.2.22 <code>linklist_match</code>	341
33.11.2.23 <code>linklist_methodall</code>	341
33.11.2.24 <code>linklist_methodindex</code>	341
33.11.2.25 <code>linklist_moveafterobjptr</code>	342
33.11.2.26 <code>linklist_movebeforeobjptr</code>	342
33.11.2.27 <code>linklist_new</code>	342

33.11.2.28linklist_next	342
33.11.2.29linklist_objptr2index	343
33.11.2.30linklist_prev	343
33.11.2.31linklist_READONLY	343
33.11.2.32linklist_reverse	343
33.11.2.33linklist_rotate	344
33.11.2.34linklist_shuffle	344
33.11.2.35linklist_sort	344
33.11.2.36linklist_substitute	344
33.11.2.37linklist_swap	345
33.12Quick Map	346
33.12.1 Detailed Description	346
33.12.2 Function Documentation	346
33.12.2.1 quickmap_add	346
33.12.2.2 quickmap_drop	347
33.12.2.3 quickmap_lookup_key1	347
33.12.2.4 quickmap_lookup_key2	347
33.12.2.5 quickmap_new	348
33.12.2.6 quickmap_READONLY	348
33.13String Object	349
33.13.1 Detailed Description	349
33.13.2 Function Documentation	349
33.13.2.1 string_getptr	349
33.13.2.2 string_new	350
33.14Symbol Object	351
33.14.1 Detailed Description	351
33.14.2 Function Documentation	351
33.14.2.1 symobject_linklist_match	351
33.14.2.2 symobject_new	352
33.15Data Types	353
33.15.1 Typedef Documentation	354
33.15.1.1 t_max_err	354
33.16Atoms	355
33.16.1 Enumeration Type Documentation	358
33.16.1.1 e_max_atom_gettext_flags	358
33.16.1.2 e_max_atomtypes	359

33.16.2 Function Documentation	360
33.16.2.1 atom_alloc	360
33.16.2.2 atom_alloc_array	360
33.16.2.3 atom_arg_getdouble	360
33.16.2.4 atom_arg_getfloat	361
33.16.2.5 atom_arg_getlong	361
33.16.2.6 atom_arg_getobjcclass	362
33.16.2.7 atom_arg_getsym	362
33.16.2.8 atom_copy	363
33.16.2.9 atom_getatom_array	363
33.16.2.10 atom_getchar_array	363
33.16.2.11 atom_getcharfix	364
33.16.2.12 atom_getdouble_array	364
33.16.2.13 atom_getfloat	364
33.16.2.14 atom_getfloat_array	365
33.16.2.15 atom_getformat	365
33.16.2.16 atom_getlong	366
33.16.2.17 atom_getlong_array	366
33.16.2.18 atom_getobj	366
33.16.2.19 atom_getobj_array	367
33.16.2.20 atom_getobjcclass	367
33.16.2.21 atom_getsym	367
33.16.2.22 atom_getsym_array	367
33.16.2.23 atom_gettext	368
33.16.2.24 atom_gettype	368
33.16.2.25 atom_setatom_array	368
33.16.2.26 atom_setchar_array	369
33.16.2.27 atom_setdouble_array	369
33.16.2.28 atom_setfloat	369
33.16.2.29 atom_setfloat_array	370
33.16.2.30 atom_setformat	370
33.16.2.31 atom_setlong	370
33.16.2.32 atom_setlong_array	371
33.16.2.33 atom_setobj	371
33.16.2.34 atom_setobj_array	371
33.16.2.35 atom_setparse	372

33.16.2.36atom_setsym	372
33.16.2.37atom_setsym_array	372
33.16.2.38atomisatomarray	373
33.16.2.39atomisdictionary	373
33.16.2.40atomisstring	373
33.16.2.41postargs	373
33.17Atombufs	374
33.17.1 Detailed Description	374
33.17.2 Function Documentation	374
33.17.2.1 atombuf_free	374
33.17.2.2 atombuf_new	374
33.17.2.3 atombuf_text	375
33.18Binbufs	376
33.18.1 Detailed Description	376
33.18.2 Function Documentation	377
33.18.2.1 binbuf_append	377
33.18.2.2 binbuf_eval	377
33.18.2.3 binbuf_getatom	377
33.18.2.4 binbuf_insert	378
33.18.2.5 binbuf_new	378
33.18.2.6 binbuf_set	378
33.18.2.7 binbuf_text	379
33.18.2.8 binbuf_totext	379
33.18.2.9 binbuf_vinsert	380
33.18.2.10readatom	380
33.19Symbols	382
33.19.1 Detailed Description	382
33.19.2 Function Documentation	383
33.19.2.1 gensym	383
33.20Files and Folders	384
33.20.1 Detailed Description	388
33.20.2 The Sysfile API	389
33.20.3 Example: filebyte (notes from the IRCAM workshop)	389
33.20.3.1 Paths	389
33.20.3.2 t_filehandle	389
33.20.3.3 File Names	389

33.20.3.4 File Path Names	390
33.20.4 Collectives and Fileusage	390
33.20.5 Filewatchers	390
33.20.6 Define Documentation	390
33.20.6.1 MAX_FILENAME_CHARS	390
33.20.7 Typedef Documentation	390
33.20.7.1 t_filehandle	390
33.20.8 Enumeration Type Documentation	391
33.20.8.1 e_max_fileinfo_flags	391
33.20.8.2 e_max_openfile_permissions	391
33.20.8.3 e_max_path_folder_flags	391
33.20.8.4 e_max_path_styles	391
33.20.8.5 e_max_path_types	392
33.20.8.6 e_max_sysfile_posmodes	392
33.20.8.7 e_max_sysfile_textflags	392
33.20.9 Function Documentation	393
33.20.9.1 fileusage_addfile	393
33.20.9.2 filewatcher_new	393
33.20.9.3 locatefile	393
33.20.9.4 locatefile_extended	394
33.20.9.5 locatefiletype	395
33.20.9.6 open_dialog	395
33.20.9.7 open_promptset	396
33.20.9.8 path_closefolder	396
33.20.9.9 path_createssysfile	396
33.20.9.10path_fileinfo	397
33.20.9.11path_foldernextfile	397
33.20.9.12path_from pathname	397
33.20.9.13path_getapppath	398
33.20.9.14path_getdefault	398
33.20.9.15path_getfilemoddate	398
33.20.9.16path_getmoddate	398
33.20.9.17path_nameconform	398
33.20.9.18path_openfolder	399
33.20.9.19path_opensysfile	399
33.20.9.20path_resolvefile	399

33.20.9.2 <code>lpath_setdefault</code>	400
33.20.9.22 <code>path_topathname</code>	400
33.20.9.23 <code>path_topotentialname</code>	400
33.20.9.24 <code>saveas_dialog</code>	401
33.20.9.25 <code>saveas_promptset</code>	401
33.20.9.26 <code>saveasdialog_extended</code>	402
33.20.9.27 <code>sysfile_close</code>	403
33.20.9.28 <code>sysfile_geteof</code>	403
33.20.9.29 <code>sysfile_getpos</code>	403
33.20.9.30 <code>sysfile_openhandle</code>	404
33.20.9.31 <code>sysfile_openptrsize</code>	404
33.20.9.32 <code>sysfile_read</code>	404
33.20.9.33 <code>sysfile_readtextfile</code>	405
33.20.9.34 <code>sysfile_readtohandle</code>	405
33.20.9.35 <code>sysfile_readtoptr</code>	405
33.20.9.36 <code>sysfile_seteof</code>	406
33.20.9.37 <code>sysfile_setpos</code>	406
33.20.9.38 <code>sysfile_spoolcopy</code>	406
33.20.9.39 <code>sysfile_write</code>	407
33.20.9.40 <code>sysfile_writetextfile</code>	407
33.21 <code>Jitter</code>	408
33.22 <code>Memory Management</code>	426
33.22.1 <code>Detailed Description</code>	427
33.22.2 <code>Sysmem API</code>	428
33.22.3 <code>Define Documentation</code>	428
33.22.3.1 <code>MM_UNIFIED</code>	428
33.22.4 <code>Function Documentation</code>	428
33.22.4.1 <code>disphandle</code>	428
33.22.4.2 <code>freebytes</code>	428
33.22.4.3 <code>freebytes16</code>	428
33.22.4.4 <code>getbytes</code>	429
33.22.4.5 <code>getbytes16</code>	429
33.22.4.6 <code>growhandle</code>	429
33.22.4.7 <code>newhandle</code>	430
33.22.4.8 <code>sysmem_copyptr</code>	430
33.22.4.9 <code>sysmem_freehandle</code>	430

33.22.4.10 <code>sysmem_freeptr</code>	430
33.22.4.11 <code>sysmem_handlesize</code>	431
33.22.4.12 <code>sysmem_lockhandle</code>	431
33.22.4.13 <code>sysmem_newhandle</code>	431
33.22.4.14 <code>sysmem_newhandleclear</code>	431
33.22.4.15 <code>sysmem_newptr</code>	432
33.22.4.16 <code>sysmem_newptrclear</code>	432
33.22.4.17 <code>sysmem_nullterminatehandle</code>	432
33.22.4.18 <code>sysmem_ptrandhand</code>	433
33.22.4.19 <code>sysmem_ptrbeforehand</code>	433
33.22.4.20 <code>sysmem_ptrsize</code>	433
33.22.4.21 <code>sysmem_resizehandle</code>	433
33.22.4.22 <code>sysmem_resizeptr</code>	434
33.22.4.23 <code>sysmem_resizeptrclear</code>	434
33.23 Miscellaneous	435
33.23.1 Define Documentation	438
33.23.1.1 <code>BEGIN_USING_C_LINKAGE</code>	438
33.23.1.2 <code>calcoffset</code>	438
33.23.1.3 <code>CLIP</code>	438
33.23.1.4 <code>InRange</code>	439
33.23.1.5 <code>MAX</code>	439
33.23.1.6 <code>MIN</code>	439
33.23.2 Enumeration Type Documentation	440
33.23.2.1 <code>e_max_errorcodes</code>	440
33.23.2.2 <code>e_max_wind_advise_result</code>	440
33.23.3 Function Documentation	440
33.23.3.1 <code>error_subscribe</code>	440
33.23.3.2 <code>error_sym</code>	441
33.23.3.3 <code>error_unsubscribe</code>	441
33.23.3.4 <code>globalsymbol_bind</code>	441
33.23.3.5 <code>globalsymbol_dereference</code>	441
33.23.3.6 <code>globalsymbol_reference</code>	442
33.23.3.7 <code>globalsymbol_unbind</code>	442
33.23.3.8 <code>maxversion</code>	442
33.23.3.9 <code>object_obex_quickref</code>	443
33.23.3.10 <code>post_sym</code>	443

33.23.3.1 <code>lquittask_install</code>	443
33.23.3.12 <code>lquittask_remove</code>	443
33.23.3.13 <code>sprintf_zero</code>	444
33.23.3.14 <code>strncat_zero</code>	444
33.23.3.15 <code>strncpy_zero</code>	444
33.23.3.16 <code>symbol_unique</code>	444
33.23.3.17 <code>symbolarray_sort</code>	445
33.23.3.18 <code>wind_advise</code>	445
33.23.3.19 <code>wind_setcursor</code>	445
33.24 <code>Console</code>	446
33.24.1 Function Documentation	446
33.24.1.1 <code>cpost</code>	446
33.24.1.2 <code>error</code>	447
33.24.1.3 <code>object_error</code>	447
33.24.1.4 <code>object_error_obtrusive</code>	448
33.24.1.5 <code>object_post</code>	448
33.24.1.6 <code>object_warn</code>	449
33.24.1.7 <code>ouchstring</code>	449
33.24.1.8 <code>post</code>	449
33.24.1.9 <code>postatom</code>	450
33.25 <code>Byte Ordering</code>	451
33.25.1 Detailed Description	452
33.25.2 Define Documentation	452
33.25.2.1 <code>BYTEORDER_LSBF32</code>	452
33.25.2.2 <code>BYTEORDER_LSBF64</code>	452
33.25.2.3 <code>BYTEORDER_LSBW16</code>	452
33.25.2.4 <code>BYTEORDER_LSBW32</code>	453
33.25.2.5 <code>BYTEORDER_MSBF32</code>	453
33.25.2.6 <code>BYTEORDER_MSBF64</code>	453
33.25.2.7 <code>BYTEORDER_MSBW16</code>	454
33.25.2.8 <code>BYTEORDER_MSBW32</code>	454
33.25.2.9 <code>BYTEORDER_SWAPF32</code>	454
33.25.2.10 <code>BYTEORDER_SWAPF64</code>	454
33.25.2.11 <code>BYTEORDER_SWAPW16</code>	455
33.25.2.12 <code>BYTEORDER_SWAPW32</code>	455
33.25.2.13 <code>C74_BIG_ENDIAN</code>	455

33.25.2.14C74_LITTLE_ENDIAN	455
33.26Extending expr	456
33.26.1 Detailed Description	457
33.26.2 Enumeration Type Documentation	457
33.26.2.1 e_max_expr_types	457
33.26.3 Function Documentation	458
33.26.3.1 expr_eval	458
33.26.3.2 expr_new	458
33.27Table Access	460
33.27.1 Detailed Description	460
33.27.2 Function Documentation	460
33.27.2.1 table_dirty	460
33.27.2.2 table_get	460
33.28Text Editor Windows	462
33.29Presets	463
33.29.1 Detailed Description	463
33.29.2 Function Documentation	464
33.29.2.1 preset_int	464
33.29.2.2 preset_set	464
33.29.2.3 preset_store	465
33.30Event and File Serial Numbers	466
33.30.1 Detailed Description	466
33.30.2 Using Event Serial Numbers	466
33.30.3 Function Documentation	466
33.30.3.1 evnum_get	466
33.30.3.2 serialno	467
33.31Loading Max Files	468
33.31.1 Detailed Description	468
33.31.2 Function Documentation	468
33.31.2.1 fileload	468
33.31.2.2 intload	469
33.31.2.3 readtohandle	469
33.31.2.4 stringload	469
33.32Monitors and Displays	471
33.32.1 Detailed Description	471
33.32.2 Function Documentation	471

33.32.2.1 jmonitor_getdisplayrect	471
33.32.2.2 jmonitor_getdisplayrect_foralldisplays	471
33.32.2.3 jmonitor_getdisplayrect_forpoint	472
33.32.2.4 jmonitor_getnumdisplays	472
33.33 Windows	473
33.33.1 Function Documentation	473
33.33.1.1 jwind_getactive	473
33.33.1.2 jwind_getat	473
33.33.1.3 jwind_getcount	473
33.34 Mouse and Keyboard	474
33.34.1 Enumeration Type Documentation	475
33.34.1.1 t_jmouse_cursortype	475
33.34.1.2 t_modifiers	476
33.34.2 Function Documentation	476
33.34.2.1 jkeyboard_getcurrentmodifiers	476
33.34.2.2 jkeyboard_getcurrentmodifiers_realtime	476
33.34.2.3 jmouse_getposition_global	476
33.34.2.4 jmouse_setcursor	477
33.34.2.5 jmouse_setposition_box	477
33.34.2.6 jmouse_setposition_global	477
33.34.2.7 jmouse_setposition_view	477
33.35 Example Projects	478
33.36 MSP	481
33.36.1 Define Documentation	483
33.36.1.1 PI	483
33.36.1.2 PIOVERTWO	483
33.36.1.3 TWOPI	483
33.36.2 Typedef Documentation	484
33.36.2.1 t_float	484
33.36.2.2 t_int	484
33.36.2.3 t_perfroutine	484
33.36.2.4 t_sample	484
33.36.2.5 t_vptr	484
33.36.2.6 vptr	484
33.36.3 Enumeration Type Documentation	484
33.36.3.1 "@20	484

33.36.4 Function Documentation	484
33.36.4.1 class_dspinit	484
33.36.4.2 class_dspinitjbox	485
33.36.4.3 dsp_add	485
33.36.4.4 dsp_addv	485
33.36.4.5 sys_getblksize	486
33.36.4.6 sys_getdspobjdspstate	486
33.36.4.7 sys_getdspstate	486
33.36.4.8 sys_getmaxblksize	486
33.36.4.9 sys_getsr	486
33.36.4.10z_dsp_free	487
33.36.4.11z_dsp_setup	487
33.37 Buffers	488
33.37.1 Detailed Description	488
33.38 PFFT	490
33.38.1 Detailed Description	490
33.39 Poly	491
33.40 Objects	492
33.40.1 Detailed Description	495
33.40.2 Define Documentation	495
33.40.2.1 MAXARG	495
33.40.3 Function Documentation	496
33.40.3.1 classname_openhelp	496
33.40.3.2 classname_openquery	496
33.40.3.3 classname_openrefpage	496
33.40.3.4 newobject_fromdictionary	496
33.40.3.5 newobject_sprintf	497
33.40.3.6 object_alloc	498
33.40.3.7 object_attach	498
33.40.3.8 object_attach_byptr	498
33.40.3.9 object_attach_byptr_register	499
33.40.3.10object_class	499
33.40.3.11object_classname	500
33.40.3.12object_classname_compare	500
33.40.3.13object_detach	500
33.40.3.14object_detach_byptr	501

33.40.3.15object_dictionaryarg	501
33.40.3.16object_findregistered	501
33.40.3.17object_findregisteredbyptr	502
33.40.3.18object_free	502
33.40.3.19object_getmethod	502
33.40.3.20object_getvalueof	502
33.40.3.21object_method	503
33.40.3.22object_method_char	504
33.40.3.23object_method_char_array	504
33.40.3.24object_method_double	505
33.40.3.25object_method_double_array	505
33.40.3.26object_method_float	505
33.40.3.27object_method_float_array	506
33.40.3.28object_method_format	506
33.40.3.29object_method_long	507
33.40.3.30object_method_long_array	507
33.40.3.31object_method_obj	507
33.40.3.32object_method_obj_array	508
33.40.3.33object_method_parse	508
33.40.3.34object_method_sym	509
33.40.3.35object_method_sym_array	509
33.40.3.36object_method_typed	509
33.40.3.37object_method_typedfun	510
33.40.3.38object_new	510
33.40.3.39object_new_typed	511
33.40.3.40object_notify	511
33.40.3.41object_obex_dumpout	512
33.40.3.42object_obex_lookup	512
33.40.3.43object_obex_store	513
33.40.3.44object_openhelp	513
33.40.3.45object_openquery	513
33.40.3.46object_openrefpage	513
33.40.3.47object_register	514
33.40.3.48object_setvalueof	514
33.40.3.49object_unregister	515
33.41 Patcher	516

33.41.1 Detailed Description	517
33.41.2 Typedef Documentation	517
33.41.2.1 t_box	517
33.41.2.2 t_patcher	517
33.41.3 Enumeration Type Documentation	517
33.41.3.1 "@3	517
33.42jpatcher	518
33.42.1 Detailed Description	520
33.42.2 Function Documentation	520
33.42.2.1 jpatcher_deleteobj	520
33.42.2.2 jpatcher_get_bgcolor	521
33.42.2.3 jpatcher_get_bghidden	521
33.42.2.4 jpatcher_get_bglocked	521
33.42.2.5 jpatcher_get_box	521
33.42.2.6 jpatcher_get_count	522
33.42.2.7 jpatcher_get_currentfileversion	522
33.42.2.8 jpatcher_get_default_fontface	522
33.42.2.9 jpatcher_get_default_fontname	522
33.42.2.10jpatcher_get_default_fontsize	522
33.42.2.11jpatcher_get_defrect	523
33.42.2.12jpatcher_get_dirty	523
33.42.2.13jpatcher_get_editing_bgcolor	523
33.42.2.14jpatcher_get_fghidden	524
33.42.2.15jpatcher_get_filename	524
33.42.2.16jpatcher_get_filepath	524
33.42.2.17jpatcher_get_fileversion	524
33.42.2.18jpatcher_get_firstline	525
33.42.2.19jpatcher_get_firstobject	525
33.42.2.20jpatcher_get_firstview	525
33.42.2.21jpatcher_get_grayscale	525
33.42.2.22jpatcher_get_lastobject	526
33.42.2.23jpatcher_get_name	526
33.42.2.24jpatcher_get_parentpatcher	526
33.42.2.25jpatcher_get_presentation	527
33.42.2.26jpatcher_get_rect	527
33.42.2.27jpatcher_get_title	527

33.42.2.28jpatcher_get_toppatcher	527
33.42.2.29jpatcher_is_patch	528
33.42.2.30jpatcher_set_bgcolor	528
33.42.2.31jpatcher_set_bghidden	528
33.42.2.32jpatcher_set_bglocked	528
33.42.2.33jpatcher_set_defrect	529
33.42.2.34jpatcher_set_dirty	529
33.42.2.35jpatcher_set_editing_bgcolor	529
33.42.2.36jpatcher_set_fghidden	529
33.42.2.37jpatcher_set_gridsize	530
33.42.2.38jpatcher_set_locked	530
33.42.2.39jpatcher_set_presentation	530
33.42.2.40jpatcher_set_rect	530
33.42.2.41jpatcher_set_title	531
33.42.2.42jpatcher_uniqueboxname	531
33.43jbox	532
33.43.1 Detailed Description	537
33.43.2 Define Documentation	538
33.43.2.1 JBOX_NOINSPECTFIRSTIN	538
33.43.3 Enumeration Type Documentation	538
33.43.3.1 "@17	538
33.43.3.2 HitTestResult	538
33.43.4 Function Documentation	538
33.43.4.1 jbox_free	538
33.43.4.2 jbox_get_annotation	539
33.43.4.3 jbox_get_background	539
33.43.4.4 jbox_get_canhilite	539
33.43.4.5 jbox_get_color	539
33.43.4.6 jbox_get_drawfirstin	540
33.43.4.7 jbox_get_drawinlast	540
33.43.4.8 jbox_get_fontname	540
33.43.4.9 jbox_get_fontsize	540
33.43.4.10jbox_get_growboth	541
33.43.4.11jbox_get_growy	541
33.43.4.12jbox_get_hidden	541
33.43.4.13jbox_get_hint	541

33.43.4.14jbox_get_hintstring	542
33.43.4.15jbox_get_id	542
33.43.4.16jbox_get_ignoreclick	542
33.43.4.17jbox_get_maxclass	542
33.43.4.18jbox_get_nextobject	543
33.43.4.19jbox_get_nogrow	543
33.43.4.20jbox_get_object	543
33.43.4.21jbox_get_outline	543
33.43.4.22jbox_get_patcher	544
33.43.4.23jbox_get_patching_position	544
33.43.4.24jbox_get_patching_rect	544
33.43.4.25jbox_get_patching_size	544
33.43.4.26jbox_get_presentation	545
33.43.4.27jbox_get_presentation_position	545
33.43.4.28jbox_get_presentation_rect	545
33.43.4.29jbox_get_presentation_size	545
33.43.4.30jbox_get_prevobject	546
33.43.4.31jbox_get_rect_for_sym	546
33.43.4.32jbox_get_rect_for_view	546
33.43.4.33jbox_get_textfield	546
33.43.4.34jbox_get_varname	547
33.43.4.35jbox_new	547
33.43.4.36jbox_notify	547
33.43.4.37jbox_ready	548
33.43.4.38jbox_redraw	548
33.43.4.39jbox_set_annotation	548
33.43.4.40jbox_set_background	548
33.43.4.41jbox_set_color	548
33.43.4.42jbox_set_fontname	549
33.43.4.43jbox_set fontsize	549
33.43.4.44jbox_set_hidden	549
33.43.4.45jbox_set_hint	549
33.43.4.46jbox_set_hintstring	550
33.43.4.47jbox_set_ignoreclick	550
33.43.4.48jbox_set_outline	550
33.43.4.49jbox_set_patching_position	551

33.43.4.50	box_set_patching_rect	551
33.43.4.51	lbox_set_patching_size	551
33.43.4.52	box_set_position	551
33.43.4.53	jbox_set_presentation	552
33.43.4.54	jbox_set_presentation_position	552
33.43.4.55	jbox_set_presentation_rect	552
33.43.4.56	jbox_set_presentation_size	552
33.43.4.57	jbox_set_rect	553
33.43.4.58	jbox_set_rect_for_sym	553
33.43.4.59	jbox_set_rect_for_view	553
33.43.4.60	jbox_set_size	553
33.43.4.61	lbox_set_varname	554
33.44	jpatchline	555
33.44.1	Detailed Description	556
33.44.2	Function Documentation	556
33.44.2.1	jpatchline_get_box1	556
33.44.2.2	jpatchline_get_box2	556
33.44.2.3	jpatchline_get_color	556
33.44.2.4	jpatchline_get_endpoint	556
33.44.2.5	jpatchline_get_hidden	557
33.44.2.6	jpatchline_get_inletnum	557
33.44.2.7	jpatchline_get_nextline	557
33.44.2.8	jpatchline_get_nummidpoints	557
33.44.2.9	jpatchline_get_outletnum	558
33.44.2.10	jpatchline_get_startpoint	558
33.44.2.11	lpatchline_set_color	558
33.44.2.12	jpatchline_set_hidden	558
33.45	jpatcherview	559
33.45.1	Detailed Description	560
33.45.2	Function Documentation	560
33.45.2.1	patcherview_findpatcherview	560
33.45.2.2	patcherview_get_jgraphics	560
33.45.2.3	patcherview_get_locked	560
33.45.2.4	patcherview_get_nextview	561
33.45.2.5	patcherview_get_patcher	561
33.45.2.6	patcherview_get_presentation	561

33.45.2.7 patcherview_get_rect	561
33.45.2.8 patcherview_get_topview	562
33.45.2.9 patcherview_get_visible	562
33.45.2.10 patcherview_get_zoomfactor	562
33.45.2.11 patcherview_set_locked	562
33.45.2.12 patcherview_set_presentation	563
33.45.2.13 patcherview_set_rect	563
33.45.2.14 patcherview_set_visible	563
33.45.2.15 patcherview_set_zoomfactor	563
33.46 Timing	564
33.47 Clocks	565
33.47.1 Detailed Description	566
33.47.2 Using Clocks	567
33.47.3 Scheduling with setclock Objects	568
33.47.3.1 Using the setclock Object Routines	568
33.47.4 Creating Schedulers	569
33.47.5 Function Documentation	569
33.47.5.1 clock_delay	569
33.47.5.2 clock_fdelay	569
33.47.5.3 clock_gettime	570
33.47.5.4 clock_new	570
33.47.5.5 clock_unset	570
33.47.5.6 gettime	570
33.47.5.7 scheduler_fromobject	571
33.47.5.8 scheduler_get	571
33.47.5.9 scheduler_gettime	571
33.47.5.10 scheduler_new	571
33.47.5.11 scheduler_run	572
33.47.5.12 scheduler_set	572
33.47.5.13 scheduler_settime	572
33.47.5.14 scheduler_shift	572
33.47.5.15 setclock_delay	573
33.47.5.16 setclock_fdelay	573
33.47.5.17 setclock_gettime	573
33.47.5.18 setclock_gettime	574
33.47.5.19 setclock_unset	574

33.47.5.20 <code>systimer_gettime</code>	574
33.48 <code>Qelems</code>	575
33.48.1 Detailed Description	575
33.48.2 Function Documentation	576
33.48.2.1 <code>qelem_free</code>	576
33.48.2.2 <code>qelem_front</code>	576
33.48.2.3 <code>qelem_new</code>	576
33.48.2.4 <code>qelem_set</code>	577
33.48.2.5 <code>qelem_unset</code>	577
33.49 <code>Systime API</code>	578
33.49.1 Detailed Description	579
33.49.2 Enumeration Type Documentation	579
33.49.2.1 <code>e_max_dateflags</code>	579
33.49.3 Function Documentation	579
33.49.3.1 <code>sysdateformat_formatdatetime</code>	579
33.49.3.2 <code>sysdateformat_strftimetodatetime</code>	579
33.49.3.3 <code>systime_datetime</code>	580
33.49.3.4 <code>systime_datetoseconds</code>	580
33.49.3.5 <code>systime_ms</code>	580
33.49.3.6 <code>systime_seconds</code>	580
33.49.3.7 <code>systime_secondstodate</code>	580
33.49.3.8 <code>systime_ticks</code>	580
33.50 <code>ITM Time Objects</code>	581
33.50.1 Detailed Description	584
33.50.2 Enumeration Type Documentation	584
33.50.2.1 <code>"@2</code>	584
33.50.3 Function Documentation	585
33.50.3.1 <code>class_time_addattr</code>	585
33.50.3.2 <code>itm_barbeatunitstoticks</code>	585
33.50.3.3 <code>itm_dereference</code>	585
33.50.3.4 <code>itm_dump</code>	586
33.50.3.5 <code>itm_getglobal</code>	586
33.50.3.6 <code>itm_getname</code>	586
33.50.3.7 <code>itm_getnamed</code>	586
33.50.3.8 <code>itm_getresolution</code>	586
33.50.3.9 <code>itm_getstate</code>	587

33.50.3.10 <code>tm_getticks</code>	587
33.50.3.11 <code>itm_gettime</code>	587
33.50.3.12 <code>itm_gettimesignature</code>	588
33.50.3.13 <code>itm_isunitfixed</code>	588
33.50.3.14 <code>itm_mstosamps</code>	588
33.50.3.15 <code>itm_mstoticks</code>	588
33.50.3.16 <code>itm_pause</code>	589
33.50.3.17 <code>itm_reference</code>	589
33.50.3.18 <code>itm_resume</code>	589
33.50.3.19 <code>itm_sampstoms</code>	589
33.50.3.20 <code>itm_setresolution</code>	589
33.50.3.21 <code>itm_settimesignature</code>	590
33.50.3.22 <code>itm_ticksbarbeatunits</code>	590
33.50.3.23 <code>itm_tickstoms</code>	590
33.50.3.24 <code>time_calcquantize</code>	591
33.50.3.25 <code>time_getitm</code>	591
33.50.3.26 <code>time_getms</code>	591
33.50.3.27 <code>time_getnamed</code>	591
33.50.3.28 <code>time_getphase</code>	592
33.50.3.29 <code>time_getticks</code>	592
33.50.3.30 <code>time_isfixedunit</code>	592
33.50.3.31 <code>time_listen</code>	592
33.50.3.32 <code>time_new</code>	593
33.50.3.33 <code>time_now</code>	593
33.50.3.34 <code>time_schedule</code>	593
33.50.3.35 <code>time_schedule_limit</code>	593
33.50.3.36 <code>time_setclock</code>	594
33.50.3.37 <code>time_setvalue</code>	594
33.50.3.38 <code>time_stop</code>	594
33.50.3.39 <code>time_tick</code>	594
33.50.4 Variable Documentation	594
33.50.4.1 <code>t_itm</code>	594
33.50.4.2 <code>t_timeobject</code>	595
33.51 Threads	596
33.51.1 Detailed Description	598
33.51.2 Define Documentation	598

33.51.2.1 ATOMIC_DECREMENT	598
33.51.2.2 ATOMIC_INCREMENT	598
33.51.3 Enumeration Type Documentation	598
33.51.3.1 e_max_systhread_mutex_flags	598
33.51.4 Function Documentation	598
33.51.4.1 defer	598
33.51.4.2 defer_low	599
33.51.4.3 isr	600
33.51.4.4 schedule	600
33.51.4.5 schedule_delay	601
33.51.4.6 systhread_create	601
33.51.4.7 systhread_exit	602
33.51.4.8 systhread_getpriority	602
33.51.4.9 systhread_ismainthread	602
33.51.4.10 systhread_istimerthread	602
33.51.4.11 systhread_join	603
33.51.4.12 systhread_self	603
33.51.4.13 systhread_setpriority	603
33.51.4.14 systhread_sleep	603
33.51.4.15 systhread_terminate	604
33.52 Critical Regions	605
33.52.1 Detailed Description	605
33.52.2 Function Documentation	606
33.52.2.1 critical_enter	606
33.52.2.2 critical_exit	607
33.52.2.3 critical_free	607
33.52.2.4 critical_new	607
33.52.2.5 critical_tryenter	607
33.53 Mutexes	609
33.53.1 Detailed Description	609
33.53.2 Function Documentation	609
33.53.2.1 systhread_mutex_free	609
33.53.2.2 systhread_mutex_lock	610
33.53.2.3 systhread_mutex_new	610
33.53.2.4 systhread_mutex_newlock	610
33.53.2.5 systhread_mutex_trylock	611

33.53.2.6 systhread_mutex_unlock	611
33.54 User Interface	612
33.55 JGraphics	613
33.55.1 Detailed Description	618
33.55.2 Define Documentation	618
33.55.2.1 JGRAPHICS_2PI	618
33.55.2.2 JGRAPHICS_3PIOVER2	618
33.55.2.3 JGRAPHICS_PI	618
33.55.2.4 JGRAPHICS_PIOVER2	618
33.55.3 Enumeration Type Documentation	618
33.55.3.1 t_jgraphics_fileformat	618
33.55.3.2 t_jgraphics_format	618
33.55.3.3 t_jgraphics_text_justification	619
33.55.4 Function Documentation	619
33.55.4.1 jgraphics_append_path	619
33.55.4.2 jgraphics_arc	619
33.55.4.3 jgraphics_arc_negative	620
33.55.4.4 jgraphics_close_path	620
33.55.4.5 jgraphics_copy_path	620
33.55.4.6 jgraphics_curve_to	620
33.55.4.7 jgraphics_destroy	621
33.55.4.8 jgraphics_device_to_user	621
33.55.4.9 jgraphics_ellipse	621
33.55.4.10 jgraphics_font_extents	621
33.55.4.11 jgraphics_get_current_point	621
33.55.4.12 jgraphics_getfiletypes	622
33.55.4.13 jgraphics_line_to	622
33.55.4.14 jgraphics_move_to	622
33.55.4.15 jgraphics_new_path	623
33.55.4.16 jgraphics_oval	623
33.55.4.17 jgraphics_ovalarc	623
33.55.4.18 jgraphics_path_destroy	624
33.55.4.19 jgraphics_path_roundcorners	624
33.55.4.20 jgraphics_position_one_rect_near_another_rect_but_keep_inside_a_- third_rect	624
33.55.4.21 jgraphics_rectangle	624
33.55.4.22 jgraphics_rectangle_rounded	625

33.55.4.23jgraphics_rectcontainsrect	625
33.55.4.24jgraphics_rectintersectsrect	625
33.55.4.25jgraphics_reference	625
33.55.4.26jgraphics_rel_curve_to	626
33.55.4.27jgraphics_rel_line_to	626
33.55.4.28jgraphics_rel_move_to	626
33.55.4.29jgraphics_round	626
33.55.4.30jgraphics_select_font_face	627
33.55.4.31jgraphics_select_jfont	627
33.55.4.32jgraphics_set_font_size	627
33.55.4.33jgraphics_set_underline	627
33.55.4.34jgraphics_show_text	627
33.55.4.35jgraphics_system_cana ntialia stexttotransparentbg	628
33.55.4.36jgraphics_text_measure	628
33.55.4.37jgraphics_text_measure_wrapped	628
33.55.4.38jgraphics_user_to_device	628
33.56JSurface	629
33.56.1 Detailed Description	630
33.56.2 Function Documentation	630
33.56.2.1 jgraphics_create	630
33.56.2.2 jgraphics_get_resource_data	631
33.56.2.3 jgraphics_image_surface_clear	631
33.56.2.4 jgraphics_image_surface_create	631
33.56.2.5 jgraphics_image_surface_create_for_data	632
33.56.2.6 jgraphics_image_surface_create_from_file	632
33.56.2.7 jgraphics_image_surface_create_from_filedata	632
33.56.2.8 jgraphics_image_surface_create_from_resource	633
33.56.2.9 jgraphics_image_surface_create_referenced	633
33.56.2.10jgraphics_image_surface_draw	633
33.56.2.11jgraphics_image_surface_draw_fast	634
33.56.2.12jgraphics_image_surface_get_height	634
33.56.2.13jgraphics_image_surface_get_pixel	634
33.56.2.14jgraphics_image_surface_get_width	635
33.56.2.15jgraphics_image_surface_scroll	635
33.56.2.16jgraphics_image_surface_set_pixel	635
33.56.2.17jgraphics_image_surface_writepng	635

33.56.2.18	graphics_surface_destroy	636
33.56.2.19	graphics_surface_reference	636
33.56.2.20	graphics_write_image_surface_to_filedata	636
33.57	Scalable Vector Graphics	638
33.57.1	Function Documentation	638
33.57.1.1	jsvg_create_from_file	638
33.57.1.2	jsvg_create_from_resource	638
33.57.1.3	jsvg_create_from_xmlstring	639
33.57.1.4	jsvg_destroy	639
33.57.1.5	jsvg_get_size	639
33.57.1.6	jsvg_render	639
33.58	JFont	640
33.58.1	Enumeration Type Documentation	641
33.58.1.1	t_jgraphics_font_slant	641
33.58.1.2	t_jgraphics_font_weight	641
33.58.2	Function Documentation	641
33.58.2.1	jbox_get_font_slant	641
33.58.2.2	jbox_get_font_weight	642
33.58.2.3	jfont_create	642
33.58.2.4	jfont_destroy	642
33.58.2.5	jfont_extents	642
33.58.2.6	jfont_getfontlist	643
33.58.2.7	jfont_reference	643
33.58.2.8	jfont_set_font_size	643
33.58.2.9	jfont_set_underline	643
33.58.2.10	font_text_measure	644
33.58.2.11	jfont_text_measure_wrapped	644
33.59	JGraphics Matrix Transformations	645
33.59.1	Detailed Description	646
33.59.2	Function Documentation	646
33.59.2.1	jgraphics_matrix_init	646
33.59.2.2	jgraphics_matrix_init_identity	646
33.59.2.3	jgraphics_matrix_init_rotate	646
33.59.2.4	jgraphics_matrix_init_scale	647
33.59.2.5	jgraphics_matrix_init_translate	647
33.59.2.6	jgraphics_matrix_invert	647

33.59.2.7 <code>jgraphics_matrix_multiply</code>	647
33.59.2.8 <code>jgraphics_matrix_rotate</code>	647
33.59.2.9 <code>jgraphics_matrix_scale</code>	648
33.59.2.10 <code>jgraphics_matrix_transform_point</code>	648
33.59.2.11 <code>jgraphics_matrix_translate</code>	648
33.60 <code>JPattern</code>	649
33.60.1 Detailed Description	649
33.61 <code>Colors</code>	650
33.61.1 Function Documentation	650
33.61.1.1 <code>atoms_to_jrgba</code>	650
33.61.1.2 <code>jrgba_attr_get</code>	651
33.61.1.3 <code>jrgba_attr_set</code>	651
33.61.1.4 <code>jrgba_compare</code>	651
33.61.1.5 <code>jrgba_copy</code>	652
33.61.1.6 <code>jrgba_set</code>	652
33.61.1.7 <code>jrgba_to_atoms</code>	652
33.62 <code>TextField</code>	653
33.62.1 Detailed Description	655
33.62.2 Function Documentation	655
33.62.2.1 <code>textfield_get_autoscroll</code>	655
33.62.2.2 <code>textfield_get_bgcolor</code>	655
33.62.2.3 <code>textfield_get_editonclick</code>	655
33.62.2.4 <code>textfield_get_emptytext</code>	655
33.62.2.5 <code>textfield_get_noactivate</code>	656
33.62.2.6 <code>textfield_get_owner</code>	656
33.62.2.7 <code>textfield_get_READONLY</code>	656
33.62.2.8 <code>textfield_get_selectallonedit</code>	656
33.62.2.9 <code>textfield_get_textcolor</code>	657
33.62.2.10 <code>textfield_get_textmargins</code>	657
33.62.2.11 <code>textfield_get_underline</code>	657
33.62.2.12 <code>textfield_get_useellipsis</code>	657
33.62.2.13 <code>textfield_get_wantsreturn</code>	658
33.62.2.14 <code>textfield_get_wantstab</code>	658
33.62.2.15 <code>textfield_get_wordwrap</code>	658
33.62.2.16 <code>textfield_set_autoscroll</code>	658
33.62.2.17 <code>textfield_set_bgcolor</code>	659

33.62.2.18 <code>textfield_set_editonclick</code>	659
33.62.2.19 <code>textfield_set_emptytext</code>	659
33.62.2.20 <code>textfield_set_noactivate</code>	659
33.62.2.21 <code>textfield_set_READONLY</code>	660
33.62.2.22 <code>textfield_set_selectallonedit</code>	660
33.62.2.23 <code>textfield_set_textcolor</code>	660
33.62.2.24 <code>textfield_set_textmargins</code>	660
33.62.2.25 <code>textfield_set_underline</code>	661
33.62.2.26 <code>textfield_set_useellipsis</code>	661
33.62.2.27 <code>textfield_set_wantsreturn</code>	661
33.62.2.28 <code>textfield_set_wantstab</code>	661
33.62.2.29 <code>textfield_set_wordwrap</code>	662
33.63 <code>TextLayout</code>	663
33.63.1 Detailed Description	664
33.63.2 Enumeration Type Documentation	664
33.63.2.1 <code>t_jgraphics_textlayout_flags</code>	664
33.63.3 Function Documentation	664
33.63.3.1 <code>jtextlayout_create</code>	664
33.63.3.2 <code>jtextlayout_destroy</code>	664
33.63.3.3 <code>jtextlayout_draw</code>	664
33.63.3.4 <code>jtextlayout_getchar</code>	665
33.63.3.5 <code>jtextlayout_getcharbox</code>	665
33.63.3.6 <code>jtextlayout_getnumchars</code>	665
33.63.3.7 <code>jtextlayout_measure</code>	665
33.63.3.8 <code>jtextlayout_set</code>	666
33.63.3.9 <code>jtextlayout_settextcolor</code>	666
33.63.3.10 <code>jtextlayout_withbgcolor</code>	666
33.64 <code>Popup Menus</code>	667
33.64.1 Detailed Description	668
33.64.2 Function Documentation	668
33.64.2.1 <code>jpopupmenu_additem</code>	668
33.64.2.2 <code>jpopupmenu_addseparator</code>	668
33.64.2.3 <code>jpopupmenu_addsubmenu</code>	668
33.64.2.4 <code>jpopupmenu_clear</code>	668
33.64.2.5 <code>jpopupmenu_create</code>	669
33.64.2.6 <code>jpopupmenu_destroy</code>	669

33.64.2.7 <code>jpopupmenu_popup</code>	669
33.64.2.8 <code>jpopupmenu_popup_abovebox</code>	669
33.64.2.9 <code>jpopupmenu_popup_nearbox</code>	670
33.64.2.10 <code>jpopupmenu_setcolors</code>	670
33.64.2.11 <code>lpopupmenu_setfont</code>	670
33.65 Box Layer	671
33.65.1 Detailed Description	671
33.65.2 Function Documentation	672
33.65.2.1 <code>jbox_end_layer</code>	672
33.65.2.2 <code>jbox_invalidate_layer</code>	672
33.65.2.3 <code>jbox_paint_layer</code>	672
33.65.2.4 <code>jbox_start_layer</code>	673
33.66 DataView	674
33.66.1 Detailed Description	674
33.66.2 Function Documentation	674
33.66.2.1 <code>jdataview_getclient</code>	674
33.66.2.2 <code>jdataview_new</code>	675
33.66.2.3 <code>jdataview_setclient</code>	675
33.67 Unicode	676
33.67.1 Detailed Description	676
33.67.2 Character Encodings	676
33.67.2.1 Example Usage	677
33.67.3 Function Documentation	677
33.67.3.1 <code>charset_convert</code>	677
33.67.3.2 <code>charset_unicodetouf8</code>	677
33.67.3.3 <code>charset_utf8_count</code>	678
33.67.3.4 <code>charset_utf8_offset</code>	678
33.67.3.5 <code>charset_utf8tounicode</code>	678
33.68 Atom Module	679
33.68.1 Function Documentation	680
33.68.1.1 <code>jit_atom_arg_getdouble</code>	680
33.68.1.2 <code>jit_atom_arg_getfloat</code>	680
33.68.1.3 <code>jit_atom_arg_getlong</code>	680
33.68.1.4 <code>jit_atom_arg_getsym</code>	681
33.68.1.5 <code>jit_atom_getcharfix</code>	681
33.68.1.6 <code>jit_atom_getfloat</code>	681

33.68.1.7 jit_atom_getlong	681
33.68.1.8 jit_atom_getobj	682
33.68.1.9 jit_atom_getsym	682
33.68.1.10 jit_atom_setfloat	682
33.68.1.11 jit_atom_setlong	682
33.68.1.12 jit_atom_setobj	683
33.68.1.13 jit_atom_setsym	683
33.69 Attribute Module	684
33.69.1 Function Documentation	686
33.69.1.1 jit_attr_caget	686
33.69.1.2 jit_attr_canset	687
33.69.1.3 jit_attr_filter_clip_new	687
33.69.1.4 jit_attr_filter_proc_new	687
33.69.1.5 jit_attr_filterget	687
33.69.1.6 jit_attr_filterset	688
33.69.1.7 jit_attr_get	688
33.69.1.8 jit_attr_getchar_array	689
33.69.1.9 jit_attr_getdouble_array	689
33.69.1.10 jit_attr_getfloat	689
33.69.1.11 jit_attr_getfloat_array	689
33.69.1.12 jit_attr_getlong	690
33.69.1.13 jit_attr_getlong_array	690
33.69.1.14 jit_attr_getmethod	690
33.69.1.15 jit_attr_getname	691
33.69.1.16 jit_attr_getsym	691
33.69.1.17 jit_attr_getsym_array	691
33.69.1.18 jit_attr_gettime	691
33.69.1.19 jit_attr_offset_array_new	692
33.69.1.20 jit_attr_offset_new	692
33.69.1.21 jit_attr_set	693
33.69.1.22 jit_attr_setchar_array	693
33.69.1.23 jit_attr_setdouble_array	693
33.69.1.24 jit_attr_setfloat	694
33.69.1.25 jit_attr_setfloat_array	694
33.69.1.26 jit_attr_setlong	694
33.69.1.27 jit_attr_setlong_array	694

33.69.1.28jit_attr_setsym	695
33.69.1.29jit_attr_setsym_array	695
33.69.1.30jit_attr_symcompare	695
33.69.1.31jit_attr_usercanget	696
33.69.1.32jit_attr_usercanset	696
33.69.1.33jit_attribute_new	696
33.70Binary Module	697
33.70.1 Function Documentation	697
33.70.1.1 jit_bin_read_chunk_info	697
33.70.1.2 jit_bin_read_header	697
33.70.1.3 jit_bin_read_matrix	698
33.70.1.4 jit_bin_write_header	698
33.70.1.5 jit_bin_write_matrix	698
33.71Class Module	699
33.71.1 Function Documentation	700
33.71.1.1 jit_class_adddornment	700
33.71.1.2 jit_class_addattr	700
33.71.1.3 jit_class_addmethod	700
33.71.1.4 jit_class_addtypedwrapper	701
33.71.1.5 jit_class_adornment_get	701
33.71.1.6 jit_class_attr_get	701
33.71.1.7 jit_class_findbyname	702
33.71.1.8 jit_class_free	702
33.71.1.9 jit_class_mess	702
33.71.1.10 jit_class_method	702
33.71.1.11 jit_class_method_addargsafe	703
33.71.1.12 jit_class_method_argsafe_get	703
33.71.1.13 jit_class_nameget	703
33.71.1.14 jit_class_new	703
33.71.1.15 jit_class_register	704
33.71.1.16 jit_class_symcompare	704
33.71.1.17 jit_class_typedwrapper_get	704
33.72Object Module	705
33.72.1 Function Documentation	706
33.72.1.1 jit_object_attach	706
33.72.1.2 jit_object_attr_get	706

33.72.1.3 jit_object_attr_usercanget	707
33.72.1.4 jit_object_attr_usercanset	707
33.72.1.5 jit_object_class	707
33.72.1.6 jit_object_classname	707
33.72.1.7 jit_object_classname_compare	708
33.72.1.8 jit_object_detach	708
33.72.1.9 jit_object_exportattrs	708
33.72.1.10 jit_object_exportsummary	708
33.72.1.11 jit_object_findregistered	709
33.72.1.12 jit_object_findregisteredbyptr	709
33.72.1.13 jit_object_free	709
33.72.1.14 jit_object_getmethod	709
33.72.1.15 jit_object_importattrs	710
33.72.1.16 jit_object_method	710
33.72.1.17 jit_object_method_argsafe_get	710
33.72.1.18 jit_object_method_typed	711
33.72.1.19 jit_object_new	711
33.72.1.20 jit_object_notify	711
33.72.1.21 jit_object_register	712
33.72.1.22 jit_object_unregister	712
33.73 Miscellaneous Utility Module	713
33.73.1 Function Documentation	713
33.73.1.1 jit_err_from_max_err	713
33.73.1.2 jit_error_code	714
33.73.1.3 jit_error_sym	714
33.73.1.4 jit_global_critical_enter	714
33.73.1.5 jit_global_critical_exit	714
33.73.1.6 jit_post_sym	714
33.73.1.7 jit_rand	715
33.73.1.8 jit_rand_setseed	715
33.73.1.9 swapf32	715
33.73.1.10 swapf64	715
33.74 Linked List Module	716
33.74.1 Function Documentation	717
33.74.1.1 jit_linklist_append	717
33.74.1.2 jit_linklist_chuck	718

33.74.1.3 jit_linklist_chuckindex	718
33.74.1.4 jit_linklist_clear	718
33.74.1.5 jit_linklist_deleteindex	719
33.74.1.6 jit_linklist_findall	719
33.74.1.7 jit_linklist_findcount	719
33.74.1.8 jit_linklist_findfirst	720
33.74.1.9 jit_linklist_getindex	720
33.74.1.10 jit_linklist_getsize	721
33.74.1.11 jit_linklist_insertindex	721
33.74.1.12 jit_linklist_makearray	721
33.74.1.13 jit_linklist_methodall	722
33.74.1.14 jit_linklist_methodindex	722
33.74.1.15 jit_linklist_new	723
33.74.1.16 jit_linklist_objptr2index	723
33.74.1.17 jit_linklist_reverse	723
33.74.1.18 jit_linklist_rotate	723
33.74.1.19 jit_linklist_shuffle	724
33.74.1.20 jit_linklist_sort	724
33.74.1.21 jit_linklist_swap	724
33.75 Math Module	726
33.75.1 Function Documentation	729
33.75.1.1 jit_math_acos	729
33.75.1.2 jit_math_acosh	729
33.75.1.3 jit_math_asin	729
33.75.1.4 jit_math_asinh	729
33.75.1.5 jit_math_atan	730
33.75.1.6 jit_math_atan2	730
33.75.1.7 jit_math_atanh	730
33.75.1.8 jit_math_ceil	730
33.75.1.9 jit_math_cos	731
33.75.1.10 jit_math_cosh	731
33.75.1.11 jit_math_exp	731
33.75.1.12 jit_math_exp2	731
33.75.1.13 jit_math_expm1	732
33.75.1.14 jit_math_fast_acos	732
33.75.1.15 jit_math_fast_asin	732

33.75.1.16jit_math_fast_atan	732
33.75.1.17jit_math_fast_cos	733
33.75.1.18jit_math_fast_invsqrt	733
33.75.1.19jit_math_fast_sin	733
33.75.1.20jit_math_fast_sqrt	733
33.75.1.21jit_math_fast_tan	734
33.75.1.22jit_math_floor	734
33.75.1.23jit_math_fmod	734
33.75.1.24jit_math_fold	734
33.75.1.25jit_math_hypot	735
33.75.1.26jit_math_is_finite	735
33.75.1.27jit_math_is_nan	735
33.75.1.28jit_math_is_poweroftwo	735
33.75.1.29jit_math_is_valid	736
33.75.1.30jit_math_j1	736
33.75.1.31jit_math_j1_0	736
33.75.1.32jit_math_log	736
33.75.1.33jit_math_log10	737
33.75.1.34jit_math_log2	737
33.75.1.35jit_math_p1	737
33.75.1.36jit_math_pow	737
33.75.1.37jit_math_q1	738
33.75.1.38jit_math_round	738
33.75.1.39jit_math_roundup_poweroftwo	738
33.75.1.40jit_math_sin	738
33.75.1.41jit_math_sinh	739
33.75.1.42jit_math_sqrt	739
33.75.1.43jit_math_tan	739
33.75.1.44jit_math_tanh	739
33.75.1.45jit_math_trunc	740
33.75.1.46jit_math_wrap	740
33.76Matrix Module	741
33.76.1 Function Documentation	743
33.76.1.1 jit_linklist_free	743
33.76.1.2 jit_matrix_clear	743
33.76.1.3 jit_matrix_data	743

33.76.1.4 jit_matrix_exprfill	744
33.76.1.5 jit_matrix_fillplane	744
33.76.1.6 jit_matrix_free	744
33.76.1.7 jit_matrix_freedata	745
33.76.1.8 jit_matrix_fromgworld	745
33.76.1.9 jit_matrix_frommatrix	745
33.76.1.10 jit_matrix_getcell	746
33.76.1.11 jit_matrix_getdata	746
33.76.1.12 jit_matrix_getinfo	747
33.76.1.13 jit_matrix_info_default	747
33.76.1.14 jit_matrix_jit_gl_texture	747
33.76.1.15 jit_matrix_new	748
33.76.1.16 jit_matrix_newcopy	748
33.76.1.17 jit_matrix_op	748
33.76.1.18 jit_matrix_setall	749
33.76.1.19 jit_matrix_setcell	749
33.76.1.20 jit_matrix_setcell1d	749
33.76.1.21 jit_matrix_setcell2d	750
33.76.1.22 jit_matrix_setcell3d	750
33.76.1.23 jit_matrix_setinfo	751
33.76.1.24 jit_matrix_setinfo_ex	751
33.76.1.25 jit_matrix_setplane1d	751
33.76.1.26 jit_matrix_setplane2d	752
33.76.1.27 jit_matrix_setplane3d	752
33.76.1.28 jit_matrix_togworld	753
33.77 Max Wrapper Module	754
33.77.1 Function Documentation	755
33.77.1.1 max_addmethod_defer	755
33.77.1.2 max_addmethod_defer_low	756
33.77.1.3 max_addmethod_usurp	756
33.77.1.4 max_addmethod_usurp_low	756
33.77.1.5 max_jit_attr_args	756
33.77.1.6 max_jit_attr_args_offset	757
33.77.1.7 max_jit_attr_get	757
33.77.1.8 max_jit_attr_getdump	757
33.77.1.9 max_jit_attr_set	757

33.77.1.10max_jit_classex_addattr	758
33.77.1.1lmax_jit_classex_setup	758
33.77.1.12max_jit_classex_standard_wrap	758
33.77.1.13max_jit_obex_adornment_get	758
33.77.1.14max_jit_obex_attr_get	759
33.77.1.15max_jit_obex_attr_set	759
33.77.1.16max_jit_obex_dumpout	759
33.77.1.17max_jit_obex_dumpout_get	759
33.77.1.18max_jit_obex_dumpout_set	760
33.77.1.19max_jit_obex_free	760
33.77.1.20max_jit_obex_gimmeback	760
33.77.1.21lmax_jit_obex_gimmeback_dumpout	760
33.77.1.22max_jit_obex_inletnumber_get	761
33.77.1.23max_jit_obex_inletnumber_set	761
33.77.1.24max_jit_obex_jitob_get	761
33.77.1.25max_jit_obex_jitob_set	761
33.77.1.26max_jit_obex_new	762
33.77.1.27max_jit_obex_proxy_new	762
33.78Memory Module	763
33.78.1 Function Documentation	763
33.78.1.1 jit_copy_bytes	763
33.78.1.2 jit_disposeptr	764
33.78.1.3 jit_freebytes	764
33.78.1.4 jit_freemem	764
33.78.1.5 jit_getbytes	765
33.78.1.6 jit_handle_free	765
33.78.1.7 jit_handle_lock	765
33.78.1.8 jit_handle_new	765
33.78.1.9 jit_handle_size_get	766
33.78.1.10jit_handle_size_set	766
33.78.1.11ljit_newptr	766
33.79MOP Module	767
33.79.1 Function Documentation	769
33.79.1.1 jit_mop_free	769
33.79.1.2 jit_mop_getinput	769
33.79.1.3 jit_mop_getinputlist	769

33.79.1.4 jit_mop_getoutput	770
33.79.1.5 jit_mop_getoutputlist	770
33.79.1.6 jit_mop_input_nolink	770
33.79.1.7 jit_mop_io_free	771
33.79.1.8 jit_mop_io_getioproc	771
33.79.1.9 jit_mop_io_getmatrix	771
33.79.1.10 jit_mop_io_ioproc	772
33.79.1.11 jit_mop_io_matrix	772
33.79.1.12 jit_mop_io_new	772
33.79.1.13 jit_mop_io_newcopy	773
33.79.1.14 jit_mop_io_restrict_dim	773
33.79.1.15 jit_mop_io_restrict_planecount	773
33.79.1.16 jit_mop_io_restrict_type	774
33.79.1.17 jit_mop_ioproc_copy_adapt	774
33.79.1.18 jit_mop_ioproc_copy_trunc	774
33.79.1.19 jit_mop_ioproc_copy_trunc_zero	775
33.79.1.20 jit_mop_ioproc_tosym	775
33.79.1.21 jit_mop_methodall	776
33.79.1.22 jit_mop_new	776
33.79.1.23 jit_mop_newcopy	776
33.79.1.24 jit_mop_output_nolink	777
33.79.1.25 jit_mop_single_planecount	777
33.79.1.26 jit_mop_single_type	777
33.80 Parallel Utility Module	778
33.80.1 Function Documentation	778
33.80.1.1 jit_parallel_ndim_calc	778
33.80.1.2 jit_parallel_ndim_simplecalc1	779
33.80.1.3 jit_parallel_ndim_simplecalc2	779
33.80.1.4 jit_parallel_ndim_simplecalc3	780
33.80.1.5 jit_parallel_ndim_simplecalc4	780
33.81 MOP Max Wrapper Module	782
33.81.1 Function Documentation	783
33.81.1.1 max_jit_classex_mop_mproc	783
33.81.1.2 max_jit_classex_mop_wrap	783
33.81.1.3 max_jit_mop_adapt_matrix_all	784
33.81.1.4 max_jit_mop_assist	784

33.81.1.5 max_jit_mop_bang	784
33.81.1.6 max_jit_mop_clear	785
33.81.1.7 max_jit_mop_free	785
33.81.1.8 max_jit_mop_get_io_by_name	785
33.81.1.9 max_jit_mop_getinput	785
33.81.1.10max_jit_mop_getoutput	786
33.81.1.11max_jit_mop_getoutputmode	786
33.81.1.12max_jit_mop_inputs	786
33.81.1.13max_jit_mop_jit_matrix	786
33.81.1.14max_jit_mop_matrix_args	787
33.81.1.15max_jit_mop_matrixout_new	787
33.81.1.16max_jit_mop_notify	787
33.81.1.17max_jit_mop_outputmatrix	787
33.81.1.18max_jit_mop_outputs	788
33.81.1.19max_jit_mop_setup	788
33.81.1.20max_jit_mop_setup_simple	788
33.81.1.21max_jit_mop_variable_addinputs	789
33.81.1.22max_jit_mop_variable_addoutputs	789
33.82OB3D Module	790
33.82.1 Function Documentation	793
33.82.1.1 jit_gl_begincapture	793
33.82.1.2 jit_gl_bindtexture	793
33.82.1.3 jit_gl_drawinfo_active_textures	793
33.82.1.4 jit_gl_drawinfo_setup	793
33.82.1.5 jit_gl_endcapture	794
33.82.1.6 jit_gl_get_extensions	794
33.82.1.7 jit_gl_get_glu_version	794
33.82.1.8 jit_gl_get_renderer	794
33.82.1.9 jit_gl_get_vendor	794
33.82.1.10jit_gl_get_version	795
33.82.1.11jit_gl_is_extension_supported	795
33.82.1.12jit_gl_is_min_version	795
33.82.1.13jit_gl_report_error	795
33.82.1.14jit_gl_texcoord1f	796
33.82.1.15jit_gl_texcoord1fv	796
33.82.1.16jit_gl_texcoord2f	796

33.82.1.17	jit_gl_texcoord2fv	796
33.82.1.18	jit_gl_texcoord3f	796
33.82.1.19	jit_gl_texcoord3fv	797
33.82.1.20	jit_gl_unbindtexture	797
33.82.1.21	jit_glchunk_copy	797
33.82.1.22	jit_glchunk_delete	797
33.82.1.23	jit_glchunk_grid_new	798
33.82.1.24	jit_glchunk_new	798
33.82.1.25	jit_ob3d_draw_chunk	798
33.82.1.26	jit_ob3d_free	798
33.82.1.27	jit_ob3d_new	799
33.82.1.28	jit_ob3d_set_context	799
33.82.1.29	jit_ob3d_setup	799
33.82.1.30	max_jit_ob3d_assist	799
33.82.1.31	max_jit_ob3d_attach	800
33.82.1.32	max_jit_ob3d_detach	800
33.82.1.33	max_ob3d_bang	800
33.82.1.34	max_ob3d_notify	800
33.82.1.35	ob3d_auto_get	801
33.82.1.36	ob3d_dest_dim_get	801
33.82.1.37	ob3d_dest_dim_set	801
33.82.1.38	ob3d_dirty_get	801
33.82.1.39	ob3d_dirty_set	801
33.82.1.40	ob3d_enable_get	802
33.82.1.41	ob3d jitob_get	802
33.82.1.42	ob3d_outlet_get	802
33.82.1.43	ob3d_render_ptr_get	802
33.82.1.44	ob3d_render_ptr_set	803
33.82.1.45	ob3d_ui_get	803
33.83	Operator Vector Module	804
33.83.1	Function Documentation	818
33.83.1.1	jit_op_vector_abs_float32	818
33.83.1.2	jit_op_vector_abs_float64	819
33.83.1.3	jit_op_vector_abs_long	819
33.83.1.4	jit_op_vector_absdiff_char	819
33.83.1.5	jit_op_vector_absdiff_float32	820

33.83.1.6 jit_op_vector_absdiff_float64	820
33.83.1.7 jit_op_vector_absdiff_long	820
33.83.1.8 jit_op_vector_acos_float32	820
33.83.1.9 jit_op_vector_acos_float64	821
33.83.1.10 jit_op_vector_acosh_float32	821
33.83.1.11 jit_op_vector_acosh_float64	821
33.83.1.12 jit_op_vector_add_char	822
33.83.1.13 jit_op_vector_add_float32	822
33.83.1.14 jit_op_vector_add_float64	822
33.83.1.15 jit_op_vector_add_long	822
33.83.1.16 jit_op_vector_adds_char	823
33.83.1.17 jit_op_vector_and_char	823
33.83.1.18 jit_op_vector_and_float32	823
33.83.1.19 jit_op_vector_and_float64	824
33.83.1.20 jit_op_vector_and_long	824
33.83.1.21 jit_op_vector_asin_float32	824
33.83.1.22 jit_op_vector_asin_float64	824
33.83.1.23 jit_op_vector_asinh_float32	825
33.83.1.24 jit_op_vector_asinh_float64	825
33.83.1.25 jit_op_vector_atan2_float32	825
33.83.1.26 jit_op_vector_atan2_float64	826
33.83.1.27 jit_op_vector_atan_float32	826
33.83.1.28 jit_op_vector_atan_float64	826
33.83.1.29 jit_op_vector_atanh_float32	826
33.83.1.30 jit_op_vector_atanh_float64	827
33.83.1.31 jit_op_vector_avg_char	827
33.83.1.32 jit_op_vector_avg_float32	827
33.83.1.33 jit_op_vector_avg_float64	828
33.83.1.34 jit_op_vector_avg_long	828
33.83.1.35 jit_op_vector_bitand_char	828
33.83.1.36 jit_op_vector_bitand_long	828
33.83.1.37 jit_op_vector_bitnot_char	829
33.83.1.38 jit_op_vector_bitnot_long	829
33.83.1.39 jit_op_vector_bitor_char	829
33.83.1.40 jit_op_vector_bitor_long	830
33.83.1.41 jit_op_vector_bitxor_char	830

33.83.1.42jit_op_vector_bitxor_long	830
33.83.1.43jit_op_vector_ceil_float32	830
33.83.1.44jit_op_vector_ceil_float64	831
33.83.1.45jit_op_vector_cos_float32	831
33.83.1.46jit_op_vector_cos_float64	831
33.83.1.47jit_op_vector_cosh_float32	832
33.83.1.48jit_op_vector_cosh_float64	832
33.83.1.49jit_op_vector_div_char	832
33.83.1.50jit_op_vector_div_float32	832
33.83.1.51jit_op_vector_div_float64	833
33.83.1.52jit_op_vector_div_long	833
33.83.1.53jit_op_vector_eq_char	833
33.83.1.54jit_op_vector_eq_float32	834
33.83.1.55jit_op_vector_eq_float64	834
33.83.1.56jit_op_vector_eq_long	834
33.83.1.57jit_op_vector_eqp_char	834
33.83.1.58jit_op_vector_eqp_float32	835
33.83.1.59jit_op_vector_eqp_float64	835
33.83.1.60jit_op_vector_eqp_long	835
33.83.1.61jit_op_vector_exp2_float32	836
33.83.1.62jit_op_vector_exp2_float64	836
33.83.1.63jit_op_vector_exp_float32	836
33.83.1.64jit_op_vector_exp_float64	836
33.83.1.65jit_op_vector_flipdiv_char	837
33.83.1.66jit_op_vector_flipdiv_float32	837
33.83.1.67jit_op_vector_flipdiv_float64	837
33.83.1.68jit_op_vector_flipdiv_long	838
33.83.1.69jit_op_vector_flipmod_char	838
33.83.1.70jit_op_vector_flipmod_float32	838
33.83.1.71jit_op_vector_flipmod_float64	838
33.83.1.72jit_op_vector_flipmod_long	839
33.83.1.73jit_op_vector_flippass_char	839
33.83.1.74jit_op_vector_flippass_float32	839
33.83.1.75jit_op_vector_flippass_float64	840
33.83.1.76jit_op_vector_flippass_long	840
33.83.1.77jit_op_vector_flipsub_char	840

33.83.1.7 ⁸ jit_op_vector_flipsub_float32	840
33.83.1.7 ⁹ jit_op_vector_flipsub_long	841
33.83.1.8 ⁰ jit_op_vector_floor_float32	841
33.83.1.8 ¹ jit_op_vector_floor_float64	841
33.83.1.8 ² jit_op_vector_fold_float32	842
33.83.1.8 ³ jit_op_vector_fold_float64	842
33.83.1.8 ⁴ jit_op_vector_gt_char	842
33.83.1.8 ⁵ jit_op_vector_gt_float32	842
33.83.1.8 ⁶ jit_op_vector_gt_float64	843
33.83.1.8 ⁷ jit_op_vector_gt_long	843
33.83.1.8 ⁸ jit_op_vector_gte_char	843
33.83.1.8 ⁹ jit_op_vector_gte_float32	844
33.83.1.9 ⁰ jit_op_vector_gte_float64	844
33.83.1.9 ¹ jit_op_vector_gte_long	844
33.83.1.9 ² jit_op_vector_gtep_char	844
33.83.1.9 ³ jit_op_vector_gtep_float32	845
33.83.1.9 ⁴ jit_op_vector_gtep_float64	845
33.83.1.9 ⁵ jit_op_vector_gtep_long	845
33.83.1.9 ⁶ jit_op_vector_gtp_char	846
33.83.1.9 ⁷ jit_op_vector_gtp_float32	846
33.83.1.9 ⁸ jit_op_vector_gtp_float64	846
33.83.1.9 ⁹ jit_op_vector_gtp_long	846
33.83.1.10 ⁰ jit_op_vector_hypot_float32	847
33.83.1.10 ¹ jit_op_vector_hypot_float64	847
33.83.1.10 ² jit_op_vector_log10_float32	847
33.83.1.10 ³ jit_op_vector_log10_float64	848
33.83.1.10 ⁴ jit_op_vector_log2_float32	848
33.83.1.10 ⁵ jit_op_vector_log2_float64	848
33.83.1.10 ⁶ jit_op_vector_log_float32	848
33.83.1.10 ⁷ jit_op_vector_log_float64	849
33.83.1.10 ⁸ jit_op_vector_lshift_char	849
33.83.1.10 ⁹ jit_op_vector_lshift_long	849
33.83.1.11 ⁰ jit_op_vector_lt_char	850
33.83.1.11 ¹ jit_op_vector_lt_float32	850
33.83.1.11 ² jit_op_vector_lt_float64	850
33.83.1.11 ³ jit_op_vector_lt_long	850

33.83.1.1 <code>jl_t_op_vector_lte_char</code>	851
33.83.1.1 <code>jl_t_op_vector_lte_float32</code>	851
33.83.1.1 <code>jl_t_op_vector_lte_float64</code>	851
33.83.1.1 <code>jl_t_op_vector_lte_long</code>	852
33.83.1.1 <code>jl_t_op_vector_ltep_char</code>	852
33.83.1.1 <code>jl_t_op_vector_ltep_float32</code>	852
33.83.1.1 <code>jl_t_op_vector_ltep_float64</code>	852
33.83.1.1 <code>jl_t_op_vector_ltep_long</code>	853
33.83.1.1 <code>jl_t_op_vector_ltp_char</code>	853
33.83.1.1 <code>jl_t_op_vector_ltp_float32</code>	853
33.83.1.1 <code>jl_t_op_vector_ltp_float64</code>	854
33.83.1.1 <code>jl_t_op_vector_ltp_long</code>	854
33.83.1.1 <code>jl_t_op_vector_max_char</code>	854
33.83.1.1 <code>jl_t_op_vector_max_float32</code>	854
33.83.1.1 <code>jl_t_op_vector_max_float64</code>	855
33.83.1.1 <code>jl_t_op_vector_max_long</code>	855
33.83.1.1 <code>jl_t_op_vector_min_char</code>	855
33.83.1.1 <code>jl_t_op_vector_min_float32</code>	856
33.83.1.1 <code>jl_t_op_vector_min_float64</code>	856
33.83.1.1 <code>jl_t_op_vector_min_long</code>	856
33.83.1.1 <code>jl_t_op_vector_mod_char</code>	856
33.83.1.1 <code>jl_t_op_vector_mod_float32</code>	857
33.83.1.1 <code>jl_t_op_vector_mod_float64</code>	857
33.83.1.1 <code>jl_t_op_vector_mod_long</code>	857
33.83.1.1 <code>jl_t_op_vector_mult_char</code>	858
33.83.1.1 <code>jl_t_op_vector_mult_float32</code>	858
33.83.1.1 <code>jl_t_op_vector_mult_float64</code>	858
33.83.1.1 <code>jl_t_op_vector_mult_long</code>	858
33.83.1.1 <code>jl_t_op_vector_neq_char</code>	859
33.83.1.1 <code>jl_t_op_vector_neq_float32</code>	859
33.83.1.1 <code>jl_t_op_vector_neq_float64</code>	859
33.83.1.1 <code>jl_t_op_vector_neq_long</code>	860
33.83.1.1 <code>jl_t_op_vector_neqp_char</code>	860
33.83.1.1 <code>jl_t_op_vector_neqp_float32</code>	860
33.83.1.1 <code>jl_t_op_vector_neqp_float64</code>	860
33.83.1.1 <code>jl_t_op_vector_neqp_long</code>	861

33.83.1.150 lt _op_vector_not_char	861
33.83.1.151 lt _op_vector_not_float32	861
33.83.1.152 lt _op_vector_not_float64	862
33.83.1.153 lt _op_vector_not_long	862
33.83.1.154 lt _op_vector_or_char	862
33.83.1.155 lt _op_vector_or_float32	862
33.83.1.156 lt _op_vector_or_float64	863
33.83.1.157 lt _op_vector_or_long	863
33.83.1.158 lt _op_vector_pass_char	863
33.83.1.159 lt _op_vector_pass_float32	864
33.83.1.160 lt _op_vector_pass_float64	864
33.83.1.161 lt _op_vector_pass_long	864
33.83.1.162 lt _op_vector_pow_float32	864
33.83.1.163 lt _op_vector_pow_float64	865
33.83.1.164 lt _op_vector_round_float32	865
33.83.1.165 lt _op_vector_round_float64	865
33.83.1.166 lt _op_vector_rshift_char	866
33.83.1.167 lt _op_vector_rshift_long	866
33.83.1.168 lt _op_vector_sin_float32	866
33.83.1.169 lt _op_vector_sin_float64	866
33.83.1.170 lt _op_vector_sinh_float32	867
33.83.1.171 lt _op_vector_sinh_float64	867
33.83.1.172 lt _op_vector_sqrt_float32	867
33.83.1.173 lt _op_vector_sqrt_float64	868
33.83.1.174 lt _op_vector_sub_char	868
33.83.1.175 lt _op_vector_sub_float32	868
33.83.1.176 lt _op_vector_sub_float64	868
33.83.1.177 lt _op_vector_sub_long	869
33.83.1.178 lt _op_vector_subs_char	869
33.83.1.179 lt _op_vector_tan_float32	869
33.83.1.180 lt _op_vector_tan_float64	870
33.83.1.181 lt _op_vector_tanh_float32	870
33.83.1.182 lt _op_vector_tanh_float64	870
33.83.1.183 lt _op_vector_trunc_float32	870
33.83.1.184 lt _op_vector_trunc_float64	871
33.83.1.185 lt _op_vector_wrap_float32	871

33.83.1.18 jit _op_vector_wrap_float64	871
33.84 QuickTime Codec Module	872
33.84.1 Function Documentation	873
33.84.1.1 jit_qt_codec_acodec2sym	873
33.84.1.2 jit_qt_codec_getcodeclist_audio	873
33.84.1.3 jit_qt_codec_getcodeclist_audio_raw	873
33.84.1.4 jit_qt_codec_getcodeclist_gfx	874
33.84.1.5 jit_qt_codec_getcodeclist_gfx_raw	874
33.84.1.6 jit_qt_codec_getcodeclist_video	874
33.84.1.7 jit_qt_codec_getcodeclist_video_raw	874
33.84.1.8 jit_qt_codec_qual2sym	875
33.84.1.9 jit_qt_codec_sym2acodec	875
33.84.1.10 jit_qt_codec_sym2qual	875
33.84.1.11 jit_qt_codec_sym2type	875
33.84.1.12 jit_qt_codec_sym2type_valid	876
33.84.1.13 jit_qt_codec_type2sym	876
33.84.1.14 jit_qt_codec_type2sym_valid	876
33.85 jit.qt.movie Module	877
33.85.1 Function Documentation	877
33.85.1.1 jit_qt_movie_matrix_calc	877
33.85.1.2 jit_qt_movie_matrix_to_image	877
33.85.1.3 jit_qt_movie_new	878
33.85.1.4 jit_qt_movie_read_typed	878
33.86 jit.qt.record Module	880
33.86.1 Function Documentation	880
33.86.1.1 jit_qt_record_matrix_calc	880
33.86.1.2 jit_qt_record_new	880
33.87 QuickTime Utilities Module	881
33.87.1 Function Documentation	882
33.87.1.1 jit_coerce_matrix_pixmap	882
33.87.1.2 jit_gworld_can_coerce_matrix	882
33.87.1.3 jit_gworld_clear	883
33.87.1.4 jit_gworld_matrix_equal_dim	883
33.87.1.5 jit_qt_utils_moviedataref_create	883
33.87.1.6 jit_qt_utils_moviefile_close	883
33.87.1.7 jit_qt_utils_moviefile_create	884

33.87.1.8 jit_qt_utils_str2type	884
33.87.1.9 jit_qt_utils_tempfile	884
33.87.1.10 jit_qt_utils_tempmoviefile_create	885
33.87.1.11 jit_qt_utils_trackmedia_add	885
33.87.1.12 jit_qt_utils_trackmedia_dispose	885
33.87.1.13 jit_qt_utils_trackmedia_get	886
33.87.1.14 jit_qt_utils_trackname_get	886
33.87.1.15 jit_qt_utils_trackname_set	886
33.87.1.16 jit_qt_utils_tracktype_get	886
33.87.1.17 jit_qt_utils_tracktypecode_get	886
33.87.1.18 jit_qt_utils_type2str	887
34 Data Structure Documentation	889
34.1 Ex_ex Struct Reference	889
34.1.1 Detailed Description	889
34.2 t_atom Struct Reference	890
34.2.1 Detailed Description	890
34.3 t_atomarray Struct Reference	891
34.3.1 Detailed Description	891
34.4 t_atombuf Struct Reference	892
34.4.1 Detailed Description	892
34.5 t_attr Struct Reference	893
34.5.1 Detailed Description	893
34.6 t_buffer Struct Reference	894
34.6.1 Detailed Description	896
34.7 t_celldesc Struct Reference	897
34.7.1 Detailed Description	897
34.8 t_charset_converter Struct Reference	898
34.8.1 Detailed Description	898
34.9 t_class Struct Reference	899
34.9.1 Detailed Description	899
34.10 t_datetime Struct Reference	900
34.10.1 Detailed Description	900
34.11 t_dictionary Struct Reference	901
34.11.1 Detailed Description	901
34.12 t_dictionary_entry Struct Reference	902
34.12.1 Detailed Description	902

34.13t_expr Struct Reference	903
34.13.1 Detailed Description	903
34.14t_fileinfo Struct Reference	904
34.14.1 Detailed Description	904
34.15t_funbuff Struct Reference	905
34.15.1 Detailed Description	906
34.16t_hashtab Struct Reference	907
34.16.1 Detailed Description	907
34.17t_hashtab_entry Struct Reference	908
34.17.1 Detailed Description	908
34.18t_indexmap Struct Reference	909
34.18.1 Detailed Description	909
34.19t_indexmap_entry Struct Reference	910
34.19.1 Detailed Description	910
34.20t_jbox Struct Reference	911
34.20.1 Detailed Description	911
34.21t_jboxdrawparams Struct Reference	912
34.21.1 Detailed Description	912
34.22t_jcolumn Struct Reference	913
34.22.1 Detailed Description	916
34.23t_jdataview Struct Reference	917
34.23.1 Detailed Description	920
34.24t_jgraphics_font_extents Struct Reference	921
34.24.1 Detailed Description	921
34.25t_jit_attr Struct Reference	922
34.25.1 Detailed Description	923
34.26t_jit_attr_filter_clip Struct Reference	924
34.26.1 Detailed Description	925
34.27t_jit_attr_filter_proc Struct Reference	926
34.27.1 Detailed Description	926
34.28t_jit_attr_offset Struct Reference	927
34.28.1 Detailed Description	928
34.29t_jit_attr_offset_array Struct Reference	929
34.29.1 Detailed Description	930
34.30t_jit_attribute Struct Reference	931
34.30.1 Detailed Description	932

34.31t_jit_gl_drawinfo Struct Reference	933
34.31.1 Detailed Description	933
34.32t_jit_glchunk Struct Reference	934
34.32.1 Detailed Description	935
34.33t_jit_matrix_info Struct Reference	936
34.33.1 Detailed Description	937
34.34t_jit_mop Struct Reference	938
34.34.1 Detailed Description	939
34.35t_jit_mop_io Struct Reference	940
34.35.1 Detailed Description	941
34.36t_jit_op_info Struct Reference	942
34.36.1 Detailed Description	942
34.37t_jmatrix Struct Reference	943
34.37.1 Detailed Description	943
34.38t_jrgb Struct Reference	944
34.38.1 Detailed Description	944
34.39t_jrgba Struct Reference	945
34.39.1 Detailed Description	945
34.40t_line_3d Struct Reference	946
34.40.1 Detailed Description	946
34.41t_linklist Struct Reference	947
34.41.1 Detailed Description	947
34.42t_llelem Struct Reference	948
34.42.1 Detailed Description	948
34.43t_matrix_conv_info Struct Reference	949
34.43.1 Detailed Description	949
34.44t_messlist Struct Reference	950
34.44.1 Detailed Description	950
34.45t_object Struct Reference	951
34.45.1 Detailed Description	951
34.46t_path Struct Reference	952
34.46.1 Detailed Description	952
34.47t_pathlink Struct Reference	953
34.47.1 Detailed Description	953
34.48t_pfftpub Struct Reference	954
34.48.1 Detailed Description	954

34.49t_privatesortrec Struct Reference	955
34.49.1 Detailed Description	956
34.50t_pt Struct Reference	957
34.50.1 Detailed Description	957
34.51t_pxjbox Struct Reference	958
34.51.1 Detailed Description	959
34.52t_pxobject Struct Reference	960
34.52.1 Detailed Description	960
34.53t_quickmap Struct Reference	961
34.53.1 Detailed Description	961
34.54t_rect Struct Reference	962
34.54.1 Detailed Description	962
34.55t_signal Struct Reference	963
34.55.1 Detailed Description	963
34.56t_size Struct Reference	964
34.56.1 Detailed Description	964
34.57t_stack_splat Struct Reference	965
34.57.1 Detailed Description	965
34.58t_string Struct Reference	966
34.58.1 Detailed Description	966
34.59t_symbol Struct Reference	967
34.59.1 Detailed Description	967
34.60t_symobject Struct Reference	968
34.60.1 Detailed Description	968
34.61t_tinyobject Struct Reference	969
34.61.1 Detailed Description	969
34.62t_wind_mouse_info Struct Reference	970
34.62.1 Detailed Description	970
34.63t_zll Struct Reference	971
34.63.1 Detailed Description	971
34.64word Union Reference	972
34.64.1 Detailed Description	972

Chapter 1

Objects in C: A Roadmap

Max has an extensive API for developing new objects in C. Before you start learning about it, however, we would like to save you time and make sure you learn the minimum about the API for what you need to do. Therefore, we've made a brief list of application areas for object development along with the sections of this document with which you'll probably want to become familiar.

1.1 Max Objects

For **logic and arithmetic objects**, such as new mathematical functions or more complex conditional operations than what is offered in Max, it should be sufficient to read the [Anatomy of a Max Object](#) section.

For objects that use [Data Structures](#), you'll want to read, in addition, the [Atoms and Messages](#) section to learn about Max's basic mechanisms for representing and communicating data.

If you are interested in writing interfaces to **operating system services**, you may need to learn about Max's [Threading](#) model and [The Scheduler](#).

For objects that deal with time and timing you'll want to learn about [The Scheduler](#). If you're interested in tempo-based scheduling, you'll want to read the section on [ITM](#) and look at the `delay2` example.

To create new user interface gadgets, you'll want to read all of the above, plus the section on Attributes and the [Anatomy of a UI Object](#). The section on [JGraphics](#) will also be helpful.

To create **objects with editing windows**, things are much more complicated than they used to be. You'll need to learn everything about UI objects, plus understand the `scripto` example object project.

For patcher scripting and interrogation objects, the section on [Scripting the Patcher](#), plus a few of the examples will be very helpful. It is also helpful to have a clear conceptual understanding of the patcher, which might be aided by reading the patcher scripting sections of the `js` object documentation.

1.2 MSP Objects

To create audio **filters** and **signal generators**, read the [Anatomy of a Max Object](#), then read the [Anatomy of a MSP Object](#) section. MSP objects make use of [Creating and Using Proxies](#) when receiving multiple audio inputs, so familiarity with that concept could be helpful.

For audio objects that output events (messages), you'll need to use the services of [The Scheduler](#), so we suggest reading about that.

For UI objects for analyzing and controlling audio, you'll need to learn about regular MSP objects as well as Max UI objects.

1.3 Jitter Objects

The [Jitter Object Model](#) outlines some important basic information about Jitter's flexible object model. [Jitter Max Wrappers](#) describes how to write Max wrapper objects that contain Jitter objects for use in the Max patcher world. [Matrix Operator QuickStart](#) and [Matrix Operator Details](#) describe how to create a particular type of Jitter object called matrix operators, or MOPs. [OB3D QuickStart](#) and [OB3D Details](#) describe how to create OB3D Jitter objects for use in rendering OpenGL scenes. [Scheduler and Low Priority Queue Issues](#) covers important threading and timing issues when building Jitter objects. [Jitter Object Registration and Notification](#) explains how Jitter objects can be registered by name and notify clients as they change or important events occur. [Using Jitter Objects in C](#) provides some examples of how to instantiate and take advantage of Jitter objects from C, just as one would from Java, Javascript, or the patcher. Finally, The [JXF File Specification](#) and [Jitter Networking Specification](#) contain information relating to the data formats involved in the JXF file format and Jitter networking protocols, respectively.

Chapter 2

Development System Information

2.1 Building

This SDK documentation is accompanied by a series of projects for compiling some example Max external objects. The details of how to build these projects are documented below in separate sections for the [Mac](#) and [Windows](#).

When you build the example projects, the resulting Max external will be located in a folder called "sdk-build" two folder-levels up from the project. If you leave the arrangement of folders intact, sdk-build will be found in the MaxSDK folder.

We recommende that you add the sdk-build folder to your Max search path using the File Preferences window. This permits you to put the MaxSDK folder wherever you like and load the objects in Max after building them without copying them to your Cycling '74 folder.

2.2 Mac

Max external objects for the Mac are Mach-O bundles (folders that appear to be files) whose file-names must end with the .mxo extension. The example projects are in Xcode 3.x format. To download Xcode, you need to open a free Apple Developer account. For more information, visit <http://developer.apple.com/>

2.2.1 XCode Project Setup

The example projects are set up to have Development and Deployment build configurations. The Development configuration does not optimize and builds only for the target platform you are using (i.e., PPC on a PPC machine, Intel on an Intel machine). The Deployment configuration creates a universal binary and performs optimization.

The files required for this projects are included in the project folders with the except of the following two files:

- Info.plist
- maxmspsdk.xcconfig

These two files are located one folder-level up from the project folder, and are required for the Xcode project to build the Max external.

2.2.2 Linking and Frameworks

External objects use dynamic linking to access the API functions provided by the Max application. When an objects is loaded, calls to functions inside the application are resolved by the operating system to the correct memory address. Each external object Xcode project must reference MaxAPI.framework in order to link with the application. Frameworks are libraries that define the functions in the Max API. Due to the fact that "Max" could exist as an application, a standalone you create, or a library inside another application, the MaxAPI.framework does not actually contain the code to implement the functions of the Max API for external objects. It serves instead to isolate external objects from the specific library or application implementation that contains the real code.

Audio objects will link against MaxAudioAPI.framework and Jitter objects link against Jitter-API.framework. Unlike MaxAPI.framework, these frameworks are real libraries. The most recent version of all frameworks will be found inside the application you are using (they are found inside the application

bundle in Contents/Frameworks). In addition, there are versions inside the c74support folder provided with the SDK. These will be used only to link your objects; they are never actually executed.

Xcode uses something called the Frameworks Search Path to locate frameworks when linking. The example SDK projects use a frameworks search path with a c74support folder two levels up from your the folder containing your Xcode project. If you rearrange the SDK folders, projects may not find the frameworks and will fail to link properly. Furthermore, even though we specify the frameworks search path, Xcode seems to look in /Library/Frameworks first. If you have installed a version of the Max SDK for version 4.6 or earlier, you may have older versions of MaxAPI.framework and MaxAudioAPI.framework in /Library/Frameworks. When you try to link objects that contain references to functions only defined in the newest MaxAPI.framework, the link may fail because the projects are using the old frameworks. To fix this, you'll need to remove the Max frameworks from /Library/Frameworks. If you want to develop objects for both the Max 4.6 and Max 5 SDKs on the same machine, you'll need to modify your 4.6 projects to specify a Frameworks Search Path, and relocate the 4.6 frameworks to the specified location.

2.3 Windows

Max external objects for Windows are Dynamic Link Libraries (DLLs) whose filenames must end with the .mxe extension. These DLLs will export a single function called "main" which is called by max when the external object is first loaded. Generally these DLLs will import functions of the Max API from the import library "MaxAPI.lib" which is located in the c74support\max-includes\ folder. External objects that use audio functionality will import functions from the import library "MaxAudio.lib" which is located in c74support\msp-includes\.

The example projects are in Visual C++ 2008 format. A free version of Visual C++ can be obtained from Microsoft at <http://www.microsoft.com/express/>. The projects are set up to have both a Debug and a Release configuration. The Release configuration is optimized whereas the Debug one is not. Note that for debugging purposes you can exercise your object in the Max Runtime since the copy protection for the Max Application will interfere when run under the debugger.

Another thing to note is that Max has a private build of the Microsoft C Runtime Library. By linking with this version of the C runtime library you won't have to worry about deployment issues due to dependencies your external may have on Microsoft's C Runtime. When you include "ext.h" from the max API it will include ext_prefix.h which for the release build will automatically cause your project to use the max C runtime library. If you prefer to use the Microsoft C Runtime you can do that by defining the C preprocessor macro MAXAPI_USE_MSCRT before including ext.h.

2.3.1 Compiling with Cygwin

It is also possible to compile Max external objects on Windows using Cygwin. The following steps show how to build the simplemax project from the MaxMSP SDK using Cygwin's gcc (Gnu Compiler Collection). This provides access to a high quality, free C compiler using the Cygwin Unix tools for Windows.

2.3.1.1 Requirements

Install the following Cygwin packages. Feel free to add on any other Cygwin packages that strike your fancy. The Cygwin installer and more information can be found at <http://www.cygwin.com/>

- Base (ALL)
- Devel
 - binutils

- gcc "GCC Compiler"
- gcc-mingw "Mingw32 support headers and libraries for GCC"
- gcc-mingw-core "Mingw32 support headers and libraries for GCC"
- mingw-runtime

2.3.1.2 Build Steps

STEP 0: cd to the directory containing the minimum SDK example project

STEP 1:

```
gcc -c -mno-cygwin -DWIN_VERSION -DWIN_EXT_VERSION -I../../c74support/max-include simplemax.c
```

Description of gcc arguments:

"-c" means compile.

"-mno-cygwin" means use the Microsoft standard C libraries, instead of Cygwin standard C libraries. This step is important if you wish to distribute your extern to people that might not have Cygwin installed.

"-DWIN_VERSION" and "-DWIN_EXT_VERSION" define these preprocessor definitions on the command line to guarantee that the header files and source code know it is being compiled for a Windows machine, instead of Macintosh.

"-I../../c74support/max-include" specifies an additional directory where the necessary headers files will be found.

"simplemax.c" is the compiler input.

STEP 2:

```
gcc -shared -mno-cygwin -o simplemax.mxe simplemax.o simplemax.def -L../../c74support/max-include -lMaxAPI
```

Description of gcc arguments:

"-shared" means link files to make a DLL.

"-mno-cygwin" means use the Microsoft standard C libraries, instead of Cygwin standard C libraries. This step is important if you wish to distribute your extern to people that might not have Cygwin installed.

"-o simplemax.mxe" specifies the name of the output file.

"simplemax.o" and "simplemax.def" are the linker input. The .def file is necessary to ensure that the function main will be exported.

"-L../../c74support/max-include" specifies an additional directory where library files will be found.

"-lMaxAPI" means link to the MaxAPI.lib linker library for MaxAPI.dll.

STEP 3: copy your file to a directory in your search path. For example:

```
cp minimum.mxe c:\Program Files\Common Files\Cycling '74\myexterns\
```

2.3.1.3 Additional Notes

You can ignore the warning that main() does not return int. This message is harmless, and only relevant to applications, not shared libraries.

2.4 Important Project Settings

The easiest way to create a new external is to choose one of the existing SDK examples, duplicate it, and then change only the settings that need to be changes (such as the name of the project). This will help to guarantee that important project settings are correct. Project settings of particular importance are noted below.

2.4.1 Mac

Particularly important for Max externals on the Mac are that the Info.plist is correct set up and that the "Force Package Info Generation" is set to true. Without these your object may fail to load on some machines.

2.4.2 Windows

In the preprocessor definitions for the Visual Studio project it is important to define WIN_VERSION and EXT_WIN_VERSION to ensure that the headers are set up properly.

2.5 Platform-specificity

If you are writing a cross-platform object and you need to do something that is specific to one platform, the Max API headers provide some predefined symbols you can use.

```
#ifdef MAC_VERSION
// do something specific to the Mac
#endif
#ifndef WIN_VERSION
// do something specific to Windows
#endif
```

Another reason for conditional compilation is to handle endianness on the Mac platform. If you are still supporting PowerPC, you may have situations where the ordering of bytes within a 16- or 32-bit word is important. ext_byteorder.h provides cross-platform tools for manipulating memory in an endian-independent way.

Chapter 3

Anatomy of a Max Object

Max objects are written in the C language, and the Max API is C-based.

You could use C++ but we don't support it at the API level. Writing a Max object in C, you have five basic tasks:

- 1) including the right header files (usually ext.h and ext_obex.h)
- 2) declaring a C structure for your object
- 3) writing an initialization routine called main that defines the class
- 4) writing a new instance routine that creates a new instance of the class, when someone makes one or types its name into an object box
- 5) writing methods (or message handlers) that implement the behavior of the object

Let's look at each of these in more detail. It's useful to open the simplemax example project as we will be citing examples from it.

3.1 Include Files

Most of the basic Max API is included in the files ext.h and ext_obex.h. These are essentially required for any object. Beyond this there are specific include files for more specialized objects.

The header files are cross-platform.

- jpatcher_api.h is required for any Max UI objects
- z_dsp.h is required for MSP audio objects

```
#include "ext.h" // should always be first, followed by ext_obex.h and any other files.
```

3.2 The Object Declaration

Basic Max objects are declared as C structures. The first element of the structure is a [t_object](#), followed by whatever you want. The example below has one long structure member.

```
typedef struct _simp
{
    t_object s_obj;      // t_object header
    long s_value;        // something else
} t_simp;
```

Your structure declaration will be used in the prototypes to functions you declare, so you'll need to place above these prototypes.

3.3 Initialization Routine

The initialization routine, which must be called main, is called when Max loads your object for the first time. In the initialization routine, you define one or more classes. Defining a class consists of the following:

- 1) telling Max about the size of your object's structure and how to create and destroy an instance
- 2) defining methods that implement the object's behavior
- 3) in some cases, defining attributes that describe the object's data
- 4) registering the class in a name space

Here is the simp class example initialization routine:

```
static t_class *s_simp_class; // global pointer to our class definition that is
    // setup in main()

int main()
{
    t_class *c;

    c = class_new("simp", (method)simp_new, (method)NULL, sizeof(t_simp), 0L, 0
    );
    class_addmethod(c, (method)simp_int, "int", A_LONG, 0);
    class_addmethod(c, (method)simp_bang, "bang", 0);

    class_register(CLASS_BOX, c);

    s_simp_class = c;

    return 0;
}
```

`class_new()` creates a class with the new instance routine (see below), a free function (in this case there isn't one, so we pass NULL), the size of the structure, a no-longer used argument, and then a description of the arguments you type when creating an instance (in this case, there are no arguments, so we pass 0).

`class_addmethod()` binds a C function to a text symbol. The two methods defined here are int and bang.

`class_register()` adds this class to the `CLASS_BOX` name space, meaning that it will be searched when a user tries to type it into a box.

Finally, we assign the class we've created to a global variable so we can use it when creating new instances.

More complex classes will declare more methods. In many cases, you'll declare methods to implement certain API features. This is particularly true for UI objects.

3.4 New Instance Routine

The standard new instance routine allocates the memory to create an instance of your class and then initializes this instance. It then returns a pointer to the newly created object.

Here is the simp new instance routine

```
void *simp_new()
{
    t_simp *x = (t_simp *)object_alloc(s_simp_class);

    x->s_value = 0;

    return x;
}
```

The first line uses the global variable `s_simp_class` we defined in the initialization routine to create a new instance of the class. Essentially, the instance is a block of memory of the size defined by the class, along with a pointer to the class that permits us to dispatch messages correctly.

The next line initializes our data. More complex objects will do a lot more here, such as creating inlets and outlets. By default, the object being created will appear with one inlet and no outlets.

Finally, in the last line, we return a pointer to the newly created instance.

3.5 Message Handlers

We are now ready to define some actual behavior for our object by writing C functions that will be called when our object is sent messages. For this simple example, we will write only two functions. `simp_int` will be called when our object receives numbers. It will store the received number in the `s_value` field. `simp_bang` will be called when our object receives a bang. It will print the value in the Max window. So, yes, this object is pretty useless!

The C functions you write will be declared according to the arguments the message requires. All functions are passed a pointer to your object as the first argument. For a function handling the int message, a single second argument that is a long is passed. For a function handling the bang message, no additional arguments are passed.

Here is the int method:

```
void simp_int(t_simp *x, long n)
{
    x->s_value = n;
}
```

This simply copies the value of the argument to the internal storage within the instance.

Here is the bang method:

```
void simp_bang(t_simp *x)
{
    post("value is %ld", x->s_value);
}
```

The `post()` function is similar to `printf()`, but puts the text in the Max window. `post()` is very helpful for debugging, particularly when you cannot stop user interaction or real-time computation to look at something in a debugger.

You can also add a float message, which is invoked when a floating-point number is sent to your object. Add the following to your initialization routine:

```
class_addmethod(c, (method) simp_float, "float", A_FLOAT, 0);
```

Then write the method that receives the floating-point value as follows:

```
void simp_float(t_simp *x, double f)
{
    post("got a float and it is %.2f", f);
}
```

Chapter 4

Inlets and Outlets

You are familiar with inlets and outlets when connecting two objects together in a patcher.

To receive data in your object or send data to other objects, you need to create the C versions of inlets and outlets. In this section, we'll explain what inlets and outlets are, how to create them, and how to use them. We'll also discuss a more advanced type of inlet called a proxy that permits a message to be received in any of your object's inlets. Proxies are used by audio objects to permit inlets to handle both signals and normal Max messages.

By default, every object shows one inlet. Additional inlets appear to the right of the default inlet, with the rightmost inlet being created last.

Inlets are essentially message translators. For example, if you create an int inlet, your object will receive the "in1" message instead of the "int" message when a number arrives at this newly created inlet. You can use the different message name to define special behavior for numbers arriving at each inlet. For example, a basic arithmetic object in Max such as + stores the number to be added when it arrives in the right inlet, but performs the computation and outputs the result when a number arrives in the left inlet.

Outlets define connections between objects and are used to send messages from your object to the objects to which it is connected. What is not obvious about an outlet, however, is that when you send a number out an outlet, the outlet-sending function does not return until all computation "below" the outlet has completed. This stack-based execution model is best illustrated by observing a patch with the Max debugger window. To understand this stack-based model it may be helpful to use the breakpoint and debugging features in Max and follow the stack display as you step through the execution of a patch. Outlets, like inlets, appear in the order you create them from right-to-left. In other words, the first inlet or outlet you create will be the visually farthest to the right.

4.1 Creating and Using Inlets

Proper use of an inlet involves two steps: first, add a method that will respond to the message sent via the inlet in your initialization routine, and second, create the inlet in your new instance routine. (Creating inlets at any other time is not supported.)

There are three types of inlets: int, float, and custom. We'll only describe int and float inlets here because proxies are generally a better way to create an inlet that can respond to any message. For int inlets, you'll bind a function to a message "in1", "in2", "in3" etc. depending on the inlet number you assign. Here's how to create a single inlet using "in1"...

In your initialization routine:

```
class_addmethod(c, (method)myobject_in1, "in1", A_LONG, 0);
```

In your new instance routine, after calling [object_alloc\(\)](#) to create your instance:

```
intin(x, 1);
```

The method that will be called when an int is received in the right inlet:

```
void myobject_in1(t_myobject *x, long n)
{
    // do something with n
}
```

Creating a single inlet in this way gives your object two inlets (remember that it always has one by default). If you want to create multiple inlets, you'll need to create them in order from right to left, as shown below:

```
intin(x, 2);      // creates an inlet (the right inlet) that will send your
                  // object the "in2" message
intin(x, 1);      // creates an inlet (the middle inlet) that will send you
                  // r object the "in1" message
```

Inlets that send float messages to your object are created with [floatin\(\)](#) and translate the float message into "ft1", "ft2", "ft3" etc. Example:

In initialization routine:

```
class_addmethod(c, (method)myobject_ft1, "ft1", A_FLOAT, 0);
```

In new instance routine:

```
floatin(x, 1);
```

Method:

```
void myobject_ft1(t_myobject *x, double f)
{
    post("float %.2f received in right inlet,f");
}
```

Note that you can mix int and float inlets, but each inlet must have a unique number. Example:

```
intin(x, 2);
floatin(x, 1);
```

4.2 Creating and Using Outlets

You create outlets in your new instance routine. Outlet creators return a pointer that you should store for later use when you want to send a message. As with inlets, outlets are created from right to left.

Here's a simple example. First we'll add two void pointers to our data structure to store the outlets for each instance.

```
typedef struct _myobject
{
    t_object m_ob;
    void *m_outlet1;
    void *m_outlet2;
} t_myobject;
```

Then we'll create the outlets in our new instance routine.

```
x = (t_myobject *)object_alloc(s_myobject_class);
x->m_outlet2 = bangout((t_object *)x);
x->m_outlet1 = intout((t_object *)x);
return x;
```

These outlets are type-specific, meaning that we will always send the same type of message through them. If you want to create outlets that can send any message, use [outlet_new\(\)](#). Type-specific outlets execute faster, because they make a direct connection to the method handler that will be called at the time you send a message. When we want to send messages out these outlets, say, in our bang method, we do the following:

```
void myobject_bang(t_myobject *x)
{
    outlet_bang(x->m_outlet2);
    outlet_int(x->m_outlet1, 74);
}
```

The bang method above sends the bang message out the `m_outlet2` outlet first, then sends the number 74 out the `m_outlet1`. This is consistent with the general design in Max to send values out outlets from right to left. However, there is nothing enforcing this design, and you could reverse the statements if you felt like it.

A more general message-sending routine, `outlet_anything()`, will be shown in the [Atoms and Messages](#) section.

4.3 Creating and Using Proxies

A proxy is a small object that controls an inlet, but does not translate the message it receives. Instead it sets a location inside your object's data structure to a value you associate with the inlet. If the message comes "directly" to your object via the left inlet, the value will be 0. However, in order to be thread-safe, you should not read the value of this "inlet number" directly. Instead, you'll use the `proxy_getinlet()` routine to determine the inlet that has received the message.

The advantage of proxies over regular inlets is that your object can respond to any message in all of its inlets, not just the left inlet. As a Max user, you may already appreciate the proxy feature without knowing it. For example, the pack object can combine ints, floats, lists, or symbols arriving in any of its inlets. It uses proxies to make this happen. MSP audio objects that accept signals in more than one inlet use proxies as well. In fact, the proxy capability is built into the way you create audio objects, as will be discussed in the [Anatomy of a MSP Object](#) section.

If your object's non-left inlets will only respond to ints or floats, implementing proxies is usually overkill.

4.4 Example

First, add a place in your object to store the proxy value. You shouldn't access this directly, but the proxy needs it. Second, you'll need to store the proxy, because you need to free it when your object goes away. If you create many proxies, you'll need to store pointers to all of them, but all proxies share the same long integer value field.

```
typedef struct _myobject
{
    t_object m_obj;
    long m_in;           // space for the inlet number used by all the proxies
    void *m_proxy;
} t_myobject;
```

In your new instance routine, create the proxy, passing your object, a non-zero code value associated with the proxy, and a pointer to your object's inlet number location.

```
x->m_proxy = proxy_new((t_object *)x, 1, &x->m_in);
```

If you want to create regular inlets for your object, you can do so. Proxies and regular inlets can be mixed, although such a design might confuse a user of your object.

Finally, here is a method that takes a different action depending on the value of `x->m_in` that we check using `proxy_getinlet()`.

```
void myobject_bang(t_myobject *x)
{
    switch (proxy_getinlet((t_object *)x)) {
        case 0:
```

```
    post("bang received in left inlet");
    break;
  case 1:
    post("bang received in right inlet");
    break;
}
}
```


Chapter 5

Atoms and Messages

When a Max object receives a message, it uses its class to look up the message selector ("int", "bang", "set" etc.

) and invoke the associated C function (method). This association is what you are creating when you use `class_admmethod()` in the initialization routine. If the lookup fails, you'll see an "object doesn't understand message" error in the Max window.

Message selectors are not character strings, but a special data structure called a symbol (`t_symbol`). A symbol holds a string and a value, but what is more important is that every symbol in Max is unique. This permits you to compare two symbols for equivalence by comparing pointers, rather than having to compare each character in two strings.

The "data" or argument part of a message, if it exists, is transmitted in the form of an array of atoms (`t_atom`). The atom is a structure that can hold integers, floats, symbols, or even pointers to other objects, identified by a tag. You'll use symbols and atoms both in sending messages and receiving them.

To illustrate the use of symbols and atoms, here is how you would send a message out an outlet. Let's say we want to send the message "green 43 crazy 8.34." This message consists of a selector "green" plus an array of three atoms.

First, we'll need to create a generic outlet with `outlet_new` in our new instance routine.

```
x->m_outlet = outlet_new((t_object *)x, NULL);
```

The second argument being `NULL` indicates that the outlet can be used to send any message. If the second argument had been a character string such as "int" or "set" only that specific message could be sent out the outlet. You'd be correct if you wondered whether `intout()` is actually just `outlet_new(x, "int")`.

Now that we have our generic outlet, we'll call `outlet_anything()` on it in a method. The first step, however, is to assemble our message, with a selector "green" plus an array of atoms. Assigning ints and floats to an atom is relatively simple, but to assign a symbol, we need to transform a character string into a symbol using `gensym()`. The `gensym()` function returns a pointer to a symbol that is guaranteed to be unique for the string you supply. This means the string is compared with other symbols to ensure its uniqueness. If it already exists, `gensym()` will supply a pointer to the symbol. Otherwise it will create a new one and store it in a table so it can be found the next time someone asks for it.

```
void myobject_bang(t_object *x)
{
    t_atom argv[3];

    atom_setlong(argv, 43);
    atom_setsym(argv + 1, gensym("crazy"));
    atom_setfloat(argv + 2, 8.34);

    outlet_anything(x->m_outlet, gensym("green"), 3, argv);
}
```

In the call to `outlet_anything()` above, `gensym("green")` represents the message selector. The `outlet_anything()` function will try to find a message "green" in each of the objects connected to the inlet. If `outlet_anything()` finds such a message, it will execute it, passing it the array of atoms it received.

If it cannot find a match for the symbol green, it does one more thing, which allows objects to handle messages generically. Your object can define a special method bound to the symbol "anything" that will be invoked if no other match is found for a selector. We'll discuss the anything method in a moment, but first, we need to return to `class_admmethod()` and explain the final arguments it accepts.

To access atoms, you can use the functions `atom_setlong()`, `atom_getlong()` etc. or you can access the `t_atom` structure directly. We recommend using the accessor functions, as they lead to both cleaner code and will permit your source to work without modifications when changes to the `t_atom` structure occur over time.

5.1 Argument Type Specifiers

In the simp example, you saw the int method defined as follows:

```
class_addmethod(c, (method) simp_int, "int", A_LONG, 0);
```

The `A_LONG`, 0 arguments to `class_addmethod()` specify the type of arguments expected by the C function you have written. `A_LONG` means that the C function accepts a long integer argument. The 0 terminates the argument specifier list, so for the int message, there is a single long integer argument.

The other options are `A_FLOAT` for doubles, `A_SYM` for symbols, and `A_GIMME`, which passes the raw list of atoms that were originally used to send the Max message in the first place. These argument type specifiers define what are known as "typed" methods in Max. Typed methods are those where Max checks the type of each atom in a message to ensure it is consistent with what the receiving object has said it expects for a given selector.

If the atoms cannot be coerced into the format of the argument type specifier, a bad arguments error is printed in the Max window.

There is a limit to the number of specifiers you can use, and in general, multiple `A_FLOAT` specifiers should be avoided due to the historically unpredictable nature of compiler implementations when passing floating-point values on the stack. Use `A_GIMME` for more than four arguments or with multiple floating-point arguments.

You can also specify that missing arguments to a message be filled in with default values before your C function receives them. `A_DEFLONG` will put a 0 in place of a missing long argument, `A_DEFFLOAT` will put 0.0 in place of a missing float argument, and `A_DEFSYM` will put the empty symbol (equal to `gensym("")`) in place of a missing symbol argument.

5.2 Writing A_GIMME Functions

A method that uses `A_GIMME` is declared as follows:

```
void myobject_message(t_myobject *x, t_symbol *s, long argc, t_atom *argv);
```

The symbol argument `s` is the message selector. Ordinarily this might seem redundant, but it is useful for the "anything" method as we'll discuss below.

`argc` is the number of atoms in the `argv` array. It could be 0 if the message was sent without arguments. `argv` is the array of atoms holding the arguments.

For typed messages, the atoms will be of type `A_SYM`, `A_FLOAT`, or `A_LONG`. Here is an example of a method that merely prints all of the arguments.

```
void myobject_printargs(t_myobject *x, t_symbol *s, long argc, t_atom *argv)
{
    long i;
    t_atom *ap;

    post("message selector is %s", s->s_name);
    post("there are %ld arguments", argc);

    // increment ap each time to get to the next atom
    for (i = 0, ap = argv; i < argc; i++, ap++) {
        switch (atom_gettype(ap)) {
            case A_LONG:
                post("%ld: %ld", i+1, atom_getlong(ap));
                break;
        }
    }
}
```

```

        case A_FLOAT:
            post("%ld: %.2f", i+1, atom_getfloat(ap));
            break;
        case A_SYM:
            post("%ld: %s", i+1, atom_getsym(ap)->s_name);
            break;
        default:
            post("%ld: unknown atom type (%ld)", i+1, atom_gettype(ap));
            break;
    }
}
}

```

You can interpret the arguments in whatever manner you wish. You cannot, however, modify the arguments as they may be about to be passed to another object.

5.3 Writing "Anything" Methods

As previously mentioned, your object can define a special method bound to the symbol "anything" that will be invoked if no other match is found for a selector. For example:

```
class_addmethod(c, (method)myobject_anything, "anything", A_GIMME, 0);
```

Your function definition for an anything method follows the same pattern as for all other [A_GIMME](#) methods:

```

void myobject_anything(t_myobject *x, t_symbol *s, long argc, t_atom *argv)
{
    object_post( (t_object*)x,
                "This method was invoked by sending the '%s' message to this object.",
                s->s_name);
    // argc and argv are the arguments, as described in above.
}

```

Chapter 6

The Scheduler

The Max scheduler permits operations to be delayed until a later time.

It keeps track of time in double-precision, but the resolution of the scheduler depends on the user's environment preferences. The scheduler also works in conjunction with a low-priority queue, which permits time-consuming operations that might be initiated inside the scheduler to be executed in a way that does not disrupt timing accuracy.

Most objects interface with the scheduler via a clock ([t_clock](#)) object. A clock is associated with a task function that will execute when the scheduler's current time reaches the clock's time. There is also a function called [schedule\(\)](#) that can be used for one-off delayed execution of a function. It creates a clock to do its job however, so if your object is going to be using the scheduler repeatedly, it is more efficient to store references to the clocks it creates so the clocks can be reused.

The scheduler is periodically polled to see if it needs to execute clock tasks. There are numerous preferences Max users can set to determine when and how often this polling occurs. Briefly:

- The Overdrive setting determines whether scheduler polling occurs in a high-priority timer thread or the main thread
- The Interval setting determines the number of milliseconds elapse between polling the scheduler
- The Throttle setting determines how many tasks can be executed in any particular scheduler poll

Similar Throttle and Interval settings exist for the low-priority queue as well.

For more information refer to the [Timing](#) documentation. While the details might be a little overwhelming on first glance, the important point is that the exact time your scheduled task will execute is subject to variability. Max permits this level of user control over the scheduler to balance all computational needs for a specific application.

6.1 Creating and Using Clocks

There are five steps to using a clock in an external object.

1. Add a member to your object's data structure to hold a pointer to the clock object

```
typedef struct _myobject
{
    t_object m_obj;
    void *m_clock;
} t_object;
```

2. Write a task function that will do something when the clock is executed. The function has only a single argument, a pointer to your object. The example below gets the current scheduler time and prints it.

```
void myobject_task(t_myobject *x)
{
    double time;

    sched_gettime(&time);
    post("instance %lx is executing at time %.2f", x, time);
}
```

3. In your new instance routine, create the clock (passing a pointer to your object and the task function) and store the result in your object's data structure.

```
x->m_clock = clock_new((t_object *)x, (method)myobject_task);
```

4. Schedule your clock. Use [clock_fdelay\(\)](#) to schedule the clock in terms of a delay from the current time. Below we schedule the clock to execute 100 milliseconds from now.

```
clock_fdelay(x->m_clock, 100.);
```

If you want to cancel the execution of a clock for some reason, you can use [clock_unset\(\)](#).

```
clock_unset(x->m_clock);
```

5. In your object's free routine, free the clock

```
object_free(x->m_clock);
```

Note that if you call [clock_delay\(\)](#) on a clock that is already set, its execution time will be changed. It won't execute twice.

6.2 Creating and Using Qelems

A qelem ("queue element") is used to ensure that an operation occurs in the low-priority thread. The task function associated with a [t_qelem](#) is executed when the low-priority queue is serviced, always in the main (user interface) thread. Any qelem that is "set" belongs to the low-priority queue and will be executed as soon as it serviced.

There are two principal things you want to avoid in the high priority thread: first, time-consuming or unpredictable operations such as file access, and second, anything that will block execution for any length of time -- for example, showing a dialog box (including a file dialog).

The procedure for using a qelem is analogous to that for using a clock.

1. Add a member to your object's data structure to hold a pointer to the qelem

```
typedef struct _myobject
{
    t_object m_obj;
    void *m_qelem
} t_myobject;
```

2. Write a task function that will do something when the qelem is executed. The function has only a single argument, a pointer to your object.

```
void myobject_qtask(t_myobject *x)
{
    post("I am being executed a low priority!");
}
```

3. In your new instance routine, create the qelem (passing a pointer to your object and the task function) and store the result in your object's data structure.

```
x->m_qelem = qelem_new((t_object *)x, (method)myobject_qtask);
```

4. Set the qelem by using [qelem_set\(\)](#). You could, for example, call [qelem_set\(\)](#) in a clock task function or in direct response to a message such as bang or int.

```
qelem_set(x->m_qelem);
```

If you want to cancel the execution of a qelem for some reason, you can use [qelem_unset\(\)](#).

```
qelem_unset(x->m_qelem);
```

5. In your object's free routine, call `qelem_free()`. Do not call `object_free()` or `freeobject()` -- unlike the clock, the qelem is not an object.

```
qelem_free(x->m_qelem);
```

Note that if you call `qelem_set()` on a qelem that is already set, it won't execute twice. This is a feature, not a bug, as it permits you to execute a low-priority task only as fast as the low-priority queue operates, not at the high-priority rate that the task might be triggered. An example would be that a number box will redraw more slowly than a counter that changes its value. This is not something you need to worry about, even if you are writing UI objects, as Max handles it internally (using a qelem).

6.3 Defer

The defer function and its variants use a qelem to ensure that a function executes at low-priority. There are three variants: `defer()`, `defer_low()`, and `defer_medium()`. The difference between using `defer()` and a qelem is that `defer()` is a one-shot deal -- it creates a qelem, sets it, and then gets rid of it when the task function has executed. The effect of this is that if you have some rapid high-priority event that needs to trigger something to happen at low-priority, `defer()` will ensure that this low-priority task happens every time the high-priority event occurs (in a 1:1 ratio), whereas using a qelem will only run the task at a rate that corresponds to the service interval of the low-priority queue. If you repeatedly `defer()` something too rapidly, the low-priority queue will become backlogged and the responsiveness of the UI will suffer.

A typical use of `defer()` is if your object implements a read message to ask the user for a file. Opening the dialog in the timer thread and waiting for user input will likely crash, but even if it didn't, the scheduler would effectively stop.

To use `defer()`, you write a deferred task function that will execute at low priority. The function will be passed a pointer to your object, plus a symbol and atom list modeled on the prototype for an anything method. You need not pass any arguments to the deferred task if you don't need them, however.

```
void myobject_deferredtask(t_myobject **x, t_symbol *s, long argc, t_atom *argv
)
{
    post("I am deferred");
}
```

To call the task, use `defer()` as shown below. The first example passes no arguments. The second passes a couple of long atoms.

```
defer((t_object *)x, (method)myobject_deferredtask, NULL, 0, NULL);

t_atom av[2];

atom_setlong(av, 1);
atom_setlong(av+ 2, 74);

defer((t_object *)x, (method)myobject_deferredtask, NULL, 2, av);
```

Defer copies any atoms you pass to newly allocated memory, which it frees when the deferred task has executed.

6.3.1 Defer Variants

`defer` has two variants, `defer_low()` and `defer_medium()`. Here is a comparison:

defer()

If executing at high priority, [defer\(\)](#) puts the deferred task at the front of the low-priority queue. If not executing at high priority, [defer\(\)](#) calls the deferred task immediately.

defer_low()

At all priority levels, [defer_low\(\)](#) puts the deferred task at the back of the low-priority queue.

defer_medium()

If executing at high priority, [defer_medium\(\)](#) puts the deferred task at the back of the low-priority queue. If not executing at high priority, [defer_medium\(\)](#) calls the deferred task immediately.

6.4 Schedule

The [schedule\(\)](#) function is to clocks as [defer\(\)](#) is to qelems. Schedule creates a clock for a task function you specify and calls [clock_fdelay\(\)](#) on it to make the task execute at a desired time. As with [defer\(\)](#), [schedule\(\)](#) can copy arguments to be delivered to the task when it executes.

A [schedule\(\)](#) variant, [schedule_defer\(\)](#), executes the task function at low priority after a specified delay.

Chapter 7

Memory Allocation

The Max API offers cross-platform calls memory management.

There are two types of calls, those for pointers and those for handles. Handles are pointers to pointers, and were used in the early Mac OS to permit memory to be relocated without changing a reference, and many Mac OS API calls used handle. There are a few legacy Max API calls that use handles as well, but in general, unless the OS or Max requires the use of a handle, you're probably better off using the simpler pointer.

Longtime Max object programmers may have used memory calls `getbytes()` and `freebytes()` in the past, but all memory calls now use same underlying OS mechanisms, so while `getbytes()` and `freebytes()` are still supported, they are restricted to 32K of memory or less due to the arguments they use, and we recommend the use of `sysmem_newptr()` and `sysmem_freeptr()` instead.

Here are some examples of allocating and freeing pointers and handles.

```
char *ptr;
char **hand;

ptr = sysmem_newptr(2000);
post("I have a pointer %lx and it is %ld bytes in size",ptr,
      sysmem_ptrsize(ptr));
ptr = sysmem_resizeptrclear(ptr, 3000);
post("Now I have a pointer %lx and it is %ld bytes in size",ptr,
      sysmem_ptrsize(ptr));
sysmem_freeptr(ptr);

hand = sysmem_newhandle(2000);
post("I have a handle %lx and it is %ld bytes in size",hand,
      sysmem_handlesize(hand));
sysmem_resizehandle(hand, 3000);
post("Now the handle %lx is %ld bytes in size",hand, sysmem_ptrsize(hand));

sysmem_freehandle(hand);
```

Chapter 8

Anatomy of a MSP Object

An MSP object that handles audio signals is a regular Max object with a few extras.

Refer to the simplemsp~ example project source as we detail these additions. simplemsp~ is simply an object that adds a number to a signal, identical in function to the regular MSP +~ object if you were to give it an argument of 1.

Here is an enumeration of the basic tasks:

8.1 Additional Header Files

After including ext.h and ext_obex.h, include z_dsp.h

```
#include "z_dsp.h"
```

8.2 C Structure Declaration

The C structure declaration must begin with a `t_pxobject`, not a `t_object`:

```
typedef struct _mydspobject
{
    t_pxobject m_obj;
    // rest of the structure's fields
} t_mydspobject;
```

8.3 Initialization Routine

When creating the class with `class_new()`, you must have a free function. If you have nothing special to do, use `dsp_free()`, which is defined for this purpose. If you write your own free function, the first thing it should do is call `dsp_free()`. This is essential to avoid crashes when freeing your object when audio processing is turned on.

```
c = class_new("mydspobject", (method)mydspobject_new, (method)dsp_free, siz
eof(t_mydspobject), NULL, 0);
```

After creating your class with `class_new()`, you must call `class_dspinit()`, which will add some standard method handlers for internal messages used by all signal objects.

```
class_dspinit(c);
```

Your signal object needs a method that is bound to the symbol "dsp" -- we'll detail what this method does below, but the following line needs to be added while initializing the class:

```
class_addmethod(c, (method)mydspobject_dsp, "dsp", A_CANT, 0);
```

8.4 New Instance Routine

The new instance routine must call `dsp_setup()`, passing a pointer to the newly allocated object pointer plus a number of signal inlets the object will have. If the object has no signal inlets, you may pass 0. The simplemsp~ object (as an example) has a single signal inlet:

```
dsp_setup((t_pxobject *)x, 1);
```

`dsp_setup()` will make the signal inlets (as proxies) so you need not make them yourself.

If your object will have audio signal outputs, they need to be created in the new instance routine with `outlet_new()`. However, you will never access them directly, so you don't need to store pointers to them as you do with regular outlets. Here is an example of creating two signal outlets:

```
outlet_new((t_object *)x, "signal");
outlet_new((t_object *)x, "signal");
```

8.5 The DSP Method and Perform Routine

The `dsp` method specifies the signal processing function your object defines along with its arguments. Your object's `dsp` method will be called when the MSP signal compiler is building a sequence of operations (known as the DSP Chain) that will be performed on each set of audio samples. The operation sequence consists of a pointers to functions (called perform routines) followed by arguments to those functions.

The `dsp` method is declared as follows:

```
void mydspobject_dsp(t_mydspobject **x, t_signal **sp, short *count);
```

To add an entry to the DSP chain, your `dsp` method uses `dsp_add()`. The `dsp` method is passed an array of signals (`t_signal` pointers), which contain pointers to the actual sample memory your object's perform routine will be using for input and output. The array of signals starts with the inputs (from left to right), followed by the outputs. For example, if your object has two inputs (because your new instance routine called `dsp_setup(x, 2)`) and three outputs (because your new instance created three signal outlets), the signal array `sp` would contain five items as follows:

```
sp[0] // left input
sp[1] // right input
sp[2] // left output
sp[3] // middle output
sp[4] // right output
```

The `t_signal` data structure (defined in `z_dsp.h`), contains two important elements: the `s_n` field, which is the size of the signal vector, and `s_vec`, which is a pointer to an array of 32-bit floats containing the signal data. All `t_signals` your object will receive have the same size. This size is not necessarily the same as the global MSP signal vector size, because your object might be inside a patcher within a `poly~` object that defines its own size. Therefore it is important to use the `s_n` field of a signal passed to your object's `dsp` method.

You can use a variety of strategies to pass arguments to your perform routine via `dsp_add()`. For simple unit generators that don't store any internal state between computing vectors, it is sufficient to pass the inputs, outputs, and vector size. For objects that need to store internal state between computing vectors such as filters or ramp generators, you will pass a pointer to your object, whose data structure should contain space to store this state. The `plus1~` object does not need to store internal state. It passes the input, output, and vector size to its perform routine. The `plus1~` `dsp` method is shown below:

```
void plus1_dsp(t_plus1 **x, t_signal **sp, short *count)
{
    dsp_add(plus1_perform, 3, sp[0]->s_vec, sp[1]->s_vec, sp[0]->s_n);
}
```

The first argument to `dsp_add()` is your perform routine, followed by the number of additional arguments you wish to copy to the DSP chain, and then the arguments.

The perform routine is not a "method" in the traditional sense. It will be called within the callback of an audio driver, which, unless the user is employing the Non-Real Time audio driver, will typically be in a high-priority thread. Thread protection inside the perform routine is minimal. You can use a clock, but you

cannot use qelems or outlets. The design of the perform routine is somewhat unlike other Max methods. It receives a pointer to a piece of the DSP chain and it is expected to return the location of the next perform routine on the chain. The next location is determined by the number of arguments you specified for your perform routine with your call to [dsp_add\(\)](#). For example, if you will pass three arguments, you need to return w + 4.

Here is the plus1 perform routine:

```
t_int *plus1_perform(t_int *w)
{
    t_float *in, *out;
    int n;

    in = (t_float *)w[1];           // get input signal vector
    out = (t_float *)w[2];          // get output signal vector
    n = (int)w[3];                 // vector size

    while (n--)                   // perform calculation on all samples
        *out++ = *in++ + 1.;

    return w + 4;                  // must return next DSP chain location
}
```

8.6 Free Function

The free function for the class must either be [dsp_free\(\)](#) or it must be written to call [dsp_free\(\)](#) as shown in the example below:

```
void mydspobject_free(t_mydspobject *x)
{
    dsp_free((t_pxobject *)x);

    // can do other stuff here
}
```

Chapter 9

Advanced Signal Object Topics

Here are some techniques for implementing additional features found in most signal objects.

9.1 Saving Internal State

To implement unit generators such as filters and ramp generators, you need to save internal state between calls to your object's perform routine. Here is a very simple low-pass filter (it just averages successive samples) that saves the value of the last sample in a vector to be averaged with the first sample of the next vector. First we add a field to our data structure to hold the value:

```
typedef struct _myfilter
{
    t_pxobject f_obj;
    t_float f_sample;
} t_myfilter;
```

Then, in our dsp method (which has one input and one output), we pass a pointer to the object as one of the DSP chain arguments. The dsp method also initializes the value of the internal state, to avoid any noise when the audio starts.

```
void myfilter_dsp(t_myfilter **x, t_signal **sp, short *count)
{
    dsp_add(myfilter_perform, 4, x, sp[0]->s_vec, sp[1]->s_vec, sp[0]->s_n);

    x->f_sample = 0;
}
```

Here is the perform routine, which obtains the internal state before entering the processing loop, then stores the most recent value after the loop is finished.

```
t_int *myfilter_perform(t_int *w)
{
    t_myfilter *x = (t_myfilter *)w[1];
    t_float *in = (t_float *)w[2];
    t_float *out = (t_float *)w[3];
    int n = (int)w[4];
    t_float samp = x->f_sample; // read from internal state
    t_float val;

    while (n--) {
        val = *in++;
        *out++ = (val + samp) * 0.5;
        samp = val;
    }
    x->f_sample = samp; // save to internal state

    return w + 5;
}
```

9.2 Observing Patcher Muting

The enable message to the pcontrol object, as well as the MSP mute~ object, can be used to disable a subpatcher. If your object is at all computationally expensive in its perform routine, it should check to see whether it has been disabled. To do this, you'll need to pass a pointer to your object as one of the DSP chain arguments when calling [dsp_add\(\)](#). Here is a simple modification of our filter object's perform routine that checks to see if the object has been disabled.

```
t_int *myfilter_perform(t_int *w)
{
    t_myfilter *x = (t_myfilter *)w[1];
    t_float *in = (t_float *)w[2];
    t_float *out = (t_float *)w[3];
    int n = (int)w[4];
    t_float samp = x->f_sample; // read from internal state
    t_float val;

    if (x->f_obj.z_disabled) // check for object being disabled
        return w + 5;

    while (n--) {
        val = *in++;
        *out++ = (val + samp) * 0.5;
        samp = val;
    }
    x->f_sample = samp; // save to internal state

    return w + 5;
}
```

9.3 Using Connection Information

The third argument to the `dsp` method is an array of numbers that enumerate the number of objects connected to each of your objects inputs and outputs. This array follows the same organization as the signal information as discussed in [The DSP Method and Perform Routine](#). More advanced `dsp` methods can use this information for optimization purposes. For example, if you find that your object has no inputs or outputs, you could avoid calling `dsp_add()` altogether. The MSP signal operator objects (such as `+~` and `*~`) to implement a basic polymorphism: they look at the connections count to determine whether the perform routine should use scalar or signal inputs. For example, if the right input has no connected signals, the user can add a scalar value sent to the right inlet.

To implement this behavior, you have a few different options. The first option is to write two different perform methods, one which handles the two-signal case, and one which handles the scalar case. The `dsp` method looks at the `count` array and passes a different function to `dsp_add()`. The example below assumes that the second element in the signal (`sp[1]`) and count (`count[1]`) arrays refer to the right input:

```
if (count[1]) // signal connected to second inlet
    dsp_add(mydspobject_twosigperform, 5, x, sp[0]->s_vec, sp[1]->s_vec, sp[
2]->s_vec, sp[0]->s_n);
else
    dsp_add(mydspobject_scalarperform, 4, x, sp[0]->s_vec, sp[2]->s_vec, sp[
0]->s_n);
```

The second option is to pass the value of the `count` array for a particular signal to the `perform` method, which can make the decision whether to use the signal value or a scalar value that has been stored inside the object. In this case, many objects use a single sample value from the signal as a substitute for the scalar. Using the first sample (i.e., the value at index 0) is a technique that works for any vector size, since vector sizes could be as small as a single sample. Here is an example of this technique for an object that has two inputs and one output. The connection count for the right input signal is passed as the second argument on the DSP chain, and the right input signal vector is passed even if it not connected:

```
dsp_add(mysdspobject_perform, 6, x, count[1], sp[0]->s_vec, sp[1]->s_vec, sp[
2]->s_vec, sp[0]->s_n);
```

Here is a `perform` routine that uses the connection count information as passed in the format shown above:

```
t_int mydspobject_perform(t_int *w)
{
    t_mydspobject *x = (t_mydspobject *)w[1];
    int connected = (int)w[2];
    t_float *in = (t_float *)w[3];
    t_float *in2 = (t_float *)w[4];
    t_float *out = (t_float *)w[5];
    int n = (int)w[6];

    double in2value;

    // get scalar sample or use signal depending on whether signal is connected

    in2value = connected? *in2 : x->m_scalarvalue;

    // do calculation here

    return w + 7;
}
```

Chapter 10

Sending Messages, Calling Methods

Max objects, such as the one you write, are C data structures in which methods are dynamically bound to functions.

Your object's methods are called by Max, but your object can also call methods itself. When you call a method, it is essential to know whether the method you are calling is **typed** or not.

Calling a typed method requires passing arguments as an array of atoms. Calling an untyped method requires that you know the exact arguments of the C function implementing the method. In both cases, you supply a symbol that names the method.

In the typed method case, Max will take the array of atoms and pass the arguments to the object according to the method's argument type specifier list. For example, if the method is declared to have an argument type specifier list of [A_LONG](#), 0, the first atom in the array you pass will be converted to an int and passed to the function on the stack. If there are no arguments supplied, invoking a typed method that has [A_LONG](#), 0 as an argument type specifier will fail. To make typed method calls, use [object_method_typed\(\)](#) or [typedmess\(\)](#).

In the untyped method case, Max merely does a lookup of the symbol in the object, and, if a matching function is found, calls the function with the arguments you pass.

Certain methods you write for your object, such as the assist method for describing your object and the DSP method in audio objects, are declared as untyped using the [A_CANT](#) argument type specifier. This means that Max will not typecheck the arguments you pass to these methods, but, most importantly, a user cannot hook up a message box to your object and send it a message to invoke an untyped method. (Try this for yourself -- send the assist message to a standard Max object.)

When you use an outlet, you're effectively making a typed method call on any objects connected to the outlet.

10.1 Attributes

Attributes are descriptions of data in your object. The standardization of these descriptions permits Max to provide a rich interface to object data, including the pattr system, inspectors, the quick reference menu, @ arguments, etc.

It is essential that you have some understanding of attributes if you are going to write a UI object. But non-UI objects can make use of attributes as well. The discussion below is not specific to UI objects. It does however, use the recently introduced system of macros in ext_obex_util.h (included in ext_obex.h) for defining attributes, as well as describing them using attributes of attributes (attr attrs). You can read more detailed descriptions of the underlying attribute definition mechanisms on a per-function basis in the [Attributes](#) reference.

10.1.1 Attribute Basics

While attributes can be defined for a specific instance of an object, it's much more common to define an attribute for a class. In such a case, each instance of the class will have the attribute description, but the value will be instance specific. The discussion here focuses only on class attributes.

When an attribute is declared and is made user-settable, a user can send a message to your object consisting of the attribute name and arguments that represent the new value of the attribute. For example, if you declare an attribute called trackcount, the message trackcount 20 will set it to 20. You don't need to do anything special to obtain this behavior. In addition, user-settable attributes will appear when the user opens the inspector on your object.

If you define your attribute as an offset attribute, you describe its location (and size) within your object's C data structure. Max can then read and write the data directly. You can also define custom getter and setter

routines if the attribute's value is more complex than simply a stored number. As a theoretical example, you could have an object with an attribute representing the Earth's population. If this value was not able to be stored inside your object, your custom getter routine could initiate a global census before returning the result. A custom setter for the earth's population might do something nasty if the value was set to zero. If you are not a misanthrope, you can take advantage of the ability to set such an attribute to be read-only.

10.1.2 Defining Attributes

Attributes are defined when you are defining methods in your initialization routine. You can define your attributes before your methods if you like, but by convention, they are typically defined after the methods. For each definition, you'll specify the name, size, and offset of the corresponding member in your object's data structure that will hold the data. For example, let's say we have an object defined as follows:

```
typedef struct _myobject {
    t_object m_ob;
    long m_targetaddress;
    t_symbol *m_shipname;
    char m_compatmode;
} t_myobject;
```

We want to create attributes for `m_targetaddress`, `m_shipname`, and `m_compatmode`. For each data type (and a few others), there are macros in `ext_obex_util.h` that will save a fair amount of typing. So, for example, we can define an attribute for `m_targetaddress` that uses `CLASS_ATTR_LONG`. Here are attribute definitions for all of the members of our data structure above.

```
CLASS_ATTR_LONG(c, "targetaddress", 0, t_myobject, m_targetaddress);
CLASS_ATTR_SYM(c, "shipname", 0, t_myobject, m_shipname);
CLASS_ATTR_CHAR(c, "compatibilitymode", 0, t_myobject, m_compatmode);
```

10.1.3 Attributes With Custom Getters and Setters

In some cases, it is not enough to have Max read and write data in your object directly. In some cases (as in the world population example above) you may have data you need to calculate before it can be returned as a value. In other cases, you may need to do something to update other object state when an attribute value changes. To handle these challenges, you can define custom attribute getter and setter routines. The getter will be called when the value of your attribute is accessed. The setter will be called when someone changes the value of your attribute.

As an example, suppose we have an object that holds onto an array of numbers, and we want to create an attribute for the size of the array. Since we'll want to resize the array when the attribute value changes, we will define a custom setter for our attribute. The default getter is adequate if we store the array size in our object, but since we want to illustrate how to write an attribute getter, we'll write the code so that the array size is computed from the size of the memory pointer we allocate. First, here is our object's data structure:

```
typedef struct _myobject {
    t_object m_ob;
    long *m_data;
} t_myobject;
```

We also have prototypes for our custom attribute setter and getter:

```
t_max_err myobject_size_get(t_myobject **x, t_object *attr, long *argc, t_atom
    **argv);
t_max_err myobject_size_set(t_myobject **x, t_object *attr, long argc, t_atom *
    argv);
```

Here is how we define our attribute using `CLASS_ATTR_ACCESSORS` macro to define the custom setter and getter. Because we aren't really using an "offset" due to the custom setter and getter, we can pass any data structure member as a dummy. (Only the default attribute getter and setter will use this offset, and they are out of the picture.)

```
CLASS_ATTR_LONG(c, "size", 0, t_myobject, m_ob);
CLASS_ATTR_ACCESSORS(c, "size", myobject_size_get, myobject_size_set);
```

Now, here is an implementation of the custom setter for the array size. For the setter, we use the handy Max API function `sysmem_resizeptr` so we can effectively "resize" our array and copy the data into it in one step. The setter uses atoms, so we have to obtain the value from the first item in the argv array.

```
t_max_err myobject_size_set(t_myobject *x, t_object *attr, long argc, t_atom *
    argv)
{
    long size = atom_getlong(argv);

    if (size < 0)      // bad size, don't change anything
        return 0;

    if (x->m_data)
        x->m_data = (long *)sysmem_resizeptr((char *)x->m_data, size * sizeof(long));
    else // first time alloc
        x->m_data = (long *)sysmem_newptr(size * sizeof(long));
    return 0;
}
```

The getter also uses atoms for access, but we are returning a pointer to an array of atoms. The caller of the getter has the option to pre-allocate the memory (passing in the length in argc and the pointer to the memory in argv) or pass in 0 for argc and set the contents of argv to NULL and have the getter allocate the memory. The easiest way to handle this case is to call the utility function `atom_alloc`, which will figure out what was passed in and allocate memory for a returned atom if necessary.

```
t_max_err myobject_size_get(t_myobject *x, t_object *attr, long *argc, t_atom *
    **argv)
{
    char alloc;
    long size = 0;

    atom_alloc(argc, argv, &alloc);      // allocate return atom

    if (x->m_data)
        size = sysmem_ptrsize((char *)x->m_data) / sizeof(long); // calculate array size based on ptr size

    atom_setlong(*argv, size);
    return 0;
}
```

10.2 Receiving Notifications

As an alternative to writing a custom setter, you can take advantage of the fact that objects receive a "notify" message whenever one of their attributes is changed. The prototype for a notify method is as follows:

```
t_max_err myobject_notify(t_myobject *x, t_symbol *s, t_symbol *msg, void *sender,
    void *data);
```

Add the following to your class initialization so your notification method will be called:

```
class_addmethod(c, (method)myobject_notify, "notify", A_CANT, 0);
```

The notify method can handle a variety of notifications (more documentation on this is coming soon!), but the one we're interested in is "attr_modified" -- the notification type is passed to the notify method in the msg argument. Here is an example of a notify method that prints out the name of the attribute that has been modified. You could take any action instead. To obtain the name, we interpret the data argument to the notify method as an attribute object. As an attribute is a regular Max object, we can use object_method to send it a message. In the case we are sending the message getname to the attribute object to obtain its name.

```
t_max_err myobject_notify(t_myobject *x, t_symbol *s, t_symbol *msg, void *sender, void *data)
{
    t_symbol *attrname;

    if (msg == gensym("attr_modified")) {      // check notification type
        attrname = (t_symbol *)object_method((t_object *)data, gensym("getname"))
    ;      // ask attribute object for name
        object_post((t_object *)x, "changed attr name is %s", attrname->s_name);
    }
    return 0;
}
```


Chapter 11

Anatomy of a UI Object

Max user interface objects are more complex than normal non-user-interface objects.

If you have nothing in particular to display, or do not need to create a unique interface for user interaction or editing, it would be better to avoid writing one. However, if you want the details, we have them for you!

In order to create a user interface object, you'll need to be familiar with [Attributes](#), as they are used extensively. If you examine a toggle object in the inspector in Max, you will see a few attributes that have been defined as belonging to the toggle class, namely:

- Background Color
- Check Color
- Border Color

We'll show how attributes are defined and described so that the inspector can edit them properly.

In addition to attributes, user interface objects draw in a box and respond to user events such as mouse clicks and keyboard events. We'll show how to implement drawing an object's paint method as well user interaction in the mousedown, mousedrag, and mouseup methods.

This chapter only covers basic drawing of lines and filled rectangles. But you can take advantage of a complete graphics API called `jgraphics`, intended to be used in a user interface object's paint method. We discuss [JGraphics](#) in more detail in a separate chapter. You may also find the `jgraphics.h` header file descriptions of the set of functions helpful.

The SDK examples contain two user interface projects -- the one we'll discuss in this chapter is called `uisimp` and is a version of the toggle object with a more complicated check box and user interaction. The second project is called `pictmeter~`, a more advanced object that uses audio as well as image files.

The `uisimp` object differs from the toggle object in a couple of ways:

- it tracks the mouse even when it isn't down and "looks excited" when the mouse passes over it
- it tracks the mouse while the user is holding the mouse down to show a sort of "depressed" appearance when turning the toggle on
- the new toggle state value is sent out when the mouse is released rather than when the mouse is down. In addition, the `uisimp` object tracks the mouse and does not change the state if the mouse is released outside of the object's box
- it doesn't have rounded corners
- it has a solid square for a "checked state" instead of an X

Otherwise, it acts largely as the toggle does.

The first thing we suggest you do is build the `uisimp` object and test it out. Once the object is properly building, type "`uisimp`" into an object box and you can try it out.

11.1 Required Headers

UI objects require that you include two header files, `jpatcher_api.h` and `jgraphics.h`:

```
#include "jpatcher_api.h"
#include "jgraphics.h"
```

The header file `jpatcher_api.h` includes data structures and accessor functions required by UI objects. The header file `jgraphics.h` includes data structures and functions for drawing.

11.2 UI Object Data Structure

The first part of a UI object is a `t_jbox`, not a `t_object`. You should generally avoid direct access to fields of a `t_jbox`, particularly when changing values, and use the accessor functions defined in `jpatcher_api.h`. For example, if you change the rectangle of a box without using the accessor function `jbox_set_rect()`, the patcher will not be notified properly and the screen will not update.

Following the `t_jbox`, you can add other fields for storing the internal state of your object. In particular, if you are going to be drawing something using color, you will want to create attributes that reference fields holding colors in your object. We'll show you how to do this below. Here is the declaration of the `t_uisimp` data structure.

```
typedef struct _uisimp
{
    t_jbox u_box;                      // header for UI objects
    void *u_out;                       // outlet pointer
    long u_state;                      // state (1 or 0)
    char u_mouseover;                  // is mouse over the object
    char u_mousedowninside;            // is mouse down within the object
    char u_trackmouse;                 // if non-zero, track mouse when button no
    t_down
    t_jrgba u_outline;                // outline color
    t_jrgba u_check;                  // check (square) color
    t_jrgba u_background;              // background color
    t_jrgba u_hilite;                 // highlight color (when mouse is over and wh
    en clicking to check box)
} t_uisimp;
```

The `t_jrgba` structure defines a color with four doubles for red, green, blue, and alpha. Each component ranges from 0-1. When red, green, and blue are all 0, the color is black; when red, green, and blue are 1, the color is white. By defining color attributes using `t_jrgba` structures, you will permit the user to use the standard color picker from the inspector to configure colors for your object.

The structure members `u_mouseover` and `u_mousedowninside` are used to signal the code that paints the toggle from the code that handles mouse interaction. We'll discuss this more in the "interaction strategy" section below.

11.3 Initialization Routine for UI Objects

Once you've declared your object's struct, you'll write your initialization (main) routine to set up the class, declaring methods and attributes used by UI objects.

The first addition to the class initialization of a normal Max object you need to make is a call to `jbox_initclass()`. This adds standard methods and attributes common to all UI objects. Here's how you should do it:

```
c = class_new("uisimp", (method)uisimp_new, (method)uisimp_free, sizeof(t_uisi
mp), 0L, A_GIMME, 0);

c->c_flags |= CLASS_FLAG_NEWDICTIONARY;
jbox_initclass(c, JBOX_FIXWIDTH | JBOX_COLOR);
```

The line `c->c_flags |= CLASS_FLAG_NEWDICTIONARY` is required, but the flags passed to `jbox_initclass` -- `JBOX_FIXWIDTH` and `JBOX_COLOR` -- are optional. `JBOX_FIXWIDTH` means that when your object is selected in a patcher, the Fix Width menu item will be enabled to resize your object to its class's default dimensions. We'll specify the default dimensions in a moment. `JBOX_COLOR` means that your object will be given a color attribute so that it can be edited with the color picked shown by the Color...

menu item. This is a way to edit a "basic" color of your object without opening the inspector. If neither of these behaviors apply to your object, feel free to pass 0 for the flags argument to `jbox_initclass()`.

11.4 UI Object Methods

Next we need to bind a few standard methods. The only required method for UI objects is `paint`, which draws the your object's content when its box is visible and needs to be redrawn.

```
class_addmethod(c, (method)uisimp_paint,      "paint", A_CANT, 0);
```

We'll discuss the `paint` method in detail below. It makes use of the [JGraphics](#) API, which is described in more detail in its own chapter.

Our `uisimp` toggle will respond to mouse gestures, so we will define a set of mouse handling methods.

```
class_addmethod(c, (method)uisimp_mousedown, "mousedown", A_CANT, 0);
class_addmethod(c, (method)uisimp_mousedrag, "mousedrag", A_CANT, 0);
class_addmethod(c, (method)uisimp_mouseup, "mouseup", A_CANT, 0);
class_addmethod(c, (method)uisimp_mouseenter, "mouseenter", A_CANT, 0);
class_addmethod(c, (method)uisimp_mouseleave, "mouseleave", A_CANT, 0);
class_addmethod(c, (method)uisimp_mousemove, "mousemove", A_CANT, 0);
class_addmethod(c, (method)uisimp_mousewheel, "mousewheel", A_CANT, 0);
```

`mousedown` is sent to your object when the user clicks on your object -- in other words, when the mouse is moved over the object and the primary mouse button is depressed. `mousedrag` is sent after an initial `mousedown` when the mouse moves and the button is still held down from the click. `mouseup` is sent when the mouse button is released after a `mousedown` is sent. `mouseenter` is sent when the mouse button is not down and the mouse moves into your object's box. `mousemove` is sent -- after a `mouseenter` -- when the mouse button is not down but the mouse position changes inside your object's box. `mouseleave` is sent when the mouse button is not down and the mouse position moves from being over your object's box to being outside of it. `mousewheel` is sent when information about the scrollwheel on the mouse (or scrolling from another source such as a trackpad) is transmitted while the cursor is hovering over your object.

You are not obligated to respond to any of these messages. You could, for example, only respond to `mousedown` and ignore the other messages.

It might be helpful to summarize mouse messages in the following "rules" (although normally it's not necessary to think about them explicitly):

- `mousedown` will always be followed by `mouseup`, but not necessarily by `mousedrag` if the button press is rapid and there is no movement while the mouse button is pressed.
- `mouseenter` will always be followed by `mouseleave`, but
- `mouseenter` will always precede `mousemove`
- `mouseleave` will be sent only after a `mouseenter` is sent
- You cannot count on any particular relationship between the `mousedown / mousedrag / mouseup` sequence and the `mouseenter /mousemove /mouseleave` sequence.

We'll look at the actual implementation of mouse handling methods below.

11.5 Defining Attributes

After the declaration of standard methods, your object will define its own attributes. By using what we call "attribute attributes" you can further describe attributes so that they can be appropriately displayed and edited in the inspector as well as saved in a patcher (or not). You can also set default values for attributes that are automatically copied to your object when it is instantiated, and mark an attribute so that your object is redrawn when its value changes.

As a convenience, we've defined a series of macros in ext_obex_util.h (which is included when your object includes ext_obex.h) that reduce the amount of typing needed to define attributes and attribute attributes.

Most UI object attributes are offset attributes; that is, they reference a location in your object's data structure by offset and size. As an example, uisimp has a char offset attribute called trackmouse that specifies whether the object will change the object's appearance when the mouse moves over it. Here's how this is defined:

```
CLASS_ATTR_CHAR(c, "trackmouse", 0, t_uisimp, u_trackmouse);
CLASS_ATTR_STYLE_LABEL(c, "trackmouse", 0, "onoff", "Track Mouse");
CLASS_ATTR_SAVE(c, "trackmouse", 0);
```

The first line, [CLASS_ATTR_CHAR](#), defines a char-sized offset attribute. If you look at the declaration of t_uisimp, you can see that the u_trackmouse field is declared to be a char. The [CLASS_ATTR_CHAR](#) macro takes five arguments.

- The first argument is the class for which the attribute is being declared.
- The second argument is the name of the attribute. You can use send a message to your object with this name and a value and set the attribute.
- The third argument is a collection of attribute flags. For the attributes (and attribute attributes) we'll be defining in the uisimp object, the flags will be 0, but you can use them to make attributes read-only with [ATTR_SET_OPAQUE_USER](#).
- The fourth argument is the name of your object's structure containing the field you want to use for the attribute
- The fifth argument is the field name you want to use for the attribute

The fourth and fifth arguments are used to calculate the offset of the beginning of the field from the beginning of the structure. This allows the attribute to read and write the memory occupied by the field directly.

The second line, [CLASS_ATTR_STYLE_LABEL](#), defines some attribute attributes for the trackmouse attribute. This macro takes five arguments as well:

- The first argument is the class for which the attribute attributes are being declared.
- The second argument is the name of the attribute, which should have already been defined by a [CLASS_ATTR_CHAR](#) or similar attribute declaration
- The third argument is usually 0 -- it is an attribute flags argument for the attribute attributes
- The fourth argument is the style of the attribute. "onoff" is used here for a setting in your object that will be a toggle. By using the onoff style the trackmouse attribute will appear with a checkbox in the inspector window. Effectively, this macro defines an attribute called "style" that is attached to the "trackmouse" attribute and set its value to the symbol "onoff" in one step.

- The fifth argument is a string used as a descriptive label for the attribute that appears in the inspector and other places in the Max user interface. If you don't supply a label, the attribute name will be shown. The string is used as the value of a newly created "label" attribute attribute.

The category attribute attribute is used to organize your object's attributes in the inspector window. For the trackmouse attribute, we use the "Behavior" category, and for the color attributes discussed below, we use "Color" -- look at the inspector category tabs for a few UI objects that come with Max for suggested standard category names. You're free to create your own.

To define a category for a single attribute, you can use the [CLASS_ATTR_CATEGORY](#) macro:

```
CLASS_ATTR_CATEGORY(c, "trackmouse", 0, "Behavior");
```

To define a category for a series of attributes, you can use [CLASS_STICKY_ATTR](#), which applies the current value of a specified attribute attribute to any attributes subsequently defined, until a [CLASS_STICKY_ATTR_CLEAR](#) is set for an attribute attribute name. [CLASS_STICKY_ATTR](#) is used in uisimp to apply the "Color" category to a set of three color attributes.

```
CLASS_STICKY_ATTR(c, "category", 0, "Color");
```

Color attributes are defined using [CLASS_ATTR_RGBA](#). The uisimp object defines four color attributes. Here is the first, called bgcolor:

```
CLASS_ATTR_RGBA(c, "bgcolor", 0, t_uisimp, u_background);
CLASS_ATTR_DEFAULTNAME_SAVE_PAINT(c, "bgcolor", 0, "1. 1. 1. 1.");
CLASS_ATTR_STYLE_LABEL(c, "bgcolor", 0, "rgba", "Background Color");
```

The difference between [CLASS_ATTR_RGBA](#) and [CLASS_ATTR_CHAR](#) for defining an attribute is that [CLASS_ATTR_RGBA](#) expects the name of a structure member declared of type [t_jrgba](#) rather than type char. When set, the attribute will assign values to the four doubles that make up the components of the color.

The next line uses the [CLASS_ATTR_DEFAULTNAME_SAVE_PAINT](#) macro. This sets three things about the bgcolor attribute. First it says that the color attribute bgcolor can be assigned a default value via the object defaults window. So, if you don't like the standard white defined by the object, you can assign you own color for the background color of all newly created uisimp objects. The four values 1 1 1 1 supplied as the last argument to [CLASS_ATTR_DEFAULTNAME_SAVE_PAINT](#) specify the "standard" default value that will be used for the bgcolor attribute in the absence of any overrides from the user.

The SAVE aspect of this macro specifies that this attribute's values should be saved with the object in a patcher. A patcher file saves an object's class, location and connections, but it can also save the object's appearance or any other attribute value you specify, by using the "save" attribute attribute.

The PAINT aspect of this macro provides the ability to have your object redrawn whenever this attribute (bgcolor) changes. However, to implement auto-repainting on attribute changes, you'll need to add the following code when initializing your class:

```
class_addmethod(c, (method) jbox_notify, "notify", A_CANT, 0);
```

The function [jbox_notify\(\)](#) will determine whether an attribute that has caused a change notification to be sent has its paint attribute attribute set, and if so, will call [jbox_redraw\(\)](#). If you write your own notify method because you want to respond to changes in attributes or other environment changes, you *must* call [jbox_notify\(\)](#) inside of it.

11.5.1 Standard Color Attribute

At the beginning of our initialization routine, we passed **JBOX_COLOR** as a flag to `jbox_initclass()`. This adds an attribute to our object called `color`, which uses storage provided in the `t_jbox` to keep track of a color for us. The `color` attribute is a standard name for the "most basic" color your object uses, and if you define it, the `Color` menu item in the `Object` menu will be enabled when your object is selected, permitting the user to change the color without opening the inspector.

If you use **JBOX_COLOR**, you don't need to define the `color` attribute using **CLASS_ATTR_RGBA** - `jbox_initclass()` will do it for you. However, the `color` attribute comes unadorned, so you are free to enhance it with attribute attributes. Here's what `uisimp` does:

```
CLASS_ATTR_DEFAULTNAME_SAVE_PAINT(c, "color", 0, "0. 0. 0. 1.");
CLASS_ATTR_STYLE_LABEL(c, "color", 0, "rgba", "Check Color");
```

11.5.2 Setting a Default Size

Another attribute defined for your object by `jbox_initclass()` is called `patching_rect`. It holds the dimensions of your object's box. If you want to set a standard size for new instances of your object, you can give the `patching_rect` a set of default values. Use 0 0 for the first two values (x and y position) and use the next two values to define the width and height. We want a small square to be the default size for `uisimp`, so we use **CLASS_ATTR_DEFAULT** to assign a default value to the `patching_rect` attribute as follows:

```
CLASS_ATTR_DEFAULT(c, "patching_rect", 0, "0. 0. 20. 20.");
```

11.6 New Instance Routine

The UI object new instance routine is more complicated than that of a normal Max object. Each UI object is passed a `t_dictionary` (a hierarchically structured collection of data accessed by symbolic names) containing the information needed to instantiate an instance. For UI objects, data elements in the dictionary correspond to attribute values. For example, if your object saved an attribute called "bgcolor" you will be able to access the saved value in your new instance routine from the dictionary using the same name `bgcolor`.

If the instance is being created from the object palette or by the typing the name of your object into an object box, the dictionary will be filled in with default values. If the object is being created by reading a patcher file, the dictionary will be filled in with the saved attributes stored in the file. In most cases, you don't need to work with the dictionary directly, unless you've added proprietary non-attribute information to your object's dictionary that you want to look for and extract. However, you do need to pass the dictionary to some standard routines, and initialize everything in the right order.

Let's take a look at the pattern you should follow for your object's new instance routine.

First, the new instance routine is declared as follows:

```
void *uisimp_new(t_symbol *s, long argc, t_atom *argv);
```

We will get the dictionary that defines the object out of the arguments passed in `argc`, `argv`. (The symbol argument `s` is the name of the object.) If obtaining the dictionary fails, we should return NULL to indicate we didn't make an instance.

```
void *uisimp_new(t_symbol *s, long argc, t_atom *argv);
{
    t_uisimp *x = NULL;
    t_dictionary *d = NULL;
```

```

long boxflags;

if (! (d = object_dictionaryarg(argc, argv)))
    return NULL;

```

Next, we allocate a new instance of the object's class:

```
x = (t_uisimp *)object_alloc(s_uisimp_class);
```

Then we need to initialize the options for our box. Our object uses the options that are not commented out.

```

boxflags = 0
| JBOX_DRAWFIRSTIN
| JBOX_NODRAWBOX
| JBOX_DRAWINLAST
| JBOX_TRANSPARENT
// | JBOX_NOGROW
| JBOX_GROWY
// | JBOX_GROWBOTH
// | JBOX_HILITE
// | JBOX_BACKGROUND
| JBOX_DRAWBACKGROUND
// | JBOX_NOFLOATINSPECTOR
// | JBOX_MOUSEDRAGDELTA
// | JBOX_TEXTFIELD
;

```

Here is some more detail about each of the box flags.

We pass the flags along with a pointer to our newly created instance and the argc, argv arguments to [jbox_new\(\)](#). The name is a little misleading. [jbox_new\(\)](#) does not instantiate your box. As we explained above, your UI object has a [t_jbox](#) at the beginning. [jbox_new\(\)](#) just initializes the [t_jbox](#) for you. [jbox_new\(\)](#) doesn't know about the other stuff in your object's data structure that comes after the [t_jbox](#). You'll have to initialize the extra items yourself.

```
jbox_new((t_jbox *)x, boxflags, argc, argv);
```

Once [jbox_new\(\)](#) has been called, you then assign the [b_firstin](#) pointer of your [t_jbox](#) header to point to your object. Essentially this assigns the object that will receive messages from objects connected to your leftmost inlet (as well as other inlets via inlets or proxies you create). This step is easily forgotten and will cause most things not to work until you remember it. [jbox_new\(\)](#) will obtain the attributes common to all boxes such as the [patching_rect](#), and assign them to your object for you.

```
x->u_box.b_firstin = (void *)x;
```

Next, you are free to initialize any members of your object's data structure, as well as declare inlets. These steps are the same for UI objects as for non-UI objects.

```
x->u_mousedowninside = x->u_mouseover = x->u_state = 0;
x->u_out = intout((t_object *)x);
```

Once your object is in a safe initialized state, call [attr_dictionary_process\(\)](#) if you've defined any attributes. This will find the attributes in the dictionary your object received, then set them to the values stored in the dictionary. There is no way to guarantee the order in which the attributes will be set. If this is a problem, you can obtain the attribute values "by hand" and assign them to your object.

Note that you do not need to call [attr_dictionary_process\(\)](#) if you have not defined any attributes. [jbox_new\(\)](#) will take care of setting all attributes common to all UI objects.

```
attr_dictionary_process(x, d);
```

As the last thing to do before returning your newly created UI object, and more specifically after you've initialized everything to finalize the appearance of your object, call [jbox_ready\(\)](#). [jbox_ready\(\)](#) will paint your object, calculate the positions of the inlets and outlets, and perform other initialization tasks to ensure that your box is a proper member of the visible patcher.

If your object does not appear when you instantiate it, you should check whether you do not have a [jbox_ready\(\)](#) call.

```
jbox_ready((t_jbox *)x);
```

Finally, as with any instance creation routine, the newly created object will be returned.

```
return x;
```

11.7 Dynamic Updating

Drawing anything to the screen must be limited to your paint method (this was not the case with the previous UI object API in Max). If you want to redraw something, you need to call [jbox_redraw\(\)](#) to cause the screen to be redrawn. This is necessary because your object is part of a compositing user interface that must be managed by the patcher as a whole to avoid screen artifacts. The [jbox_redraw\(\)](#) routine calculates the area of the screen that needs to be redrawn, then informs the Mac or Windows "window manager" to mark this area as invalid. At some later point in time, the OS will invoke the patcher's paint routine, which will dispatch to all of the boxes inside the invalid area according to the current Z-order of all the boxes. Boxes that are in the background are drawn first, so that any transparent or semi-transparent boxes can be drawn on top of them. In addition, unless you specify otherwise, the last drawn image of a box is cached in a buffer, so that your paint method will only be called when you explicitly invalidate your object's content with [jbox_redraw\(\)](#). In other words, you can't count on "global patcher drawing" to invoke your paint method.

The basic strategy you'll want to use in thinking about redrawing is that you will set internal state in other methods, then call [jbox_redraw\(\)](#). The paint method will read the internal state and adjust its drawing appropriately. You'll see this strategy used in the uisimp object as it tracks the mouse.

11.8 The Paint Method

Your object's paint method uses the jgraphics API to draw. The header file, jgraphics.h, provides a description of each of the routines in the API. Here we will only discuss general principles and features of drawing with uisimp's relatively simple paint method. There is also a jgraphics example UI object that contains a number of functions showing how various drawing tasks can be performed.

Drawing in Max is resolution-independent. The "size" of your object's rectangle is always the pixel size when the patcher is scaled to 100% regardless of the zoom level, and any magnification or size reduction to the actual screen is automatically handled by matrix transforms. Another thing that is handled automatically for you is drawing to multiple views. If a patcher is invisible (i.e., a subpatcher that has not been double-clicked), it does not have any views. But if it is visible, a patcher can have many patcherviews. If your UI object box is in a patcher with multiple views open, your paint method will be called once for each view, and will be passed different a patcherview object each time. For most objects, this will pose few problems, but for objects to work properly when there are anywhere from zero to ten views open, they cannot change their internal state in the paint method, they can only read it. As an example, if your object had a boolean "painted" field in its structure that would be set when the paint method had finished, it would not work

properly in the cases where the box was invisible or where it was shown in multiple patcher views, because it would either be set zero or more than once.

The first step for any paint method is to obtain the `t_jgraphics` object from the patcherview object passed to the paint method. The patcherview is an opaque `t_object` that you will use to access information about your box's rectangle and its graphics context. A patcherview is not the same thing as a patcher; as mentioned above, there could be more than one patcherview for a patcher if it has multiple views open.

```
void uisimp_paint(t_uisimp *x, t_object *patcherview)
{
    t_rect rect;

    t_jgraphics *g = (t_jgraphics*) patcherview_get_jgraphics(patcherview);      /
    / obtain graphics context
```

After obtaining the `t_jgraphics` object, the next thing that you'll need to do is determine the rectangle of your box. A view of a patcher may be in either patching or presentation mode. Since each mode can have its own rectangle, it is necessary to use the patcherview to obtain the rectangle for your object.

```
jbox_get_rect_for_view((t_object *)x, patcherview, &rect);
```

The `t_rect` structure specifies a rectangle using the x and y coordinates of the top left corner, along with the width and height. However, the coordinates of the `t_jgraphics` you'll be using to draw into always begin at 0 for the top left corner, so you'll only care about the width and height, at least for drawing.

The first thing we'll draw is just an outline of our box using the value of the outline color attribute. First we'll set the color we want to use, then make a rectangular path, then finally we'll stroke the path we've made.

With calls such as `jgraphics_rectangle()`, the rectangular shape is added to the existing path. The initial path is empty, and after calling `jgraphics_stroke()` or `jgraphics_fill()`, the path is again cleared. (If you want to retain the path, you can use the `jgraphics_stroke_preserve()` and `jgraphics_fill_preserve()` variants.)

```
jgraphics_set_source_jrgba(g, &x->u_outline);
jgraphics_set_line_width(g, 1.);
jgraphics_rectangle(g, 0., 0., rect.width, rect.height);
jgraphics_stroke(g);
```

You do not need to destroy the path before your paint method is finished. This will be done for you, but the fact that the path does not survive after the paint method is finished means you can't make a path and then store it without copying it first. Such a strategy is not recommended in any case, since your object's rectangle might change unpredictably from one paint method invocation to the next, which will likely cause your path to be the wrong shape or size.

The next feature of the paint method is to draw an inner outline if the mouse is moved over the box. Detecting the mouse's presence over the box happens in the `mouseenter` / `mouseleave` methods described below -- but essentially, we know that the mouse is over our object if the `u_mouseover` has been set by these mouse tracking methods.

To draw a rectangle that is inset by one pixel from the box rectangle, we use the rectangle starting at 1, 1 with a width of the box width - 2 and a height of the box height - 2.

```
// paint "inner highlight" to indicate mouseover
if (x->u_mouseover && !x->u_mousedowninside) {
    jgraphics_set_source_jrgba(g, &x->u_hilite);
    jgraphics_set_line_width(g, 1.);
    jgraphics_rectangle(g, 1., 1., rect.width - 2, rect.height - 2);
    jgraphics_stroke(g);
}
```

Some similar code provides the ability to show the highlight color when the user is about to check (turn on) the toggle:

```
if (x->u_mousedowninside && !x->u_state) {      // paint hilite color
    jgraphics_set_source_jrgba(g, &x->u_hilite);
    jgraphics_rectangle(g, 1., 1., rect.width - 2, rect.height - 2);
    jgraphics_fill(g);
}
```

Finally, we paint a square in the middle of the object if the toggle state is non-zero to indicate that the box has been checked. Here we are filling a path instead of stroking it. Note also that we use the call [jbox_get_color\(\)](#) to get the "standard" color of our object that is stored inside the [t_jbox](#). As we've specified by using the [JBOX_COLOR](#) flag for [jbox_initclass\(\)](#) in our initialization routine, the color obtained by [jbox_get_color\(\)](#) for the "check" (really just a square of solid color) is the one the user can change with the Color... item in the Object menu.

```
if (x->u_state) {
    t_jrgba col;

    jbox_get_color((t_object *)x, &col);
    jgraphics_set_source_jrgba(g, &col);
    if (x->u_mousedowninside)      // make rect bigger if mouse is down and we are unchecking
        jgraphics_rectangle(g, 3., 3., rect.width - 6, rect.height - 6);
    else
        jgraphics_rectangle(g, 4., 4., rect.width - 8, rect.height - 8);
    jgraphics_fill(g);
}
```

Clearly, a quick perusal of the `jgraphics.h` header file will demonstrate that there is much more to drawing than we've discussed here. But the main purpose of the `uisimp` paint method is to show how to implement "dynamic" graphics that follow the mouse. Now we'll see the mouse tracking side of the story.

11.9 Handling Mouse Gestures

When the mouse is clicked, dragged, released, or moved inside its box, your object will receive messages. In the `uisimp` example we've defined methods for most of the mouse gesture messages available, and we've implemented them to change internal state in the object, then call [jbox_redraw\(\)](#) to repaint the object to reflect the new state. This strategy produces a "dynamic" appearance of a gadget users associate with a typical graphical interface -- in this case a toggle checkbox.

All mouse gesture methods are declared in the same way:

```
void myobject_mouse(t_myobject *x, t_object *patcherview, t_pt pt, long modifiers);
```

Let's first look at the most commonly implemented mouse gesture handler, the `mousedown` method that responds to an initial click on the object. As you can see, it is very simple; it merely sets `u_mousedowninside` to true, then calls [jbox_redraw\(\)](#), causing the box to be repainted. We've defined this toggle not to change the actual state until the mouse is released (unlike the standard Max toggle object), but we do want to give the user some feedback on the initial mouse down that something is going to happen. If you look back at the paint method, you can see that `u_mousedowninside` is used to change the way the object is painted to give it a "pending state change" appearance that will be finalized when the mouse is released inside the box.

```

void uisimp_mousedown(t_uisimp *x, t_object *patcherview, t_pt pt, long modifiers)
{
    x->u_mousedowninside = true; // wouldn't get a click unless it was inside
    the box
    jbox_redraw((t_jbox *)x);
}

```

If we test the mouse position to ensure that it is inside the box when it is released, we provide the opportunity for the user to cancel the act of toggling the state of the object by moving the cursor outside of the box before releasing the button. To provide feedback to the user that this is going to happen, we've implemented a mousedrag method that performs this test and redraws the object if the "mouse inside" condition has changed from its previous state. The mousedrag message will be sent to your object as long as the mouse button is still down after an initial click and the cursor has moved, even if the cursor moves outside of the boundaries of your object's box.

Note that, as with the paint method, we use the patcherview to get the current box rectangle. We can then test the point we are given to see if it is inside or outside the box.

```

void uisimp_mousedrag(t_uisimp *x, t_object *patcherview, t_pt pt, long modifiers)
{
    t_rect rect;

    // test to see if mouse is still inside the object
    jbox_get_rect_for_view((t_object *)x, patcherview, &rect);

    // redraw if changed
    if (pt.x >= 0 && pt.x <= rect.width && pt.y >= 0 && pt.y <= rect.height) {
        if (!x->u_mousedowninside) {
            x->u_mousedowninside = true;
            jbox_redraw((t_jbox *)x);
        }
    } else {
        if (x->u_mousedowninside) {
            x->u_mousedowninside = false;
            jbox_redraw((t_jbox *)x);
        }
    }
}

```

Our mouseup method uses the last value of u_mousedowninside as the determining factor for whether to toggle the object's internal state. If u_mousedowninside is false, no state change happens. But if it is true, the state changes and the new state value is sent out the object's outlet (inside `uisimp_bang()`).

```

if (x->u_mousedowninside) {
    x->u_state = !x->u_state;
    uisimp_bang(x);
    x->u_mousedowninside = false;
    jbox_redraw((t_jbox *)x);
}

```

Finally, we've implementedmouseenter, mousemove, and mouseleave methods to provide another level of "mouse over" style highlighting for the object. Rather than changing u_mousedowninside, a u_mouseover field is set when themouseenter message is received, and cleared when themouseleave method is received. And again, after this variable is manipulated, we repaint the box with `jbox_redraw()`.

```

void uisimp_mouseenter(t_uisimp *x, t_object *patcherview, t_pt pt, long modifiers)
{

```

```
x->u_mouseover = true;
jbox_redraw((t_jbox *)x);
}

void uisimp_mouseleave(t_uisimp *x, t_object *patcherview, t_pt pt, long modifiers)
{
    x->u_mouseover = false;
    jbox_redraw((t_jbox *)x);
}
```

11.10 Freeing a UI Object

If your object has created any clocks or otherwise allocated memory that should be freed when the object goes away, you should handle this in the free routine. But, most importantly, you must call the function [jbox_free\(\)](#). If your UI object doesn't need to do anything special in its free routine, you can pass [jbox_free\(\)](#) as the free routine argument to [class_new\(\)](#) in your initialization routine. We chose not to do this, since having an actual function permits easy modification should some memory need to be freed at some point in the future evolution of the object.

```
void uisimp_free(t_uisimp *x)
{
    jbox_free((t_jbox *)x);
}
```


Chapter 12

File Handling

Max contains a cross-platform set of routines for handling files.

These routines permit you to search for files, show file open and save dialogs, as well as open, read, write, and close them. The file API is based around a "path identifier" -- a number that describes the location of a file. When searching or reading a file, path identifiers can be either a folders or collectives. Path identifiers that are negative (or zero) describe actual folders in the computer's file system, while path identifiers that are positive refer to collectives.

A basic thing you might want to do make your object accept the read message in a manner similar to existing Max objects. If the word read is followed by no arguments, a file dialog appears for the user to choose a file. If read is followed by an argument, your object will search for the file. If a file is found (or chosen), your object will open it and read data from it.

First, make your object accept the read message. The simplest way to make the filename argument optional is to use the [A_DEFSYM](#) argument type specifier. When the symbol argument is not present, Max passes your method the empty symbol.

```
class_addmethod(c, (method)myobject_read, "read", A_DEFSYM, 0);
```

The next requirement for any method that reads files is that it must defer execution to the low-priority thread, as shown in the following implementation, where the filename argument is passed as the symbol argument to defer.

```
void myobject_read(t_myobject *x, t_symbol *s)
{
    defer(x, (method)myobject_doread, s, 0, NULL);
}
```

The myobject_doread() function compares the filename argument with the empty symbol -- if the argument was not supplied, the [open_dialog\(\)](#) is used, otherwise, we call [locatefile_extended\(\)](#) to search for the file. This object looks for text files, so we use a four-character code 'TEXT' as our file type to either open or locate. File type codes define a set of acceptable extensions. The file max-fileformats.txt permits contains standard definitions, and you can add your own by creating a similar text file and placing it in the init folder inside the Cycling '74 folder.

```
void myobject_doread(t_myobject *x, t_symbol *s)
{
    long filetype = 'TEXT', outtype;
    short numtypes = 1;
    char filename[512];
    short path;

    if (s == gensym("")) { // if no argument supplied, ask for file
        if (open_dialog(filename, &path, &outtype, &filetype, 1)) // non-zero
            ro: user cancelled
                return;
        } else {
            strcpy(filename, s->s_name); // must copy symbol before calling locatefile_extended
            if (locatefile_extended(filename, &path, &outtype, &filetype, 1)) { // non-zero: not found
                object_error(x, "%s: not found", s->s_name);
                return;
            }
        }
    // we have a file
    myobject_openfile(x, filename, path);
}
```

To open and read files, you can use the cross-platform sysfile API. Files can be opened using a filename plus path identifier. If successfully opened, the file can be accessed using a [t_filehandle](#). Note that "files" inside collective files are treated identically to regular files, with the exception that they are read-only.

12.1 Reading Text Files

First, we'll implement reading the text file whose name and path identifier are passed to `myobject_openfile()` using a high-level routine `sysfile_readtextfile()` specifically for reading text files that handles text encoding conversion for you. If you are reading text files, using this routine is strongly recommended since converting text encodings is unpleasant to say the least.

```
void myobject_openfile(t_myobject *x, char *filename, short path)
{
    t_filehandle fh;
    char **texthandle;

    if (path_opensysfile(filename, path, &fh, READ_PERM)) {
        object_error(x, "error opening %s", filename);
        return;
    }
    // allocate some empty memory to receive text
    texthandle = sysmem_newhandle(0);
    sysfile_readtextfile(fh, texthandle, 0, 0);      // see flags explanation below
    post("the file has %ld characters", sysmem_gethandlesize(texthandle));
    sysfile_close(fh);
    sysmem_freehandle(texthandle);
}
```

In most situations, you will pass 0 for the final two arguments to `sysfile_readtextfile()`. The third argument specifies a maximum length to read, but if the value is 0, the entire file is read in, regardless of its size. The final argument is a set of flags specifying options for reading in the text. The options concern the conversion of line breaks, text encoding, and the ability to add a null character to the end of the data returned.

Line breaks are converted on the basis of any line break flags. When reading text files, Max converts line breaks to "native" format, which is

\r\n

on Windows and

\n

on the Mac; this is the behavior you get if you either pass no line break flags or use `TEXT_LB_NATIVE`. Other options include `TEXT_LB_MAC`, `TEXT_LB_UNIX`, or `TEXT_LB_PC`.

By default, text files are converted from their source encoding to UTF-8. If you do not want this conversion to occur, you can use the `TEXT_ENCODING_USE_FILE` flag. This puts the burden on determining the encoding on you, which is probably not what you want. For example, the source text file might use UTF-16 encoding, which requires very different parsing than an 8-bit encoding.

Finally, you can have the memory returned from `sysfile_readtextfile()` terminated with a NULL character if you use the `TEXT_NULL_TERMINATE` flag.

12.2 Reading Data Files

To read data files where you do not want to do text encoding conversion or worry about line breaks, you can use the same technique shown above for text files, but write the `myobject_openfile` function using `sysfile_read()` instead of `sysfile_readtextfile()`. This example shows how to read an entire file into a single block of memory.

```

void myobject_openfile(t_myobject *x, char *filename, short path)
{
    t_filehandle fh;
    char *buffer;
    long size;

    if (path_opensysfile(filename, path, &fh, READ_PERM)) {
        object_error(x, "error opening %s", filename);
        return
    }
    // allocate memory block that is the size of the file
    sysfile_geteof(fh, &size);
    buffer = sysmem_newptr(size);

    // read in the file
    sysfile_read(fh, &size, buffer);

    sysfile_close(fh);

    // do something with data in buffer here

    sysmem_freeteptr(buffer);      // must free allocated memory
}

```

12.3 Writing Files

Some Max objects respond to the write message to save data into a file. If there is no argument present after the word write, a save file dialog is shown and the user specifies a file name and location. If an argument is present, it can either specify a complete path name or a filename. In the filename case, the file is written to the current "default" directory, which is the location where a patcher was last opened. In the full pathname case, the file is written to the location specified by the pathname.

Here's how to implement this behavior. We'll show how to handle the message arguments, then provide text and data file writing examples.

Message and argument handling is very similar to the way we implemented the read message above, including the use of deferred execution.

```

class_addmethod(c, (method)myobject_write, "write", A_DEFSYM, 0);

void myobject_write(t_myobject *x, t_symbol *s)
{
    defer(x, (method)myobject_dowrite, s, 0, NULL);
}

```

The myobject_dowrite() function compares the filename argument with the empty symbol -- if the argument was not supplied, [saveasdialog_extended\(\)](#) is used to obtain the user's choice for filename and location. Our first example looks for text files, so we use a four-character code 'TEXT' as our file type for saving. File type codes define a set of acceptable extensions. The file max-fileformats.txt permits contains standard definitions, and you can add your own by creating a similar text file and placing it in the init folder inside the Cycling '74 folder.

```

void myobject_dowrite(t_myobject *x, t_symbol *s)
{
    long filetype = 'TEXT', outtype;
    short numtypes = 1;
    char filename[512];
    short path;

    if (s == gensym("")) {      // if no argument supplied, ask for file

```

```

        if (saveasdialog_extended(filename, &path, &outtype, &filetype, 1))      /
    / non-zero: user cancelled
        return;
    } else {
        strcpy(filename, s->s_name);
        path = path_getdefault();
    }
    myobject_writefile(x, filename, path);
}

```

Here is the text file variant of myobject_writefile() using the high-level [sysfile_writetextfile\(\)](#) routine. We just write a sentence as our "text file" but your object will presumably have some text data stored internally that it will write. The buffer passed to [sysfile_writetextfile\(\)](#) must be NULL-terminated, and will be assumed to be UTF-8 encoded.

Note that [path_createsysfile\(\)](#) can accept a full path in the filename argument, in which case, the path argument is ignored. This means your object's write message can either accept a filename or full pathname and you needn't do anything special to accept both.

```

void myobject_writefile(t_myobject *x, char *filename, short path)
{
    char *buf = "write me into a file";
    long err;
    t_filehandle fh;

    err = path_createsysfile(filename, path, 'TEXT', &fh);
    if (err)
        return;
    err = sysfile_writetextfile(fh, &buf, TEXT_LB_NATIVE);
    sysfile_close(fh);
}

```

Here is a data file variant of myobject_writefile(). It writes a small buffer of ten numbers to a file.

```

void myobject_writefile(t_myobject *x, char *filename, short path)
{
    char *buf[10];
    long count, i;
    long err;
    t_filehandle fh;

    // create some data

    for (i = 0; i < 10; i++)
        buf[i] = i + 1;

    count = 10;

    err = path_createsysfile(filename, path, 'TEXT', &fh);
    if (err)
        return;
    err = sysfile_write(fh, &count, buf);
    sysfile_close(fh);
}

```


Chapter 13

Scripting the Patcher

Your object can use scripting capabilities of the patcher to learn things about its context, such as the patcher's name, hierarchy, or the peer objects to your object in its patcher.

You can also modify a patcher, although any actions your object takes are not undoable and may not work in the runtime version.

13.1 Knowing the Patcher

To obtain the patcher object containing your object, you can use the obex hash table. The obex (for "object extensions") is, more generally, a way to store and recall data in your object. In this case, however, we are just using it in a read-only fashion.

Note that unlike the technique discussed in previous versions of the SDK, using the obex to find the patcher works at any time, not just in the new instance routine.

```
void myobject_getmypatcher(t_myobject **x)
{
    t_object *mypatcher;

    object_obex_lookup(x, gensym("#P"), &mypatcher);
    post("my patcher is at address %lx", mypatcher);
}
```

The patcher is an opaque Max object. To access data in a patcher, you'll use attributes and methods.

13.1.1 Patcher Name and File Path

To obtain the name of the patcher and its file path (if any), obtain attribute values as shown below.

```
t_symbol *name = object_attr_getsym(patcher, gensym("name"));
t_symbol *path = object_attr_getsym(patcher, gensym("filepath"));
```

These attributes may return NULL or empty symbols.

13.1.2 Patcher Hierarchy

To determine the patcher hierarchy above the patcher containing your object, you can use `jpatcher_getparentpatcher()`. A patcher whose parent is NULL is a top-level patcher. Here is a loop that prints the name of each parent patcher as you ascend the hierarchy.

```
t_object *parent, *patcher;
t_symbol *name;

object_obex_lookup(x, gensym("#P"), &patcher);
parent = patcher;
do {
    parent = jpatcher_getparentpatcher(parent);
    if (parent) {
        name = object_attr_getsym(parent, gensym("name"));
        if (name)
            post("%s", name->s_name)
    }
} while (parent != NULL);
```

13.1.3 Getting Objects in a Patcher

To obtain the first object in a patcher, you can use [jpatcher_get_firstobject\(\)](#). Subsequent objects are available with [jbox_get_nextobject\(\)](#).

If you haven't read the [Anatomy of a UI Object](#), we'll mention that the patcher does not keep a list of non-UI objects directly. Instead it keeps a list of UI objects called boxes, and the box that holds non-UI objects is called a newobj. The "objects" you obtain with calls such as [jpatcher_get_firstobject\(\)](#) are boxes. The [jbox_get_object\(\)](#) routine can be used to get the pointer to the actual object, whether the box is a UI object or a newobj containing a non-UI object. In the case of UI objects such as dials and sliders, the pointer returned by [jbox_get_object\(\)](#) will be the same as the box. But for non-UI objects, it will be different.

Here is a function that prints the class of every object (in a box) in a patcher containing an object.

```
void myobject_printpeers(t_myobject *x)
{
    t_object *patcher, *box, *obj;

    object_obex_lookup(x, gensym("#P"), &patcher);

    for (box = jpatcher_get_firstobject(patcher); box; box =
        jbox_get_nextobject(box)) {
        obj = jbox_get_object(box);
        if (obj)
            post("%s", object_classname(obj)->s_name);
        else
            post("box with NULL object");
    }
}
```

13.1.4 Iteration Using Callbacks

As an alternative to the technique shown above, you can write a callback function for use with the patcher's iteration service. The advantage of using iteration is that you can descend into the patcher hierarchy without needing to know the details of the various objects that may contain subpatchers (patcher, poly~, bpatcher, etc.). If you want to iterate only at one level of a patcher hierarchy, you can do that too.

Your iteration function is defined as follows. It will be called on every box in a patcher (and, if you specify, the patcher's subpatchers).

```
long myobject_iterator(t_myobject *x, t_object *b);
```

The function returns 0 if iteration should continue, or 1 if it should stop. This permits you to use an iterator as a way to search for a specific object.

Here is an example of using an iterator function:

```
t_object *patcher;
long result = 0;
t_max_err err;

err = object_obex_lookup(x, gensym("#P"), &patcher);

object_method(patcher, gensym("iterate"), myobject_iterator, (void *)x,
PI_WANTBOX | PI_DEEP, &result);
```

The [PI_WANTBOX](#) flag tells the patcher iterator that it should pass your iterator function the box, rather than the object contained in the box. The [PI_DEEP](#) flag means that the iteration will descend, depth first, into subpatchers. The result parameter returns the last value returned by the iterator. For example, if the

iterator terminates early by returning a non-zero value, it will contain that value. If the iterator function does not terminate early, result will be 0.

Assuming the iterator function receives boxes, here is an example iterator that prints out the class and scripting name (if any) of all of the objects in a patcher. Note that the scripting name is an attribute of the box, while the class we would like to know is of the object associated with the box.

```
long myobject_iterator(t_myobject *x, t_object *b)
{
    t_symbol *name = object_attr_getsym(b, gensym("varname"));
    t_symbol *cls = object_classname(jbox_get_object(b));

    if (name)
        post("%s (%s)", cls->s_name, name->s_name);
    else
        post("%s", cls->s_name);
    return 0;
}
```

13.2 Creating Objects

Much of the Max user interface is implemented using patcher scripting. For example, the inspectors are patchers in which an inspector object has been created. The file browser window has four or five separate scripted objects in it. Even the debug window is a dynamically scripted patcher. We point this out just to inform you that creating objects in a patcher actually works (if you get all the details right). The xxx example object shows how to use patcher scripting to create an "editing window" similar to the ones you see when double-clicking on a table or buffer~ object.

Creating objects in a patcher generally requires the use of a [Dictionary](#) (see discussion of UI objects above), but there is a convenience function [newobject_sprintf\(\)](#) that can be used to avoid some of the complexity.

To create an object, your task is to set some attributes. In the absence of any specific values, an object's attributes will be set to some default, but you'll probably care, at the very least, about specifying the object's location. Here is an example that creates a toggle and metro object using a combination of attribute parse syntax and sprintf. If you're interested in creating objects with [newobject_sprintf\(\)](#), it may help to examine a Max document to see some of the attribute name - value pairs used to specify objects.

```
t_object *patcher, *toggle, *metro;
t_max_err err;

err = object_obex_lookup(x, gensym("#P"), &patcher);

toggle = newobject_sprintf(patcher, "@maxclass toggle
@patching_position %.2f %.2f",
                           x->togxpos, x->togxpos);

metro = newobject_sprintf(patcher, "@maxclass newobj @text \"metro 400\"
@patching_position %.2f %.2f",
                           x->metxpos, x->metypos);
```

Note that to create a non-UI object, you use set the maxclass attribute to newobj and the text attribute to the contents of the object box. Attributes can be specified in any order. Using the patching_position attribute permits you to specify only the top-left corner and use the object's default size. For text objects, the default size is based on the default font for the patcher.

Finally, note that [newobject_sprintf\(\)](#) returns a pointer to the newly created box, not the newly created object inside the box. To get the object inside the box, use [jbox_get_object\(\)](#).

13.2.1 Connecting Objects

If you'd like to script the connections between two objects, you can do so via a message to the patcher. Assuming you have the patcher, toggle, and metro objects above, you'll create an array of atoms to send the message using [object_method_typed\(\)](#).

```
t_atom msg[4], rv;

atom_setobj(msg, toggle);      // source
atom_setlong(msg + 1, 0);       // outlet number (0 is leftmost)
atom_setobj(msg + 2, metro);    // destination
atom_setlong(msg + 3, 0);       // inlet number (0 is leftmost)

object_method_typed(patcher, gensym("connect"), 4, msg, &rv);
```

If you want to have a hidden connection, pass an optional fifth argument that is any negative number.

13.3 Deleting Objects

To delete an object in a patcher you call [object_free\(\)](#) on the box. As of Max 5.0.6 this will properly redraw the patcher and remove any connected patch cords.

13.4 Obtaining and Changing Patcher and Object Attributes

You can use object attribute functions to modify the appearance and behavior of objects in a patcher or the patcher itself. Note that only a few of these attributes can be modified by the user. The C level access to attributes is much more extensive.

Attributes whose type is object can be accessed via [object_attr_getobj\(\)](#) / [object_attr_setobj\(\)](#). Attributes whose type is char can be accessed with [object_attr_getchar\(\)](#) / [object_attr_setchar\(\)](#). Attributes whose type is long can be accessed with [object_attr_getlong\(\)](#) / [object_attr_setlong\(\)](#). Attributes whose type is symbol can be accessed via [object_attr_getsym\(\)](#) / [object_attr_setsym\(\)](#). For attributes that are arrays, such as colors and rectangles, use [object_attr_getvalueof\(\)](#) / [object_attr_setvalueof\(\)](#).

13.4.1 Patcher Attributes

Name	Type	Settable	Description
box	object	No	The box containing the patcher (NULL for top-level patcher)
locked	char	Yes (not in runtime)	Locked state of the patcher
presentation	char	Yes	Presentation mode of the patcher
openinpresentation	char	Yes	Will patcher open in presentation mode?
count	long	No	Number of objects in a patcher
fgcount	long	No	Number of objects in the patcher's foreground layer
bgcount	long	No	Number of objects in the patcher's background layer
numviews	long	No	Number of currently open views of the patcher
numwindowviews	long	No	Number of currently open window-based views of the patcher
firstobject	object	No	First box in the patcher
lastobject	object	No	Last box in the patcher
firstline	object	No	First patch cord in the patcher
firstview	object	No	First view object in the patcher
title	symbol	Yes	Window title
fulltitle	symbol	No	Complete title including "unlocked" etc.
name	symbol	No	Name (could be different from title)
filename	symbol	No	Filename
filepath	symbol	No	File path (platform-independent file path syntax)
fileversion	long	No	File version
noedit	char	No	Whether patcher can be unlocked
collective	object	No	Collective object, if patcher is inside a collective
cansave	char	No	Whether patcher can be saved
dirty	char	Yes (not in runtime)	Whether patcher is modified
bglocked	char	Yes	Whether background is locked
rect	double[4]	Yes	Patcher's rect (left, top, width, height)
Generated on Tue Jan 25 14:35:38 2011 for Max5 API by Doxygen			
defrect	double[4]	Yes	Patcher's default rect (used when opening the first view)
openrect	double[4]	Yes	Fixed initial window location
parentpatcher	object	No	Immediate parent

13.4.2 Box Attributes

Name	Type	Settable	Description
rect	double[4]	Settable only	Changes both patching_rect and presentation_rect
presentation_rect	double[4]	Yes	Presentation mode rect
patching_rect	double[4]	Yes	Patching mode rect
position	double[2]	Settable only	Changes both patching_position and presentation_position
size	double[2]	Settable only	Changes both patching_size and presentation_size
patching_position	double[2]	Yes	Patching mode position (top, left corner)
presentation_position	d[2]	Yes	Presentation mode position
patching_size	double[2]	Yes	Patching mode size (width, height)
presentation_size	double[2]	Yes	Presentation mode size
maxclass	symbol	No	Name of Max class (newobj for non-UI objects)
object	object	No	Associated object (equivalent to jbox_get_object)
patcher	object	No	Containing patcher
hidden	char	Yes	Is box hidden on lock?
fontname	symbol	Yes	Font name (if box has font attributes or a text field)
fontface	long	Yes	"Fake" font face (if box has font attribute or a text field)
fontsize	long	Yes	Font size (if box has font attributes or a text field)
textcolor	double[4]	Yes	Text color (if box has font attributes or a text field)
hint	symbol	Yes	Associated hint
color	double[4]	Yes	Standard color attribute (may not be present in all objects)
nextobject	object	No	Next object in the patcher's list
prevobject	object	No	Previous object in the patcher's list
varname	symbol	Yes	Scripting name
id	symbol	No	Immutable object ID (stored in files)
canhilite	char	No	Does this object accept focus?
background	char	Yes	Include in background
Generated on Tue Jan 25 14:35:38 2011 for Max5 API by Doxygen			
ignoreclick	char	Yes	Ignores clicks
maxfilename	symbol	No	Filename if class is external
description	symbol	No	Description used by assistance
drawfirstin	char	No	Is leftmost inlet

To access an attribute of a non-UI object, use [jbox_get_object\(\)](#) on the box to obtain the non-UI object first.

Chapter 14

Enhancements to Objects

14.1 Preset Support

Presets are a simple state-saving mechanism. Your object receives a preset message when state is being saved. You respond by creating a message that will be sent back to your object when the preset is recalled.

For more powerful and general state-saving, use the pattr system described below.

To support saving a single integer in a preset, you can use the [preset_int\(\)](#) convenience function. The [preset_int\(\)](#) function records an int message with the value you pass it in the preset, to be sent back to your object at a later time.

```
class_addmethod(c, (method)myobject_preset, "preset", 0);

void myobject_preset(t_myobject *x)
{
    preset_int(x, x->m_currentvalue);
}
```

More generally, you can use [preset_store\(\)](#). Here is an example of storing two values (m_xvalue and m_yvalue) in a list.

```
preset_store("ossl1", x, ob_sym(x), gensym("list"), x->m_xvalue, x->m_yvalue);
```

14.2 Pattr Support

In most cases, you need only to define your object's state using [Attributes](#) and it will be ready for use with Max's pattr system. For more complex scenarios you may also wish to investigate [object_notify\(\)](#), [object_attach\(\)](#), and the section on [Receiving Notifications](#).

14.3 Assistance

To show descriptions of your object's inlets and outlets while editing a patcher, your object can respond to the assist message with a function that copies the text to a string.

```
class_addmethod(c, (method)myobject_assist, "assist", A_CANT, 0);
```

The function below has two inlets and one outlet. The io argument will be 1 for inlets, 2 for outlets. The index argument will be 0 for the leftmost inlet or outlet. You can copy a maximum of 512 characters to the output string s. You can use [strncpy_zero\(\)](#) to copy the string, or if you want to format the assistance string based on a current value in the object, you could use [snprintf_zero\(\)](#).

```
void myobject_assist(t_myobject *x, void *b, long io, long index, char *s)
{
    switch (io) {
        case 1:
            switch (index) {
                case 0:
                    strncpy_zero(s, "This is a description of the leftmost inlet",
                                512);
                    break;
                case 1:
                    strncpy_zero(s, "This is a description of the rightmost inlet",
                                512);
                    break;
            }
    }
}
```

```

        }
        break;
    case 2:
        strncpy_zero(s, "This is a description of the outlet", 512);
        break;
    }
}

```

14.4 Hot and Cold Inlets

Objects such as operators (+, -, etc.) and the int object have inlets that merely store values rather than performing an operation and producing output. These inlets are labeled with a blue color to indicate they are "cold" rather than action-producing "hot" inlets. To implement this labeling, your object can respond to the inletinfo message.

```
class_addmethod(c, (method)myobject_inletinfo, "inletinfo", A_CANT, 0);
```

If all of your object's non-left inlets are "cold" you can use the function stdinletinfo() instead of writing your own, as shown below:

```
class_addmethod(c, (method)stdinletinfo, "inletinfo", A_CANT, 0);
```

To write your own function, just look at the index argument (which is 0 for the left inlet). This example turns the third inlet cold. You don't need to do anything for "hot" inlets.

```

void myobject_inletinfo(t_myobject *x, void *b, long index, char *t)
{
    if (index == 2)
        *t = 1;
}

```

14.5 Showing a Text Editor

Objects such as coll and text display a text editor window when you double-click. Users can edit the contents of the objects and save the updated data (or not). Here's how to do the same thing in your object.

First, if you want to support double-clicking on a non-UI object, you can respond to the dblclick message.

```

class_addmethod(c, (method)myobject_dblclick, "dblclick", A_CANT, 0);

void myobject_dblclick(t_myobject *x)
{
    // open editor here
}

```

You'll need to add a [t_object](#) pointer to your object's data structure to hold the editor.

```

typedef struct _myobject
{
    t_object m_obj;
    t_object *m_editor;
} t_myobject;

```

Initialize the m_editor field to NULL in your new instance routine. Then implement the dblclick method as follows:

```

if (!x->m_editor)
    x->m_editor = object_new(CLASS_NOBOX, gensym("jed"), (t_object *)x, 0);
else
    object_attr_setchar(x->m_editor, gensym("visible"), 1);

```

The code above does the following: If the editor does not exist, we create one by making a "jed" object and passing our object as an argument. This permits the editor to tell our object when the window is closed.

If the editor does exist, we set its visible attribute to 1, which brings the text editor window to the front.

To set the text of the edit window, we can send our jed object the settext message with a zero-terminated buffer of text. We also provide a symbol specifying how the text is encoded. For best results, the text should be encoded as UTF-8. Here is an example where we set a string to contain "Some text to edit" then pass it to the editor.

```

char text[512];

strcpy(text, "Some text to edit");
object_method(x->m_editor, gensym("settext"), text, gensym("utf-8"));

```

The title attribute sets the window title of the text editor.

```
object_attr_setsym(x->m_editor, gensym("title"), gensym("crazytext"));
```

When the user closes the text window, your object (or the object you passed as an argument when creating the editor) will be sent the edclose message.

```
class_addmethod(c, (method)myobject_edclose, "edclose", A_CANT, 0);
```

The edclose method is responsible for doing something with the text. It should also zero the reference to the editor stored in the object, because it will be freed. A pointer to the text pointer is passed, along with its size. The encoding of the text is always UTF-8.

```

void myobject_edclose(t_myobject *x, char **ht, long size)
{
    // do something with the text
    x->m_editor = NULL;
}

```

If your object will be showing the contents of a text file, you are still responsible for setting the initial text, but you can assign a file so that the editor will save the text data when the user chooses Save from the File menu. To assign a file, use the filename message, assuming you have a filename and path ID.

```
object_method(x->m_editor, gensym("filename"), x->m_myfilename, x->m_mypath
);
```

The filename message will set the title of the text editor window, but you can use the title attribute to override the simple filename. For example, you might want the name of your object to precede the filename:

```

char titlename[512];

sprintf(titlename, "myobject: %s", x->m_myfilename);
object_attr_setsym(x->m_editor, gensym("title"), gensym(titlename));

```

Each time the user chooses Save, your object will receive an edsav message. If you return zero from your edsav method, the editor will proceed with saving the text in a file. If you return non-zero, the editor assumes you have taken care of saving the text. The general idea is that when the user wants to save the text, it is either updated inside your object, updated in a file, or both. As an example, the js object uses its edsav message to trigger a recompile of the Javascript code. But it also returns 0 from its edsav method so that the text editor will update the script file. Except for the return value, the prototype of the edsav method is identical to the edclose method.

```
class_addmethod(c, (method)myobject_edsave, "edsave", A_CANT, 0);

long myobject_edsave(t_myobject *x, char **ht, long size)
{
    // do something with the text
    return 0;      // tell editor it can save the text
}
```

14.6 Accessing Data in table Objects

Table objects can be given names as arguments. If a table object has a name, you can access the data using [table_get\(\)](#). Supply a symbol, as well as a place to assign a pointer to the data and the length. The following example accesses a table called foo, and, if found, posts all its values.

```
long **data = NULL;
long i, size;

if (!table_get(gensym("foo"), &data, &size)) {
    for (i = 0; i < size; i++) {
        post("%ld: %ld", i, (*data)[i]);
    }
}
```

You can also write data into the table. If you would like the table editor to redraw after doing so, use [table_dirty\(\)](#). Here's an example where we set all values in the table to zero, then notify the table to redraw.

```
long **data = NULL;
long i, size;

if (!table_get(gensym("foo"), &data, &size)) {
    for (i = 0; i < size; i++) {
        (*data)[i] = 0;
    }
    table_dirty(gensym("foo"));
}
```

buffer~ support

Chapter 15

Data Structures

15.1 Linked Lists

The Max `t_linklist` data structure is useful for maintaining ordered lists of items where you want to be able to insert and delete items efficiently. Random access of individual items, however, gets appreciably slower as the list grows in size. The `t_linklist` is thread-safe by default, but thread safety can be turned off for performance benefits in single-threaded applications. However, ensure that your use of the linked list is truly single-threaded (based on an understanding of Max's [Threading](#) model) before turning off the thread safety features.

By default, the `t_linklist` holds pointers to Max objects. However, you can treat what the linklist holds as data rather than objects to be freed by using the `linklist_flags()` function.

Here is a simple example of the use of `t_linklist`. The code below stores five symbols, sorts them, searches for a specific item, deletes an item, prints all items, and then frees the entire structure. Since symbols in Max are never freed, `linklist_flags()` is used to specify that data, rather than object pointers, are being stored.

```
void mylistfun()
{
    t_linklist *list;

    list = (t_linklist *)linklist_new();

    linklist_flags(list, OBJ_FLAG_DATA);

    // add some data
    linklist_append(list, gensym("one"));
    linklist_append(list, gensym("two"));
    linklist_append(list, gensym("three"));
    linklist_append(list, gensym("four"));
    linklist_append(list, gensym("five"));

    // sort
    linklist_sort(list, (t_cmpfn)mysortfun);

    // search
    index = linklist_findfirst(list, &found, mysearchfun, gensym("four")); // fin
    d the "four" symbol

    if (index != -1) // found
        linklist_chuckindex(list, index);

    // iterate
    linklist_funall(list, myprintfun, NULL);

    // delete
    linklist_chuck(list);
}
```

The sorting function compares two items in the list and returns non-zero if the first one should go before the second one.

```
long mysortfun(void *a, void *b)
{
    t_symbol *sa = (t_symbol *)a;
    t_symbol *sb = (t_symbol *)b;

    return strcmp(sa->s_name, sb->s_name) > 0;
```

```
}
```

The search function is passed the final argument to [linklist_findfirst\(\)](#) and, in this case, just returns the symbol that matches, which is just testing for pointer equivalence since all Max symbols are unique. You could do more sophisticated searching if you store more complex data in a linklist.

```
long mysearchfun(t_symbol *elem, t_symbol *match)
{
    return elem == match;
}
```

The iteration function takes some action on all items in the list. The third argument to [linklist_funall\(\)](#) is passed as the second argument to your iteration function. In this example, we don't do anything with it.

```
void myprintfun(t_symbol *item, void *dummy)
{
    post ("%s", item->s_name);
}
```

There are many more functions for operating on linked lists you can read about by exploring the [Linked List](#) function reference.

15.2 Hash Tables

A hash table is a data structure that associates some data with a unique key. If you know the key, you can get back the data much more quickly than with a linked list, particularly as the number of items stored grows larger. The Max hash table [t_hashtab](#) is optimized to work with symbol pointers as keys, but you can use any pointer or number, as long as it is unique.

To create a [t_hashtab](#), you use [hashtab_new\(\)](#). To add items, use [hashtab_store\(\)](#). To find items that have been added, use [hashtab_lookup\(\)](#).

By contrast with linked lists and arrays, hash tables do not have a strong sense of ordering. You can iterate through all items using [hashtab_funall\(\)](#), but the exact order is not under your control as items are added and removed. There is also no way to "sort" a hash table.

Example:

The following example creates a hashtab, shows how to add some data (in this case, just a number), look it up, and delete the hashtab.

```
t_hashtab *tab = (t_hashtab *)hashtab_new(0);
long result, value;

hashtab_store(tab, gensym("a great number"), (t_object *)74);

result = hashtab_lookup(tab, gensym("a great number"), (t_object **)value);

if (!result)
    post("found the value and it is %ld", value);
else
    post("did not find the value");

hashtab_chuck(tab);
```

Note that the Max [t_dictionary](#) used for managing patcher data is implemented as a [t_hashtab](#).

Chapter 16

Threading

The Max systhread API has two main purposes.

First, it can be used to implement thread protection which works in conjunction with Max's existing threading model and is cross-platform. Thread protection prevents data corruption in the case of simultaneously executing threads in the same application. We'll discuss the Max threading model and show you a simple example of thread protection below, but you can often avoid the need to use thread protection by using one of the thread-safe [Data Storage](#) Max provides.

The second use of the systhread API is a cross-platform way to create and manage threads. This is an advanced feature that very few programmers will ever need. For information on creating and managing threads look at the systhread API header file.

16.1 Max Threading Operation

Please note that this description of how Max operates is subject to change and may not apply to future versions. For more information about the Max scheduler and low-priority queue, see the [The Scheduler](#) section.

Max (without audio) has two threads. The main or event thread handles user interaction, asks the system to redraw the screen, processes events in the low-priority queue. When not in Overdrive mode, the main thread handles the execution of events in the Max scheduler as well. When Overdrive is enabled, the scheduler is moved to a high-priority timer thread that, within performance limits imposed by the operating system, attempts to run at the precise scheduler interval set by user preference. This is usually 1 or 2 milliseconds.

The basic idea is to put actions that require precise timing and are relatively computationally cheap in the high-priority thread and computationally expensive events that do not require precise timing in the main thread. On multi-core machines, the high-priority thread may (or may not) be executing on a different core.

On both Mac and Windows, either the main thread or the timer thread can interrupt the other thread, even though the system priority level of the timer thread is generally much higher. This might seem less than optimal, but it is just how operating systems work. For example, if the OS comes to believe the Max timer thread is taking too much time, the OS may "punish" the thread by interrupting it with other threads, even if those threads have a lower system priority.

Because either thread can be interrupted by the other, it is necessary to use thread protection to preserve the integrity of certain types of data structures and logical operations. A good example is a linked list, which can be corrupted if a thread in the process of modifying the list is interrupted by another thread that tries to modify the list. The Max [t_linklist](#) data structure is designed to be thread-safe, so if you need such a data structure, we suggest you use [t_linklist](#). In addition, Max provides thread protection between the timer thread and the main thread for many of its common operations, such as sending messages and using outlets.

When we add audio into the mix (so to speak), the threading picture gets more complicated. The audio perform routine is run inside a thread that is controlled by the audio hardware driver. In order to eliminate excessive thread blocking and potential race conditions, the thread protection offered inside the audio perform routine is far less comprehensive, and as discussed in the MSP section of the API documentation, the only supported operation for perform routines to communicate to Max is to use a clock. This will trigger a function to run inside the Max scheduler.

The Max scheduler can be run in many different threading conditions. As explained above it can be run either in the main thread or the timer thread. When Scheduler in Audio Interrupt (SIAI) is enabled, the scheduler runs with an interval equal to every signal vector of audio inside the audio thread. However, if the Non-Real-Time audio driver is used, the audio thread is run inside the main thread, and if SIAI is enabled, the scheduler will also run inside the main thread. If not, it will run either in the main thread or the timer thread depending on the Overdrive setting. (Using the Non-Real-Time audio driver without SIAI will generally lead to unpredictable results and is not recommended.)

16.2 Thread Protection

The easiest method for thread protection is to use critical sections. A critical section represents a region of code that cannot be interrupted by another thread. We speak of entering and exiting a critical section, and use `critical_enter()` and `critical_exit()` to do so.

Max provides a default global critical section for your use. This same critical section is used to protect the timer thread from the main thread (and vice versa) for many common Max data structures such as outlets. If you call `critical_enter()` and `critical_exit()` with argument of 0, you are using this global critical section. Typically it is more efficient to use fewer critical sections, so for many uses, the global critical section is sufficient. Note that the critical section is recursive, so you if you exit the critical section from within some code that is already protected, you won't be causing any trouble.

16.2.1 When Messages Arrive

It's possible that a message sent to your object could interrupt the same message sent to your object ("my-object"). For example, consider the patch shown below. On the left is a bang button connected to the left inlet. On the right is a metro connected to the same inlet.

When a user clicks on the bang button, the message is sent to your object in the main thread. When Overdrive is enabled, the metro will send a bang message to your object in the timer thread. Either could interrupt the other. If your object performs operations on a data structure that cannot be interrupted, you should use thread protection.

16.2.2 Critical Section Example

Here is an example that uses the global critical section to provide thread protection for an array data structure. Assume we have an operation `array_read()` that reads data from an array, and `array_insert()` that inserts data into the same array. We wish to ensure that reading doesn't interrupt writing and vice versa.

```
long array_read(t_myobject *x, long index)
{
    critical_enter(0);
    result = x->m_data[index];
    critical_exit(0);
    return result;
}
```

Note that all paths of your code must exit the critical region once it is entered, or the other threads in Max will never execute.

```
long array_insert(t_myobject *x, long index, long value)
{
    critical_enter(0);
    // move existing data
    sysmem_copyptr(x->m_data + index, x->m_data + index + 1, (x->m_size - x->m_
    index) * sizeof(long));
    // write new data
    x->m_data[index] = value;
    critical_exit(0);
}
```


Chapter 17

Drag'n'Drop

The Max file browser permits you to drag files to a patcher window or onto objects to perform file operations.

Your object can specify the file types accepted as well as a message that will be sent when the user releases the mouse button with the file on top of the object. UI and non-UI objects use the same interface to drag'n'drop.

Example UI object: *pictmeter~*. Example non-UI: TBD.

Messages to support:

```
acceptsdrag_locked (A_CANT)
```

Sent to an object during a drag when the mouse is over the object in an unlocked patcher.

```
acceptsdrag_unlocked (A_CANT)
```

Sent to an object during a drag when the mouse is over the object in a locked patcher.

17.1 Discussion

Why two different scenarios? `acceptsdrag_unlocked()` can be thought of as an "editing" operation. For example, objects such as `pictslider` accept new image files for changing their appearance when the patcher is unlocked, but not when the patcher is locked. By contrast, `sfplay~` can accept audio files for playback in either locked or unlocked patchers, since that is something you can do with a message (rather than an editing operation that changes the patcher).

Message handler definitions:

```
long myobject_acceptsdrag_unlocked(t_myobject **x, t_object *drag, t_object *vi
                                    ew);
long myobject_acceptsdrag_locked(t_myobject **x, t_object *drag, t_object *view
                                  );
```

The handlers return true if the file(s) contained in the drag can be used in some way by the object. To test the filetypes, use `jdrag_matchdragrole()` passing in the drag object and a symbol for the file type. Here is list of pre-defined file types:

- audiofile
- imagefile
- moviefile
- patcher
- helpfile
- textfile

or to accept all files, use file

If `jdrag_matchdragrole()` returns true, you then describe the messages your object receives when the drag completes using `jdrag_object_add()`. You can add as many messages as you wish. If you are only adding a single message, use `jdrag_object_add()`. For more control over the process, and for adding more than one message, `jdrag_add()` can be used. If you add more than one message, the user can use the option key to specify the desired action. By default, the first one you add is used. If there are two actions, the

option key will cause the second one to be picked. If there are more than two, a pop-up menu appears with descriptions of the actions (as passed to `jdrag_add()`), and the selected action is used.

Example:

This code shows how to respond to an audiofile being dropped on your object by having the `read` message sent.

```
if (jdrag_matchdragrole(drag, gensym("audiofile"), 0)) {
    jdrag_object_add(drag, (t_object *)x, gensym("read"));
    return true;
}
return false;
```

Your `acceptsdrag` handler can test for multiple types of files and add different messages.

Chapter 18

ITM

ITM is the tempo-based timing system introduced with Max 5.

It allows users to express time in tempo-relative units as well as milliseconds, samples, and an ISO 8601 hour-minute-second format. In addition, ITM supports one or more transports, which can be synchronized to external sources. An ITM-aware object can schedule events to occur when the transport reaches a specific time, or find out the current transport state.

The ITM API is provided on two different levels. The time object ([t_timeobject](#)) interface provides a higher-level way to parse time format information and schedule events. In addition, you can use lower-level routines to access ITM objects ([t_itm](#)) directly. An ITM object is responsible for maintaining the current time and scheduling events. There can be multiple ITM objects in Max, each running independently of the others.

18.1 Scheduling Temporary Events

There are two kinds of events in ITM. Temporary events are analogous to Max clock objects in that they are scheduled and fire at a dynamically assigned time. Once they have executed, they are removed from the scheduler. Permanent events always fire when the transport reaches a specific time, and are not removed from the scheduler. The ITM-aware metro is an example of an object that uses temporary events, while the timepoint object uses permanent events. We'll show how to work both types using an example included in the SDK called `delay2`. The existing Max delay object provides this capability, but this example shows most of the things you can do with the time object interface. To see the complete object, look at the *delay2 example*. We'll introduce a simpler version of the object, then proceed to add the quantization and the additional outlet that generates a delayed bang based on low-level ITM calls.

The ITM time object API is based on a Max object you create that packages up common ways you will be using ITM, including attribute support, quantization, and, if you want it, the ability to switch between traditional millisecond-based timing and tempo-based timing using an interface that is consistent with the existing Max objects such as metro and delay. (If you haven't familiarized yourself with attributes, you may want to read through the discussion about them in [Attributes](#) before reading further.)

To use the time object, you'll first need to provide some space in your object to hold a pointer to the object(s) you'll be creating.

```
typedef struct _delay2simple
{
    t_object m_ob;
    t_object *m_timeobj;
    void *m_outlet;
} _delay2simple;
```

Next, in your main routine, you'll create attributes associated with the time object using the [class_time_addattr\(\)](#) function.

```
class_time_addattr(c, "delaytime", "Delay Time", TIME_FLAGS_TICKSONLY |
TIME_FLAGS_USECLOCK | TIME_FLAGS_TRANSPORT);
```

The second argument, "delaytime", is a string that names the attribute. Users of your object will be able to change the delay value by sending a `delaytime` message. "Delay Time" is the label users see for the attribute in the inspector. The flags argument permits you to customize the type of time object you'd like. [TIME_FLAGS_TICKSONLY](#) means that the object can only be specified in tempo-relative units. You would not use this flag if you want the object to use the regular Max scheduler if the user specifies an absolute time (such as milliseconds). [TIME_FLAGS_USECLOCK](#) means that it is a time object that will actually schedule events. If you do not use this flag, you can use the time object to hold and convert time values, which you use to schedule events manually. [TIME_FLAGS_TRANSPORT](#) means that an additional attribute for specifying the transport name is added to your object automatically (it's called

"transport" and has the label "Transport Name"). The combination of flags above is appropriate for an object that will be scheduling events on a temporary basis that are only synchronized with the transport and specified in tempo-relative units.

The next step is to create a time object in your new instance routine using time_new. The time_new function is something like clock_new -- you pass it a task function that will be executed when the scheduler reaches a certain time (in this case, delay2simple_tick, which will send out a bang). The first argument to time_new is a pointer to your object, the second is the name of the attribute created via class_time_addattr, the third is your task function, and the fourth are flags to control the behavior of the time object, as explained above for class_time_addattr.

Finally, we use time_setvalue to set the initial delay value to 0.

```
void *delay2simple_new()
{
    t_delay2simple *x;
    t_atom a;

    x = (t_delay2simple *)object_alloc(s_delay2simple_class);
    x->m_timeobj = (t_object *)time_new((t_object *)x, gensym("delaytime"), (
method)delay2simple_tick, TIME_FLAGS_TICKSONLY | TIME_FLAGS_USECLOCK);
    x->m_outlet = bangout((t_object *)x);
    atom_setfloat(&a, 0.);
    time_setvalue(x->d_timeobj, NULL, 1, &a);
    return x;
}
```

To make a delayed bang, we need a delay2simple_bang function that causes our time object to put its task function into the ITM scheduler. This is accomplished using time_schedule. Note that unlike the roughly equivalent clock_fdelay, where the delay time is an argument, the time value must already be stored inside the time object using time_setvalue. The second argument to time_schedule is another time object that can be used to control quantization of an event. Since we aren't using quantization in this simple version of delay2, we pass NULL.

```
void delay2simple_bang(t_delay2 *x)
{
    time_schedule(x->d_timeobj, NULL);
}
```

Next, our simple task routine, delay2simple_tick. After the specified number of ticks in the time object has elapsed after the call to time_schedule, the task routine will be executed.

```
void delay2_tick(t_delay2 *x)
{
    outlet_bang(x->d_outlet);
}
```

Now let's add the two more advanced features found in delay2: quantization and a second (unquantized) bang output using low-level ITM routines. Here is the delay2 data structure. The new elements are a proxy (for receiving a delay time), a time object for quantization (d_quantize), a clock to be used for low-level ITM scheduling, and an outlet for the use of the low-level clock's task.

```
typedef struct delay2
{
    t_object d_obj;
    void *d_outlet;
    void *d_proxy;
    long d_inletnum;
    t_object *d_timeobj;
    t_object *d_outlet2;
    t_object *d_quantize;
    void *d_clock;
} t_delay2;
```

In the initialization routine, we'll define a quantization time attribute to work in conjunction with the d_quantize time object we'll be creating. This attribute does not have its own clock to worry about. It just holds a time value, which we specify will only be in ticks (quantizing in milliseconds doesn't make sense in the ITM context). If you build delay2 and open the inspector, you will see time attributes for both Delay Time and Quantization.

```
class_time_addattr(c, "quantize", "Quantization", TIME_FLAGS_TICKSONLY);
```

Here is part of the revised delay2 new instance routine. It now creates two time objects, plus a regular clock object.

```
x->d_inletnum = 0;
x->d_proxy = proxy_new(x, 1, &x->d_inletnum);
x->d_outlet2 = bangout(x);
x->d_outlet = bangout(x);

x->d_timeobj = (t_object*) time_new((t_object *)x, gensym("delaytime"), (
    method)delay2_tick, TIME_FLAGS_TICKSONLY | TIME_FLAGS_USECLOCK);
x->d_quantize = (t_object*) time_new((t_object *)x, gensym("quantize"), NULL,
    TIME_FLAGS_TICKSONLY);
x->d_clock = clock_new((t_object *)x, (method)delay2_clocktick);
```

To use the quantization time object, we can pass it as the second argument to time_schedule. If the value of the quantization is 0, there is no effect. Otherwise, time_schedule will move the event time so it lies on a quantization boundary. For example, if the quantization value is 4n (480 ticks), the delay time is 8n (240 ticks) and current time is 650 ticks, the delay time will be adjusted so that the bang comes out of the delay2 object at 980 ticks instead of 890 ticks.

In addition to using quantization with time_schedule, delay2_bang shows how to calculate a millisecond equivalent for an ITM time value using itm_ticksoms. This delay value is not quantized, although you read the time value from the d_quantize object and calculate your own quantized delay if wanted. The "calculated" delay is sent out the right outlet, since the clock we created uses delay2_clocktick.

```
void delay2_bang(t_delay2 **x)
{
    double ms, tix;

    time_schedule(x->d_timeobj, x->d_quantize);

    tix = time_getticks(x->d_timeobj);
    tix += (tix / 2);
    ms = itm_ticksoms(time_getitm(x->d_timeobj), tix);
    clock_fdelay(x->d_clock, ms);
}

void delay2_clocktick(t_delay2 **x)
{
    outlet_bang(x->d_outlet2);
}
```

18.2 Permanent Events

A permanent event in ITM is one that has been scheduled to occur when the transport reaches a specific time. You can schedule a permanent event in terms of ticks or bars/beats/units. An event based in ticks will occur when the transport reaches the specified tick value, and it will not be affected by changes in time signature. An event specified for a time in bars/beats/units will be affected by the time signature. As an example, consider an event scheduled for bar 2, beat 1, unit 0. If the time signature of the ITM object on which the event has been scheduled is 3/4, the event will occur at 480 times 3 or 1440 ticks. But if the

time signature is 4/4, the event will occur at 1920 ticks. If, as an alternative, you had scheduled the event to occur at 1920 ticks, setting the time signature to 3/4 would not have affected when it occurred.

You don't "schedule" a permanent event. Once it is created, it is always in an ITM object's list of permanent events. To specify when the event should occur, use `time_setvalue`.

The high-level time object interface handles permanent events. Let's say we want to have a time value called "targettime." First, we declare an attribute using `class_time_addattr`. The flags used are `TIME_FLAGS_TICKSONLY` (required because you can't specify a permanent event in milliseconds), `TIME_FLAGS_LOCATION` (which interprets the bar/beat/unit times where 1 1 0 is zero ticks), `TIME_FLAGS_PERMANENT` (for a permanent event), and `TIME_FLAGS_TRANSPORT` (which adds a transport attribute permitting a user to choose a transport object as a destination for the event) and `TIME_FLAGS_POSITIVE` (constrains the event to happen only for positive tick and bar/beat/unit values).

```
class_time_addattr(c, "targettime", "Target Time", TIME_FLAGS_TICKSONLY |
    TIME_FLAGS_LOCATION | TIME_FLAGS_PERMANENT | TIME_FLAGS_TRANSPORT |
    TIME_FLAGS_POSITIVE);
```

The `TIME_FLAGS_TRANSPORT` flag is particularly nice. Without any intervention on your part, it creates a transport attribute for your object, and takes care of scheduling the permanent event on the transport the user specifies, with a default value of the global ITM object. If you want to cause your event to be rescheduled dynamically when the user changes the transport, your object can respond to the `reschedule` message as follows.

```
class_addmethod(c, (method)myobject_reschedule, "reschedule", A_CANT, 0);           /
/ for dynamic transport reassignment
```

All you need to do in your `reschedule` method is just act as if the user has changed the time value, and use the current time value to call `time_setvalue`.

In your new instance routine, creating a permanent event with `time_new` uses the same flags as were passed to `class_time_addattr`:

```
x->t_time = (t_object*) time_new((t_object *)x, gensym("targettime"), (method)
    myobject_tick, TIME_FLAGS_TICKSONLY | TIME_FLAGS_USECLOCK | TIME_FLAGS_PERMANENT
    | TIME_FLAGS_LOCATION | TIME_FLAGS_POSITIVE);
```

The task called by the permanent time object is identical to a clock task or an ITM temporary event task.

18.3 Cleaning Up

With all time objects, both permanent and temporary, it's necessary to free the objects in your object's `free` method. Failure to do so will lead to crashes if your object is freed but its events remain in the ITM scheduler. For example, here is the `delay2` `free` routine:

```
void delay2_free(t_delay2 **)
{
    freeobject(x->d_timeobj);
    freeobject(x->d_quantize);
    freeobject((t_object *) x->d_proxy);
    freeobject((t_object *)x->d_clock);
}
```


Chapter 19

Jitter Object Model

19.1 Jitter Object Model Basics

Jitter objects use an object model which is somewhat different than the one traditionally used for developing Max external objects. The first big difference between Jitter objects and traditional Max external objects is that Jitter objects don't have any notion of the patcher themselves. This allows for the flexible instantiation and use of Jitter objects from C, Java, JavaScript, as well as in the Max patcher. The use of these Jitter objects is exposed to the patcher with a Max "wrapper" object, which will be discussed in the following chapter.

In this chapter we'll restrict our discussion to the fundamentals of defining the Jitter object which can be used in any of these languages. While Jitter's primary focus is matrix processing and real-time graphics, these tasks are unrelated to the object model, and will be covered in later chapters on developing Matrix Operator (MOP) and OB3D objects. Like Max objects, Jitter objects are typically written in C. While C++ can be used to develop Jitter objects, none of the object oriented language features will be used to define your object as far as Jitter is concerned. Similar to C++ or Java objects, Jitter objects are defined by a class with methods and member variables - we will refer to the member variables as "attributes". Unlike C++ or Java, there are no language facilities that manage class definition, class inheritance, or making use of class instances. In Jitter this must all be managed with sets of standard C function calls that will define your class, exercise methods, and get and set object attributes.

Max and Jitter implement their object models by maintaining a registry of ordinary C functions and struct members that map to methods and attributes associated with names. When some other code wishes to make use of these methods or attributes, it asks the Jitter object to look up the method or attribute in its registry based on a name. This is called dynamic binding, and is similar to Smalltalk or Objective C's object model. C++ and Java typically make use of static binding — i.e. methods and member variables are resolved at compile time rather than being dynamically looked up at run time.

19.2 Defining a Jitter Class

A Jitter class is typically defined in a C function named something like `your_object_name_init()`. Class definition begins with a call to `jit_class_new()`, which creates a new class associated with a specified name, constructor, destructor, and size in bytes of the object as stored in a C structure. This is followed by calls to `jit_class_addmethod()` and `jit_class_addattr()`, which register methods and attributes with their corresponding names in the class. The class is finally registered with a call to `jit_class_register()`. A minimal example class definition is shown below:

```
typedef struct _jit_foo
{
    t_jit_object      ob;
    float             myval;
} t_jit_foo;

static t_jit_class *_jit_foo_class=NULL;

t_jit_err jit_foo_init(void)
{
    long attrflags=0;
    t_jit_object *attr;

    // create new class named "jit_foo" with constructor + destructor
    _jit_foo_class = jit_class_new("jit_foo", (method) jit_foo_new,
        (method) jit_foo_free, sizeof(t_jit_foo), 0L);

    // add method to class
    jit_class_addmethod(jit_foo_scream, "scream", A_DEFLONG, 0L);

    // define attribute
```

```

attr = jit_object_new(      // instantiate an object
    _jit_sym_jit_attr_offset, // of class jit_attr_offset
    "myval",                // with name "myval"
    _jit_sym_float32,       // type float32
    attrflags,              // default flags
    (method)0L,              // default getter accessor
    (method)0L,              // default setter accessor
    calcoffset(t_jit_foo,myval)); // byte offset to struct member

// add attribute object to class
jit_class_addattr(_jit_foo_class, attr);

// register class
jit_class_register(_jit_foo_class);

return JIT_ERR_NONE;
}

// constructor
t_jit_foo *jit_foo_new(void)
{
    t_jit_foo *x;

    // allocate object
    if (x= jit_object_alloc(_jit_foo_class))
    {
        // if successful, perform any initialization
        x->myval = 0;
    }
    return x;
}

// destructor
void jit_foo_free(t_jit_foo *x)
{
    // would free any necessary resources here
}

// scream method
void jit_foo_scream(t_jit_foo *x, long i)
{
    post("MY VALUE IS %f! AND MY ARGUMENT IS %d", x->myval, i);
}

```

The above example has a constructor, `jit_foo_new()`; a destructor, `jit_foo_free()`; one 32 bit floating point attribute, `myval`, a member of the object struct accessed with default accessor methods; and a method `jit_foo_scream()`, which posts the current value of `myval` to the Max window.

19.3 Object Struct

Each instance of an object occupies some region of organized memory. The C structure that defines this organization of memory is typically referred to as the "object struct". It is important that the object struct always begin with an entry of type `t_jit_object`. It is within the `t_jit_object` where special information about the class is kept. The C structure can contain additional information, either exposed as attributes or not, but it is important that the size of the object struct does not exceed 16384 bytes. This means that it is not safe to define a large array as a struct entry if it will cause the size of the object struct to be larger than this limit. If additional memory is required, the object struct should contain a pointer to memory allocated from within the constructor, and freed within the destructor.

The class registration in the above code makes use of the object struct both to record in the class how large each object instance should be—i.e. `sizeof(t_jit_foo)`; and at what byte offset in the object struct an

attribute is located—i.e. `calcoffset(t_jit_foo, myval)`. When methods of an object are called, the instance of the object struct is passed as the first argument to the C functions which define the object methods. This instance may be thought of as similar to the "this" keyword used in C++ and Java - actually the C++ and Java underlying implementation works quite similarly to what has been implemented here in pure C. Object struct entries may be thought of as similar to object member variables, but methods must be called via functions rather than simply dereferencing instances of the class as you might do in C++ or Java. The list of object methods and other class information is referenced by your object's `t_jit_object` entry.

19.4 Constructor/Destructor

The two most important methods that are required for all objects are the constructor and the destructor. These are typically named `your_object_name_new()`, and `your_object_name_free()`, respectively. It is the constructor's responsibility to allocate and initialize the object struct and any additional resources the object instance requires. The object struct is allocated via `jit_object_alloc()`, which also initializes the `t_jit_object` struct entry to point at your relevant class information. The class information resides in your global class variable, e.g. `_jit_foo_class`, which you pass as an argument to `jit_object_alloc()`. This allocation does not, however initialize the other struct entries, such as "myval", which you must explicitly initialize if your allocation is successful. Note that because the constructor allocates the object instance, no object instance is passed as the first argument to the function which defines the constructor, unlike other object methods.

The constructor also has the option of having a typed argument signature with the same types as defined in the Writing Max Externals documentation—i.e. `A_LONG`, `A_FLOAT`, `A_SYM`, `A_GIMME`, etc. Typically, Jitter object constructors either have no arguments or use the `A_GIMME` typed argument signature.

In earlier versions of Jitter, the constructors were often specified as private and "untyped" using the `A_CANT` type signature. While this obsolete style of an untyped constructor will work for the exposure of a Jitter class to the patcher and C, it is now discouraged, as there must be a valid type signature for exposure of a class to Javascript or Java, though that signature may be the empty list.

It is the destructor's responsibility to free any resources allocated, with the exception of the object struct itself. The object struct is freed for you after your destructor exits.

19.5 Methods

You can define additional methods using the `jit_class_admmethod()` function. This example defines the `scream` method associated with the function `jit_foo_scream()`, with no additional arguments aside from the standard first argument of a pointer to the object struct. Just like methods for ordinary Max objects, these methods could have a typed argument signature with the same types as defined in the Writing Max Externals documentation — i.e. `A_LONG`, `A_FLOAT`, `A_SYM`, `A_GIMME`. Typically in Jitter objects, public methods are specified either without arguments, or use `A_GIMME`, or the low priority variants, `A_DEFER_LOW`, or `A_USURP_LOW`, which will be discussed in following chapters. Private methods, just like their Max equivalent should be defined as untyped, using the `A_CANT` type signature. Object methods can be called from C either by calling the C function directly, or by using `jit_object_method()` or `jit_object_method_typed()`. For example, the following calls that relate to the above `jit_foo` example are equivalent:

```
// call scream method directly
jit_foo_scream(x, 74);

// dynamically resolve and call scream method
jit_object_method(x, gensym("scream"), 74);

// dynamically resolve and call scream method with typed atom arguments
```

```
t_atom a[1];
jit_atom_setlong(a, 74);
jit_object_method_typed(x, gensym("scream"), 1, a, NULL);
```

What the `jit_object_method()` and `jit_object_method_typed()` functions do is look up the provided method symbol in the object's class information, and then calls the corresponding C function associated with the provided symbol. The difference between `jit_object_method()` and `jit_object_method_typed()` is that `jit_object_method()` will not require that the method is typed and public, and blindly pass all of the arguments following the method symbol on to the corresponding method. For this reason, it is required that you know the signature of the method you are calling, and pass the correct arguments. This is not type checked at compile time, so you must be extremely attentive to the arguments you pass via `jit_object_method()`. It is also possible for you to define methods which have a typed return value with the `A_GIMMEBACK` type signature. When calling such methods, the final argument to `jit_object_method_typed()`, should point to a `t_atom` to be filled in by the callee. This and the subject of "typed wrappers" for exposing otherwise private methods to language bindings that require typed methods (e.g. Java!/JavaScript) will be covered in a later chapter.

19.6 Attributes

You can add attributes to the class with `jit_class_addattr()`. Attributes themselves are Jitter objects which share a common interface for getting and setting values. While any class which conforms to the attribute interface could be used to define attributes of a given class, there are a few common classes which are currently used: `jit_attr_offset()`, which specifies a scalar attribute of a specific type (char, long, float32, float64, symbol, or atom) at some byte offset in the object struct; `jit_attr_offset_array()` which specifies an array (vector) attribute of a specific type (char, long, float32, float64, symbol, or atom) at some byte offset in the object struct; and `jit_attribute`, which is a more generic attribute object that can be instantiated on a per object basis. We will not document the usage of `jit_attribute` at this time. The constructor for the class `jit_attr_offset()` has the following prototype:

```
t_jit_object *jit_attr_offset_new(char *name, t_symbol *type, long flags,
                                 method mget, method mset, long offset);
```

When this constructor is called via `jit_object_new()`, additionally the class name, `_jit_sym_jit_attr_offset` (a global variable equivalent to `gensym("jit_attr_offset")`) must be passed as the first parameter, followed by the above arguments, which are passed on to the constructor. The name argument specifies the attribute name as a null terminated C string. The type argument specifies the attribute type, which may be one of the following symbols: `_jit_sym_char`, `_jit_sym_long`, `_jit_sym_float32`, `_jit_sym_float64`, `_jit_sym_symbol`, `_jit_sym_atom`, `_jit_sym_object`, or `_jit_sym_pointer`. The latter two are only useful for private attributes as these types are not exposed to, or converted from Max message atom values.

The flags argument specifies the attribute flags, which may be a bitwise combination of the following constants:

<code>#define JIT_ATTR_GET_OPAQUE</code>	<code>0x00000001 // cannot query</code>
<code>#define JIT_ATTR_SET_OPAQUE</code>	<code>0x00000002 // cannot set</code>
<code>#define JIT_ATTR_GET_OPAQUE_USER</code>	<code>0x00000100 // user cannot query</code>
<code>#define JIT_ATTR_SET_OPAQUE_USER</code>	<code>0x00000200 // user cannot set</code>
<code>#define JIT_ATTR_GET_DEFER</code>	<code>0x00010000 // (deprecated)</code>
<code>#define JIT_ATTR_GET_USURP</code>	<code>0x00020000 // (deprecated)</code>
<code>#define JIT_ATTR_GET_DEFER_LOW</code>	<code>0x00040000 // query in low priority</code>
<code>#define JIT_ATTR_GET_USURP_LOW</code>	<code>0x00080000 // query in low, usurping</code>
<code>#define JIT_ATTR_SET_DEFER</code>	<code>0x01000000 // (deprecated)</code>
<code>#define JIT_ATTR_SET_USURP</code>	<code>0x02000000 // (deprecated)</code>
<code>#define JIT_ATTR_SET_DEFER_LOW</code>	<code>0x04000000 // set at low priority</code>
<code>#define JIT_ATTR_SET_USURP_LOW</code>	<code>0x08000000 // set at low, usurping</code>

Typically attributes in Jitter are defined with flags `JIT_ATTR_GET_DEFER_LOW`, and `JIT_ATTR_SET_USURP_LOW`. This means that multiple queries from the patcher will generate a response for each query, and that multiple attempts to set the value at high priority will collapse into a single call with the last received value. For more information on defer and usurp, see the chapter on Jitter scheduling issues.

The mget argument specifies the attribute "getter" accessor method, used to query the attribute value. If this argument is zero (NULL), then the default getter accessor will be used. If you need to define a custom accessor, it should have a prototype and form comparable to the following custom getter:

```
t_jit_err jit_foo_myval_get(t_jit_foo **x, void *attr, long *ac, t_atom **av)
{
    if ((*ac)&&(*av)) {
        //memory passed in, use it
    } else {
        //otherwise allocate memory
        *ac = 1;
        if (!(*av = jit_getbytes(sizeof(t_atom) * (*ac)))) {
            *ac = 0;
            return JIT_ERR_OUT_OF_MEM;
        }
    }
    jit_atom_setfloat(*av,x->myval);

    return JIT_ERR_NONE;
}
```

Note that getters require memory to be allocated, if there is not memory passed into the getter. Also the attr argument is the class' attribute object and can be queried using `jit_object_method()` for things like the attribute flags, names, filters, etc.. The mset argument specifies the attribute "setter" accessor method, used to set the attribute value. If this argument is zero (NULL), then the default setter accessor will be used. If we need to define a custom accessor, it should have a prototype and form comparable to the following custom setter:

```
t_jit_err jit_foo_myval_set(t_jit_foo **x, void *attr, long ac, t_atom *av)
{
    if (ac&&av) {
        x->myval = jit_atom_getfloat(av);
    } else {
        // no args, set to zero
        x->myval = 0;
    }
    return JIT_ERR_NONE;
}
```

The offset argument specifies the attribute's byte offset in the object struct, used by default getters and setters to automatically query and set the attribute's value. If you have both custom accessors, this value is ignored. This can be a useful strategy to employ if you wish to have an object attribute that does not correspond to any actual entry in your object struct. For example, this is how we implement the time attribute of jit.qt.movie — i.e. it uses a custom getter and setter which make QuickTime API calls to query and set the current movie time, rather than manipulating the object struct itself, where no information about movie time is actually stored. In such an instance, you should set this offset to zero.

After creating the attribute, it must be added to the Jitter class using the `jit_class_addattr()` function:

```
t_jit_err jit_class_addattr(void *c, t_jit_object *attr);
```

To put it all together: to define a `jit_attribute_offset()` with the custom getter and setter functions defined above, you'd make the following call:

```
long attrflags = JIT_ATTR_GET_DEFER_LOW | JIT_ATTR_SET_USURP_LOW;
t_jit_object *attr = jit_object_new(_jit_sym_jit_attr_offset, "myval",
    _jit_sym_float32, attrflags,
    (method) jit_foo_myval_get, (method) jit_foo_myval_set, NULL);
jit_class_addattr(_jit_foo_class, attr);
```

And to define a completely standard jit_attribute_offset(), using the default getter and setter methods:

```
long attrflags = JIT_ATTR_GET_DEFER_LOW | JIT_ATTR_SET_USURP_LOW;
t_jit_object *attr = jit_object_new(_jit_sym_jit_attr_offset, "myval",
    _jit_sym_float32, attrflags,
    (method) NULL, (method) NULL, calcoffset(t_jit_foo, myval));
jit_class_addattr(_jit_foo_class, attr);
```

19.7 Array Attributes

Attributes can, in addition to referencing single values, also refer to arrays of data. The class jit_attribute_offset_array is used in this instance. The constructor for the class jit_attr_offset_array() has the following prototype:

```
t_jit_object *jit_attr_offset_array_new(char *name, t_symbol *type, long size,
    long flags, method mget, method mset, long offsetcount, long offset);
```

When this constructor is called via [jit_object_new\(\)](#), additionally the class name, [_jit_sym_jit_attr_offset_array\(\)](#) (a global variable equivalent to gensym("jit_attr_offset_array")) must be passed as the first parameter, followed by the above arguments, which are passed on to the constructor.

The name, type, flags, mget, mset and offset arguments are identical to those specified above.

The size argument specifies the maximum length of the array (the allocated size of the array in the Jitter object struct). The offsetcount specifies the byte offset in the object struct, where the actual length of the array can be queried/set. This value should be specified as a long. This value is used by default getters and setters when querying and setting the attribute's value. As with the jit_attr_offset object, if you have both custom accessors, this value is ignored.

The following sample listing demonstrates the creation of a simple instance of the jit_attr_offset_array() class for an object defined as:

```
typedef struct _jit_foo
{
    t_jit_object      ob;
    long              myarray[10]; // max of 10 entries in this array
    long              myarraycount; // actual number being used
} t_jit_foo;

long attrflags = JIT_ATTR_GET_DEFER_LOW | JIT_ATTR_SET_USURP_LOW;
t_jit_object *attr = jit_object_new(_jit_sym_jit_attr_offset_array, "myarray",
    _jit_sym_long, 10, attrflags, (method)0L, (method)0L,
    calcoffset(t_jit_foo, myarraycount), calcoffset(t_jit_foo, myarray));
jit_class_addattr(_jit_foo_class, attr);
```

19.8 Attribute Notification

Although the subject of object registration and notification will be covered in greater depth in a forthcoming chapter, it bears noting that attributes of all types (e.g. jit_attr_offset, jit_attr_offset_array and jit_attribute) will, if registered, automatically send notifications to all attached client objects, each time the attribute's value is set.

Chapter 20

Jitter Max Wrappers

20.1 Max Wrapper Classes

In order to expose the Jitter object to the Max patcher, a Max "wrapper" class must be defined. For simple classes, this is largely facilitated by a handful of utility functions that take a Jitter class and create the appropriate wrapper class with default functionality. However, there are occasions which warrant additional intervention to achieve special behavior, such as the use of additional inlets and outlets, integrating with MSP, converting matrix information to and from Max lists, etc. The first Max wrapper class we'll demonstrate won't have any extra complication beyond simply containing a basic Jitter class.

In general it is preferable to design the Jitter class so that it knows nothing about the Max patcher, and that any logic necessary to communicate with the patcher is maintained in the Max wrapper class. In situations where this might seem difficult, this can typically be accomplished by making special methods in the Jitter class that are only meant to be called by the Max wrapper, or by using Jitter's object notification mechanism, which we'll discuss in a future chapter. Below is the minimal Max wrapper class for the minimal Jitter class shown in the last chapter.

```

typedef struct _max_jit_foo
{
    t_object      ob;
    void *obex;
} t_max_jit_foo;

void *class_max_jit_foo;

void main()
{
    void *p,*q;

    // initialize the Jitter class
    jit_foo_init();

    // create the Max class as documented in Writing Max Externals
    setup(&class_max_jit_foo,
          (method) max_jit_foo_new,
          (method) max_jit_foo_free,
          (short)sizeof(t_max_jit_foo),
          0L, A_GIMME, 0);

    // specify a byte offset to keep additional information
    p = max_jit_classex_setup(calcoffset(t_max_jit_foo, obex));

    // look up the Jitter class in the class registry
    q = jit_class_findbyname(gensym("jit_foo"));

    // wrap the Jitter class with the standard methods for Jitter objects
    max_jit_classex_standard_wrap(p, q, 0);

    // add an inlet/outlet assistance method
    addmess((method)max_jit_foo_assist, "assist", A_CANT, 0);
}

void max_jit_foo_assist(t_max_jit_foo **x, void *b, long m, long a, char *s)
{
    // no inlet/outlet assistance
}

void max_jit_foo_free(t_max_jit_foo **x)
{
    // lookup the internal Jitter object instance and free
    jit_object_free(max_jit_obex_jitob_get(x));

    // free resources associated with the obex entry
    max_jit_obex_free(x);
}

```

```

void *max_jit_foo_new(t_symbol *s, long argc, t_atom *argv)
{
    t_max_jit_foo *x;
    long attrstart;
    void *o;

    // create the wrapper object instance based on the
    // max wrapper class, and the jitter class
    if (x = (t_max_jit_foo *)max_jit_obex_new(class_max_jit_foo,
                                                gensym("jit_foo")))
    {
        // add a general purpose outlet (rightmost)
        max_jit_obex_dumpout_set(x, outlet_new(x, 0L));

        // get normal args if necessary
        attrstart = max_jit_attr_args_offset(argc, argv);

        // instantiate Jitter object
        if (o = jit_object_new(gensym("jit_foo")))
        {
            // set internal jitter object instance
            max_jit_obex_jitob_set(x, o);

            // process attribute arguments
            max_jit_attr_args(x, argc, argv);
        }
        else
        {
            // couldn't instantiate, clean up and report an error
            freeobject((void *)x);
            x = NULL;
            error("jit.foo: out of memory");
        }
    }

    return (x);
}

```

20.2 Object Struct

The first thing you must do is define your Max class object struct. As is typical, for standard Max objects the first entry of the object struct must be of type [t_object](#); for UI objects, it must be of type [t_jbox](#); for MSP objects, it must be of type [t_pxobject](#); and for MSP UI objects, it must be of type [t_pxjbox](#). For more information on these different Max/MSP object types, please consult the Max/MSP developer documentation. Jitter objects can be wrapped within any of these object types.

You also need to define a pointer to point to extra information and resources needed to effectively wrap your Jitter class. This is typically referred to as the "obex" data, and it is where Jitter stores things like attribute information, the general purpose "dumpout", the internal Jitter object instance, Matrix Operator resources for inlets/outlets, and other auxiliary object information that is not required in a simple Max object. As of Max 4.5 there is also the facility for making use of such additional object information for ordinary Max objects. At the time of this writing, such information is provided in the Pattr developer documentation, as it is relevant to the definition of object attributes, which may be stored and operated upon by the patcher attribute suite of objects.

20.3 Defining Your Max Class

In your Max class registration, which takes place in your external's main function, you should begin by calling your Jitter class's registration function, typically named something like `your_object_name_init()`. Then you should proceed to define the Max class's constructor, destructor, object struct size, and typed arguments as is typically accomplished for Max objects via the setup function. In order for your wrapper class to be able to find the obex data, you need to specify a byte offset where this pointer is located within each object instance and allocate the resource in which this is stored in your Max class. This is accomplished with the `max_jit_classex_setup()` function. You should then look up the Jitter class via `jit_class_findbyname()`, and wrap it via the `max_jit_classex_standard_wrap()` function. The `max_jit_classex_standard_wrap()` function will add all typed methods defined in the Jitter class, as well getter and setter methods for attributes that are not opaque (i.e. private), and all the methods that are common to Jitter objects like getattributes, getstate, summary, importatrs, exportatrs, etc.

Now that you have wrapped the Jitter class, you can add any additional methods that you wish, such as your inlet/outlet assistance method, or something specific to the Max object. Like Jitter objects, you can also add methods which have defer or usurp wrappers, and these should be added via the `max_addmethod_defer_low()` or `max_addmethod_usurp_low()` functions, rather than simply using the traditional `addmess()` function. C

20.4 Constructor

Inside the Max object constructor, there are a few things which are different than building an ordinary Max external. If your object is to respond to attribute arguments, the constructor must be defined to take variable number of typed atom arguments, accomplished with the `A_GIMME` signature. You allocate your Max object with the `max_jit_obex_new()` function, instead of the traditional newobject function. You need to pass your Jitter class name to the `max_jit_obex_new()` function, which also allocates and initializes your obex data. If successful, you should proceed to add your general purpose "dumpout" outlet, used for returning attribute queries and other methods that provide information like `*jit.qt.movie*`'s framedump method's frame number or read method success code, with the `max_jit_object_dumpout_set()` function. If your object is a Matrix Operator that calls `max_jit_mop_setup_simple()` you will not need to explicitly call `max_jit_object_dumpout_set()`, as `max_jit_mop_setup_simple()` calls `max_jit_object_dumpout_set()` internally.

You then allocate your Jitter object with `jit_object_new()`, and store it in your obex data via `max_jit_obex_jitob_set()`. Note that this Jitter object instance can always be found with the function `max_jit_obex_jitob_get()`. If you wish, prior to allocating your Jitter object, you can look at your non-attribute arguments first — those arguments up to the location returned by `max_jit_attr_args_offset()` — and make use of them in your Jitter object constructor. It is typical to process attribute arguments after you've allocated both the Max and Jitter object instances, with `max_jit_attr_args()`, which is passed the Max object instance. If you wanted to use the attribute arguments somehow in your Jitter object constructor, you would need to parse the attribute arguments yourself. If you are not able to allocate your Jitter object (as is the case if you have run out of memory or if Jitter is present but not authorized), it is important that you clean up your Max wrapper object, and return NULL.

20.5 Destructor

In your Max object destructor, you additionally need to free your internal Jitter object with `jit_object_free()`, and free any additional obex data with `max_jit_obex_free()`. Matrix operators will typically require that `max_jit_mop_free()` is called, to free the resources allocated for matrix inputs and outputs. If your object has attached to a registered object for notification via `jit_object_attach()`, you should detach from

that object in your destructor using [jit_object_detach\(\)](#) to prevent invalid memory accesses as the registered object might attempt to notify the memory of a now freed object. Object registration and notification is discussed in further detail in following chapters.

20.6 Dumpout

The general purpose outlet, also known as "dumpout", is automatically used by the Max wrapper object when calling attribute getters and several of the standard methods like summary, or getattributes. It is also available for use in any other Max method you want, most easily accessed with the [max_jit_obex_dumpout\(\)](#) function that operates similar to [outlet_anything\(\)](#), but uses the max object pointer rather than the outlet pointer as the first argument. The outlet pointer which has been set in your constructor can be queried with the [max_jit_obex_dumpout_get\(\)](#) function, and used in the standard outlet calls. However, it is recommended for routing purposes that any output through the dumpout outlet is a message beginning with a symbol, rather than simply a bang, int, or float. Therefore, [outlet_anything\(\)](#) makes the most sense to use.

20.7 Additional inlets/outlets

To add additional inlets and outlets to your Max external, a few things should be noted. First, if your object is a Matrix Operator, matrix inlets and outlets will be added either through either the high level [max_jit_mop_setup_simple\(\)](#), or lower level [max_jit_mop_inputs\(\)](#) or [max_jit_mop_outputs\(\)](#) calls. These Matrix Operator functions will be covered in the chapter on Matrix Operators. Secondly, if your object is an MSP object, all signal inlets and outlets must be leftmost, and all non-signal inlets and outlets must be to the right of any single inlets or outlets—i.e. they cannot be intermixed. Lastly, additional inlets should use proxies (covered in detail in the Max/MSP developer documentation) so that your object knows which inlet a message has been received. This is accomplished with the [max_jit_obex_proxy_new\(\)](#) function. The inlet number is zero based, and you do not need to create a proxy for the leftmost inlet. Inside any methods which need to know which inlet the triggering message has been received, you can use the [max_jit_obex_inletnumber_get\(\)](#) function.

20.8 Max Wrapper Attributes

Sometimes you will need additional attributes which are specific to the Max wrapper class, but are not part of the internal Jitter class. Attributes objects for the Max wrapper class are defined in the same way as those for the Jitter class, documented in the previous chapter. However, these attributes are not added to the Max class with the [jit_class_addattr\(\)](#) function, but instead with the [max_jit_classex_addattr\(\)](#) function, which takes the classex pointer returned from [max_jit_classex_setup\(\)](#). Attribute flags, and custom getter and setter methods should be defined exactly as they would for the Jitter class.

Chapter 21

Matrix Operator QuickStart

The purpose of this chapter is to give a quick and high level overview of how to develop a simple Matrix Operator (MOP), which can process the matrix type most commonly used for video streams—i.e.

4 plane char data. For this task, we will use the jit.scalebias SDK example. More details such as how to make a Matrix Operator which deals with multiple types, plane count, dimensionality, inputs, outputs, etc. will appear in the following chapter. This chapter assumes familiarity with Jitter's multi-dimensional matrix representation and Matrix Operators used from the Max patcher, as discussed in the Jitter Tutorial, and as well as the preceding chapters on the Jitter object model and Max wrapper classes.

21.1 Defining the MOP Jitter Class

In the Jitter class definition, we introduce a few new concepts for Matrix Operators. In addition to the standard method and attribute definitions discussed in the Jitter object model chapter, you will want to define things like how many inputs and outputs the operator has, and what type, plane count, and dimension restrictions the operator has. These are accomplished by creating an instance of the `jit_mop` class, setting some state for the `jit_mop` object and adding this object as an adornment to your Jitter class. The following code segment references the `jit.scalebias` SDK example.

```
// create a new instance of jit_mop with 1 input, and 1 output
mop = jit_object_new(_jit_sym_jit_mop,1,1);

// enforce a single type for all inputs and outputs
jit_mop_single_type(mop,_jit_sym_char);

// enforce a single plane count for all inputs and outputs
jit_mop_single_planecount(mop,4);

// add the jit_mop object as an adornment to the class
jit_class_addadornment(_jit_scalebias_class,mop);
```

You create your `jit_mop` instance in a similar fashion to creating your attribute instances, using [jit_object_new\(\)](#). The `jit_mop` constructor has two integer arguments for inputs and outputs, respectively. By default, each MOP input and output is unrestricted in plane count, type, and dimension, and also are linked to the plane count, type, and dimensions of the first (i.e. leftmost) input. This default behavior can be overridden, and this simple 4 plane, char type, `jit.scalebias` example enforces the corresponding type and plane count restrictions via the [jit_mop_single_type\(\)](#) and [jit_mop_single_planecount\(\)](#) utility functions. For more information on the `jit_mop` class, please see the following chapter on MOP details and the Jitter API reference.

Once you have created your `jit_mop` instance, and configured it according to the needs of your object, you add it as an adornment to your Jitter class with the `jit_class_add_adornment()` function. Adornments are one way for Jitter objects to have additional information, and in some instances behavior, tacked onto an existing class. Adornments will be discussed in detail in a later chapter.

You also want to define your matrix calculation method, where most of the work of a Matrix Operator occurs, with the [jit_class_addmethod\(\)](#) function as a private, untyped method bound to the symbol `matrix_calc`.

```
jit_class_addmethod(_jit_scalebias_class,
    (method) jit_scalebias_matrix_calc,
    "matrix_calc", A_CANT, 0L);
```

21.2 The Jitter Class Constructor/Destructor

You don't need to add anything special to your Matrix Operator's constructor or destructor, aside from the standard initialization and cleanup any Jitter object would need to do. Any internal matrices for input and outputs are maintained, and only required, by the Max wrapper's asynchronous interface. The Jitter MOP contains no matrices for inputs and outputs, but rather expects that the matrix calculation method is called with all inputs and outputs synchronously. When used from languages like C, Java, and JavaScript, it is up to the programmer to maintain and provide any matrices which are being passed into the matrix calculation method.

21.3 The Matrix Calculation Method

The most important method for Matrix Operators, and the one in which the most work typically occurs is in the matrix calculation, or "matrix_calc" method, which should be defined as a private, untyped method with the `A_CANT` type signature, and bound to the symbol "matrix_calc". In this method your object receives a list of input matrices and output matrices to use in its calculation. You need to lock access to these matrices, inquire about important attributes, and ensure that any requirements with respect to type, plane count, or dimensionality for the inputs are met before actually processing the data, unlocking access to the matrices and returning. It should be defined as in the following example.

```
t_jit_err jit_scalebias_matrix_calc(t_jit_scalebias *x,
    void *inputs, void *outputs)
{
    t_jit_err err=JIT_ERR_NONE;
    long in_savelock,out_savelock;
    t_jit_matrix_info in_minfo,out_minfo;
    char *in_bp,*out_bp;
    long i,dimcount,planecount,dim[JIT_MATRIX_MAX_DIMCOUNT];
    void *in_matrix,*out_matrix;

    // get the zeroth index input and output from
    // the corresponding input and output lists
    in_matrix    = jit_object_method(inputs,_jit_sym_getindex,0);
    out_matrix   = jit_object_method(outputs,_jit_sym_getindex,0);

    // if the object and both input and output matrices
    // are valid, then process, else return an error
    if (x&&in_matrix&&out_matrix)
    {
        // lock input and output matrices
        in_savelock =
            (long) jit_object_method(in_matrix,_jit_sym_lock,1);
        out_savelock =
            (long) jit_object_method(out_matrix,_jit_sym_lock,1);

        // fill out matrix info structs for input and output
        jit_object_method(in_matrix,_jit_sym_getinfo,&in_minfo);
        jit_object_method(out_matrix,_jit_sym_getinfo,&out_minfo);

        // get matrix data pointers
        jit_object_method(in_matrix,_jit_sym_getdata,&in_bp);
        jit_object_method(out_matrix,_jit_sym_getdata,&out_bp);

        // if data pointers are invalid, set error, and cleanup
        if (!in_bp) { err=JIT_ERR_INVALID_INPUT; goto out; }
        if (!out_bp) { err=JIT_ERR_INVALID_OUTPUT; goto out; }

        // enforce compatible types
        if ((in_minfo.type!=_jit_sym_char) ||
            (in_minfo.type!=out_minfo.type))
    }
```

```

{
    err=JIT_ERR_MISMATCH_TYPE;
    goto out;
}

// enforce compatible planecount
if ((in_minfo.planecount!=4) ||
    (out_minfo.planecount!=4))
{
    err=JIT_ERR_MISMATCH_PLANE;
    goto out;
}

// get dimensions/planecount
dimcount = out_minfo.dimcount;
planecount = out_minfo.planecount;
for (i=0;i<dimcount;i++)
{
    // if input and output are not matched in
    // size, use the intersection of the two
    dim[i] = MIN(in_minfo.dim[i],out_minfo.dim[i]);
}

// calculate, using the parallel utility function to
// call the calculate_ndim function in multiple
// threads if there are multiple processors available
jit_parallel_ndim_simplecalc2(
    (method) jit_scalebias_calculate_ndim,
    x, dimcount, dim, planecount,
    &in_minfo, in_bp, &out_minfo, out_bp,
    0, 0);
} else {
    return JIT_ERR_INVALID_PTR;
}

out:
// restore matrix lock state to previous value
jit_object_method(out_matrix,_jit_sym_lock,out_savelock);
jit_object_method(in_matrix,_jit_sym_lock,in_savelock);
return err;
}

```

21.4 Processing N-Dimensional Matrices

Since Jitter supports the processing of N-dimensional matrices where N can be any number from 1 to 32, most Matrix Operators are designed with a recursive function that will process the data in some lower dimensional slice, most often 2 dimensional. The recursive function that does this is typically named myobject_calculate_ndim(), and is called by your matrix_calc method either directly or via one of the parallel processing utility functions, which are discussed in a future chapter.

It is out of the scope of this documentation to provide a detailed tutorial on fixed point or pointer arithmetic, both of which are used in this example. The code increments a pointer through the matrix data, scaling each planar element of each matrix cell by some factor and adding some bias amount. This is done with fixed point arithmetic (assuming an 8bit fractional component), since a conversion from integer to floating point data and back is an expensive operation. The jit.scalebias object also has two modes, one which sums the planes together, and one which processes each plane independently. You can improve performance by case handling on a per row, rather than per cell basis, and reduce your code somewhat by case handling on a per row, rather than per matrix basis. While a slight performance increase could be made by handling on a per matrix basis, per row is usually a decent point at which to make such an optimization trade off.

```
// recursive function to handle higher dimension matrices,
```

```

// by processing 2D sections at a time
void jit_scalebias_calculate_ndim(t_jit_scalebias **,
    long dimcount, long *dim, long planecount,
    t_jit_matrix_info *in_minfo, char *bip,
    t_jit_matrix_info *out_minfo, char *bop)
{
    long i,j,width,height;
    uchar *ip,*op;
    long ascale,rscale,gscale,bscale;
    long abias,rbias,gbias,bbias,sumbias;
    long tmp;

    if (dimcount<1) return; //safety

    switch(dimcount)
    {
    case 1:
        // if only 1D, interpret as 2D, falling through to 2D case
        dim[1]=1;
    case 2:
        // convert floating point scale factors to a fixed point int
        ascale = x->ascale*256.;
        rscale = x->rscale*256.;
        gscale = x->gscale*256.;
        bscale = x->bscale*256.;

        // convert floating point bias values to a fixed point int
        abias = x->abias*256.;
        rbias = x->rbias*256.;
        gbias = x->gbias*256.;
        bbias = x->bbias*256.;

        // for efficiency in sum mode (1), make a single bias value
        sumbias = (x->abias+x->rbias+x->gbias+x->bbias)*256.;

        width = dim[0];
        height = dim[1];

        // for each row
        for (i=0;i<height;i++)
        {
            // increment data pointers according to byte stride
            ip = bip + i*in_minfo->dimstride[1];
            op = bop + i*out_minfo->dimstride[1];

            switch (x->mode) {
            case 1:
                // sum together, clamping to the range 0-255
                // and set all output planes
                for (j=0;j<width;j++) {
                    tmp = (long)(*ip++)*ascale;
                    tmp += (long)(*ip++)*rscale;
                    tmp += (long)(*ip++)*gscale;
                    tmp += (long)(*ip++)*bscale;
                    tmp = (tmp>>8L) + sumbias;
                    tmp = (tmp>255)?255:((tmp<0)?0:tmp);
                    *op++ = tmp;
                    *op++ = tmp;
                    *op++ = tmp;
                    *op++ = tmp;
                }
                break;
            default:
                // apply to each plane individually
                // clamping to the range 0-255
                for (j=0;j<width;j++) {
                    tmp = (((long)(*ip++)*ascale)>>8L)+abias;

```

```

        *op++ = (tmp>255)?255:((tmp<0)?0:tmp);
        tmp = (((long)(*ip++)*rscale)>>8L)+rbias;
        *op++ = (tmp>255)?255:((tmp<0)?0:tmp);
        tmp = (((long)(*ip++)*gscale)>>8L)+gbias;
        *op++ = (tmp>255)?255:((tmp<0)?0:tmp);
        tmp = (((long)(*ip++)*bscale)>>8L)+bbias;
        *op++ = (tmp>255)?255:((tmp<0)?0:tmp);
    }
    break;
}
break;
default:
    // if processing higher dimension than 2D,
    // for each lower dimensioned slice, set
    // base pointer and recursively call this function
    // with decremented dimcount and new base pointers
    for (i=0;i<dim[dimcount-1];i++)
    {
        ip = bip + i*in_minfo->dimstride[dimcount-1];
        op = bop + i*out_minfo->dimstride[dimcount-1];
        jit_scalebias_calculate_ndim(x,dimcount1,
                                      dim,planeccount,in_minfo,ip,out_minfo,op);
    }
}
}

```

Rather than using multidimensional arrays, Jitter matrix data is packed in a single dimensional array, with defined byte strides for each dimension for greatest flexibility. This permits matrices to reference subregions of larger matrices, as well as support data that is not tightly packed. Therefore, rather than using multidimensional array syntax, this code uses pointer arithmetic to access each plane of each cell of the matrix, adding the corresponding byte strides to the base pointer for each dimension across which it is iterating. These byte strides are stored in the dimstride entry of the `t_jit_matrix_info` struct. Note that Jitter requires that planes within a cell, and cells across the first dimension (`dim[0]`) are tightly packed. The above code assumes that this is the case, using a simple pointer increment for each plane and cell, rather than looking up byte strides for `dim[0]`.

21.5 Defining the MOP Max Wrapper Class

In order to use the MOP class in a Max patcher you need to make a Max wrapper class. In addition to the standard methods used to wrap any Jitter class, MOPs need to add special methods and information to the Max class. One of the things that needs to happen is that the Max wrapper class needs to allocate and maintain instances of jit.matrix for each matrix input and output other than the leftmost input, to accommodate Max's asynchronous event model. In order to perform this maintenance, the Max wrapper class must have special methods and attributes for setting the type, plane count, dimensions, adaptability, and named references for the internal matrices. All of these messages are exclusive to the Max wrapper implementation, and are not used by the C, Java, or JavaScript usage of Matrix Operators. There are also common methods and attributes for the matrix output mode, and the jit_matrix and bang messages, all of which are specific to the MOP's Max wrapper. These special attributes and methods are added by the `max_jit_classex_mop_wrap()` function, which should be called inside your Max external's main function, after calling `max_jit_classex_setup()` and `jit_class_findbyname()`, and before calling `max_jit_classex_standard_wrap()`. Several default methods and attributes can be overridden using the various flags that can be combined for the flags argument to `max_jit_classex_mop_wrap()`. These flags, which for most simple MOPs won't be necessary, are listed below.

```
#define MAX_JIT_MOP_FLAGS_OWN_ALL 0xFFFFFFFFFFF  
#define MAX_JIT_MOP_FLAGS_OWN_JIT_MATRIX 0x00000001
```

```
#define MAX_JIT_MOP_FLAGS_OWN_BANG          0x00000002
#define MAX_JIT_MOP_FLAGS_OWN_OUTPUTMATRIX    0x00000004
#define MAX_JIT_MOP_FLAGS_OWN_NAME           0x00000008
#define MAX_JIT_MOP_FLAGS_OWN_TYPE            0x00000010
#define MAX_JIT_MOP_FLAGS_OWN_DIM             0x00000020
#define MAX_JIT_MOP_FLAGS_OWN_PLANECOUNT     0x00000040
#define MAX_JIT_MOP_FLAGS_OWN_CLEAR           0x00000080
#define MAX_JIT_MOP_FLAGS_OWN_NOTIFY          0x00000100
#define MAX_JIT_MOP_FLAGS_OWN_ADAPT           0x00000200
#define MAX_JIT_MOP_FLAGS_OWN_OUTPUTMODE      0x00000400
```

21.6 The Max Class Constructor/Destructor

Inside your Max class' constructor you need to allocate the matrices necessary for the MOP inputs and outputs, the corresponding matrix inlets and outlets, process matrix arguments and other MOP setup. The [max_jit_mop_setup_simple\(\)](#) function takes care of these functions and some of the other necessary tasks of wrapping your Jitter instance. As such, the use of this function simplifies your Jitter class wrapping even further for the simple case where no special behavior, incompatible with [max_jit_mop_setup_simple\(\)](#) is required. Here is the constructor for the Max class of the jit.scalebias object.

```
void *max_jit_scalebias_new(t_symbol *s, long argc, t_atom *argv)
{
    t_max_jit_scalebias *x;
    void *o;

    if (x = (t_max_jit_scalebias *) max_jit_obex_new(
        max_jit_scalebias_class,
        gensym("jit_scalebias")))
    {
        // instantiate Jitter object
        if (o= jit_object_new(gensym("jit_scalebias")))
        {
            // handle standard MOP max wrapper setup tasks
            max_jit_mop_setup_simple(x,o,argc,argv);

            // process attribute arguments
            max_jit_attr_args(x,argc,argv);
        }
        else
        {
            error("jit.scalebias: could not allocate object");
            freeobject(x);
        }
    }
    return (x);
}
```

Below is the listing of the [max_jit_mop_setup_simple\(\)](#) function, demonstrating the smaller pieces, it manages for you. If your object has special requirements, you can use whatever subset of the following function as necessary.

```
t_jit_err max_jit_mop_setup_simple(void *x, void *o, long argc, t_atom *argv)
{
    max_jit_obex_jitob_set(x,o);
    max_jit_obex_dumpout_set(x,outlet_new(x,NULL));
    max_jit_mop_setup(x);
    max_jit_mop_inputs(x);
    max_jit_mop_outputs(x);
    max_jit_mop_matrix_args(x,argc,argv);
```

```
    return JIT_ERR_NONE;
}
```

In your Max class' destructor, you need to free the resources allocated for your MOP. This is accomplished with the [max_jit_mop_free\(\)](#) function, which should be called before you free your internal Jitter instance, and your Max class' obex data. As an example, the jit.scalebias destructor is listed below.

```
void max_jit_scalebias_free(t_max_jit_scalebias *x)
{
    // free MOP max wrapper resources
    max_jit_mop_free(x);

    // lookup internal Jitter object instance and free
    jit_object_free(max_jit_obex_jitob_get(x));

    // free resources associated with obex entry
    max_jit_obex_free(x);
}
```

Chapter 22

Matrix Operator Details

The purpose of this chapter is to fill in the details of what a Matrix Operator is and how it works.

Matrix data in Jitter is typically considered raw data without respect to what the data represents. This permits simple fundamental operations to be applied to different sorts of data without needing to know any special information. For this reason most MOPs are general purpose. The jit.scalebias example from the preceding chapter could be considered video specific in its terminology, and type and plane count restrictions, but fundamentally it is just calculating a product and sum on each plane of an incoming matrix. In this chapter, we'll cover the details of how to configure MOP inputs and outputs, any attribute restrictions or linking for those inputs and outputs, what you must do in your matrix_calc method and how you expose your MOP to the Max environment, overriding default behavior if necessary.

22.1 Defining the MOP Jitter Class

As discussed in the Matrix Operator Quick Start, for MOPs you must create an instance of jit_mop with the [jit_object_new\(\)](#) function and add it to your Jitter class as an adornment with the [jit_class_addadornment\(\)](#) function. The jit_mop object holds information such as how many inputs and outputs the object has, what types, plane count, and dimension counts are supported, and how inputs should respond to incoming matrices. This information is only relevant to wrappers of the Jitter object which actually maintain additional matrices for inputs and outputs, as is the case with the MOP Max wrapper class. When used from C, Java, or JavaScript, it is the the programmer's responsibility to pass in matrices that conform to any restrictions imposed by the MOP. An example of instantiating and adding the jit_mop object is below.

```
// create a new instance of jit_mop with 1 input, and 1 output
mop = jit_object_new(_jit_sym_jit_mop,1,1);

// add jit_mop object as an adornment to the class
jit_class_addadornment(_jit_your_class,mop);
```

22.2 The jit_mop_io Object

Each instance of jit_mop contains some number of inputs and outputs, specified by the input and output arguments to the constructor. For each of these inputs and outputs there is an instance of jit_mop_io which records information specific to that input or output, such as type, plane count, and dimension restrictions. You can access the input or output objects by calling the getinput or getoutput methods with an integer index argument as below:

```
input = jit_object_method(mop,_jit_sym_getinput,1);
output = jit_object_method(mop,_jit_sym_getoutput,1);
```

Once you have obtained references to these inputs or outputs, you may query or set the jit_mop_io attributes. The attributes typically configured are: types, which is a list of symbols of permitted types, the first of which being the default; mindim and maxdim, which are the minimum and maximum permitted sizes for each dimension; mindimcount and maxdimcount, which are the minimum and maximum permitted number of dimensions permitted; minplanecount and maxplanecount, which are the minimum and maximum number of planes permitted; typelink, which is the flag that determines if the I/O should change its type to whatever the leftmost incoming matrix is; dimlink, which is the flag that determines if the I/O should change its dimensions to whatever the leftmost incoming matrix is; and planelink, which is the flag that determines if the I/O should change its plane count to whatever the leftmost incoming matrix is.

22.3 Restricting Input/Output Attributes

By default, all types, dimensions and plane count are permitted, and all linking is enabled. If you wish your MOP to have some specific restrictions, or difference in linking behaviors for any input or output in particular, you can set the corresponding attributes. For example, to set the plane count to always be four planes, you would set both the minplanecount and maxplanecount attributes to 4, as below:

```
output = jit_object_method(mop,_jit_sym_getoutput,1);
jit_attr_setlong(output,_jit_sym_minplanecount,4);
jit_attr_setlong(output,_jit_sym_maxplanecount,4);
```

The jit.scalebias example could have set the planecount using the minplanecount and maxplanecount attributes rather than calling the utility function [jit_mop_single_planecount\(\)](#), which internally sets these attributes. A similar thing could be done to restrict type and dimensions. As for linking, if you wish to develop an object where the right hand input does not adapt to the size of the leftmost input, as is the case with jit.convolve, you would turn off the dimlink attribute, as below:

```
input2 = jit_object_method(mop,_jit_sym_getinput,2);
jit_attr_setlong(input2,_jit_sym_dimlink,0);
```

Similar could be done to remove type and planecount linking, and the utility functions [jit_mop_input_nolink\(\)](#) and [jit_mop_output_nolink\(\)](#) set all of these link attributes to false (zero).

22.4 The ioproc Function

For right hand matrix inputs, incoming data is typically copied by the MOP Max wrapper class. When an incoming matrix is received by the MOP Max wrapper class, a function called the ioproc is called, and the default ioproc copies the data, using the current input attributes (which might be linked to the lefthand input). The default ioproc can be overridden by calling the ioproc method followed by a function with the signature as listed below in the [jit_mop_ioproc_copy_adapt\(\)](#) function. The [jit_mop_ioproc_copy_adapt\(\)](#) function will always adapt to that inlet's incoming matrix attributes, as long as they don't conflict with any restrictions. The SDK project for jit.concat demonstrates the use of the [jit_mop_ioproc_copy_adapt\(\)](#) function.

```
t_jit_err jit_mop_ioproc_copy_adapt(void *mop, void *mop_io, void *matrix)
{
    void *m; // destination matrix
    t_jit_matrix_info info;

    // look up destination matrix from mop_io
    if (matrix&&(m= jit_object_method(mop_io,_jit_sym_getmatrix)))
    {
        // retrieve incoming matrix info
        jit_object_method(matrix,_jit_sym_getinfo,&info);

        //restrict matrix info based on mop_io attribtues
        jit_object_method(mop_io,_jit_sym_restrict_type,&info);
        jit_object_method(mop_io,_jit_sym_restrict_dim,&info);
        jit_object_method(mop_io,_jit_sym_restrict_planecount,&info);

        // set destination matrix info
        jit_object_method(m,_jit_sym_setinfo,&info);

        // copy the data with the frommatrix method
        jit_object_method(m,_jit_sym_frommatrix,matrix,NULL);
    }
}
```

```

    return JIT_ERR_NONE;
}

```

22.5 Variable Inputs/Outputs

You can specify variable input/output MOPs with a negative argument for input and/or outputs when constructing your jit_mop object. When the using variable inputs and/or outputs, there is not a jit_mop_io for each input and/or output within your class definition, and therefore the template type, dim, planeCount, and linking attributes are not settable. If anything but the default behavior is required, you must accomplish it in another way — for example, either by overriding the jit_matrix method of the MOP Max wrapper class, or defining an mproc method to be called from within the standard jit_matrix method of the MOP Max wrapper class. The jit.pack, jit.unpack, jit.scissors, and jit.glue objects are a few SDK examples of MOPs with variable inputs and outputs. More information on overriding the jit_matrix, mproc, and other default methods of the MOP Max wrapper class is covered later in this chapter.

22.6 Adding jit_mop as a Class Adornment

Once you have configured all of the inputs and outputs of your jit_mop object, you must add your jit_mop object to your Jitter class with the [jit_class_addadornment\(\)](#) function. Adornments can be queried from the Jitter class at any time by calling [jit_class_adornment_get\(\)](#) with the Jitter class pointer and the class name of the adornment object, as demonstrated below.

```

// add jit_mop object as an adornment to the class
jit_class_addadornment(_jit_your_class,mop);

// look up jit_mop adornment
mop = jit_class_adornment_get(_jit_your_class,_jit_sym_jit_mop);

```

22.7 The Matrix Calculation Method

The entry point of the MOP Jitter class is the matrix_calc method, which is passed a list of matrices for the input, and a list of matrices for the output. It is not the responsibility of the matrix_calc method to perform any copying and adaptation behavior, but rather simply ensure that the matrices are valid, compatible, and if so, process. Certain objects may modify the dim, type, or planeCount of the output matrices — e.g. the SDK project, jit.thin. However, it is the calling party's responsibility to perform any copying and conformance to MOP I/O restrictions as defined by the jit_mop_io objects—i.e. either the Max wrapper class, or the C, Java, or Javascript code which calls the matrix_calc method.

22.8 Accessing the Input and Output Lists

The input and output lists passed as arguments to your matrix_calc method are Jitter objects, and pointers to the individual inputs and outputs are acquired by calling the getindex method with an integer argument specifying the zero based list index. The return values should be tested to make sure they are not null. For example:

```

// get the zeroth index input and output from
// the corresponding input and output lists
in_matrix      = jit_object_method(inputs,_jit_sym_getindex,0);

```

```

out_matrix = jit_object_method(outputs,_jit_sym_getindex,0);

// if the object and both input and output matrices
// are valid, then process, else return an error
if (x&&in_matrix&&out_matrix)
{
    // ... process data ...

} else {
    return JIT_ERR_INVALID_PTR;
}

```

Technically, you can also pass in an instance of jit_matrix in place of a list for the input or output arguments, since jit_matrix has a getindex method which returns the jit_matrix instance. This is an example of dynamic binding at work. Another example of dynamic binding inside the matrix_calc method is that the list elements might be instances of jit_mop_io, rather than instances of jit_matrix. However, since Jitter uses dynamic binding and the jit_mop_io object is a "decorator" class for jit_matrix, all corresponding methods are passed on to the jit_matrix referenced by the jit_mop_io. In fact, any Jitter objects which respond to the standard interface for jit_matrix could be passed as inputs or outputs. If this seems confusing, you need not think about the underlying implementation further, but instead can assume that what is being passed in is simply an instance of jit_matrix. After all it should behave like one, even if it is not.

22.9 Locking and Unlocking Matrices

Prior to working with a matrix, it is necessary to "lock" it so that the data and attributes will not be changed across the duration of the operation. This is accomplished by calling the jit_matrix instance's lock method with an integer argument of 1 (true) to lock the matrix. You should store the current lock state to restore when you're done processing. The lock operation should be the first thing to do after ensuring that the matrix objects are not NULL. For example

```

// lock input and output matrices
in_savelock = (long) jit_object_method(in_matrix,_jit_sym_lock,1);
out_savelock = (long) jit_object_method(out_matrix,_jit_sym_lock,1);

// ... process data ...

out:
// restore matrix lock state to previous value
jit_object_method(out_matrix,_jit_sym_lock,out_savelock);
jit_object_method(in_matrix,_jit_sym_lock,in_savelock);

```

22.10 Retrieving Matrix Information

Once you have locked the matrices, you are ready to find out some information about them. This is accomplished by calling the getinfo method with a pointer to an instance of the [t_jit_matrix_info](#) struct. The [t_jit_matrix_info](#) struct contains several common attributes of the matrix and data organization of the matrix data, and is a useful way to obtain this information in one call, rather than querying each attribute individually. This information is typically tested to verify compatibility with any assumptions the matrix_calc method needs to make (since this method might be called from C, Java, or Javascript, you cannot assume that the MOP Max wrapper will have enforced these assumptions). It is also used to perform the appropriate pointer arithmetic based on type, plane count, dimensions, and the byte stride of those dimensions, since higher dimensions may not be tightly packed. The [t_jit_matrix_info](#) struct is listed below:

```
typedef struct _jit_matrix_info
```

```
{
    long      size;           // in bytes (0xFFFFFFFF=UNKNOWN)
    t_symbol *type;          // primitive type
    long      flags;          // matrix flags: my data?, handle?
    long      dimcount;       // # of dimensions
    long      dim[JIT_MATRIX_MAX_DIMCOUNT]; // dimension sizes
    long      dimstride[JIT_MATRIX_MAX_DIMCOUNT]; // in bytes
    long      planecount;     // # of planes
} t_jit_matrix_info;
```

And here is an example of calling the getinfo method to fill out the `t_jit_matrix_info` struct:

```
// fill out matrix info structs for input and output
jit_object_method(in_matrix,_jit_sym_getinfo,&in_minfo);
jit_object_method(out_matrix,_jit_sym_getinfo,&out_minfo);
```

22.11 Retrieving the Data Pointer

The `t_jit_matrix_info` struct is the meta data, but the actual matrix data can be accessed by acquiring the data pointer. You accomplish this by calling the matrix's getdata method, passing in a pointer to a pointer. This pointer can be any type, but it is typically a char (or byte) pointer since you may need to perform bytewise pointer arithmetic depending on the type and dimstride of your matrix. It is essential to verify that this pointer is valid before attempting to operate on the data, as demonstrated below.

```
// get matrix data pointers
jit_object_method(in_matrix,_jit_sym_getdata,&in_bp);
jit_object_method(out_matrix,_jit_sym_getdata,&out_bp);

// if data pointers are invalid, set error, and cleanup
if (!in_bp) { err=JIT_ERR_INVALID_INPUT; goto out; }
if (!out_bp) { err=JIT_ERR_INVALID_OUTPUT; goto out; }
```

22.12 Processing the Data

While it is possible to incorporate the data processing code inside the `matrix_calc` method, it is typical to rely on other routines to accomplish the N dimensional processing through recursion, potentially dispatching to multiple processors. The N-dimensional recursive processing function (typically named `myobject_calculate_ndim`) is discussed in the next section. You should pass in to the `calculate_ndim` function your object pointer, the overall dimension count, dimension sizes, planecount to consider in your calculation, together with the necessary matrix info structs and data pointers for each input and output. You can call this method directly as is the case in the following code:

```
// call calculate_ndim function directly in current thread
jit_scalebias_calculate_ndim(x, dimcount, dim, planecount,
    &in_minfo, in_bp, &out_minfo, out_bp);
```

Or you can call this method with the parallel processing utility functions provided with Jitter 1.5 to automatically dispatch the processing of large matrices across multiple processors when available. This figure illustrates the dispatching and calculating of the parallel processing utility:

The parallel processing is accomplished by breaking up the matrix into smaller matrices that each reference subregions of the original inputs and outputs. No new objects are created, but rather just additional `t_jit_matrix_info` structs and offset data pointers. Jitter 1.5 maintains a pool of worker threads for this purpose, so there is no thread creation overhead, but rather only some small thread synchronization overhead. Jitter

1.5 only dispatches across multiple threads when the data count is large enough to justify this thread synchronization overhead.

An important thing worth noting is that if your object performs some kind of spatial operation (e.g. convolution, rotation, scaling, etc.), you will either need to account for the matrix segmentation used by the parallel utilities or avoid using parallel processing and call directly in the current thread. Since the jit.scalebias example only processes one pixel at a time (i.e. a pointwise operation), it is inherently parallelizable, so it takes advantage of multiple processors as below:

```
// calculate, using the parallel utility function to
// call the calculate_ndim function in multiple
// threads if there are multiple processors available
jit_parallel_ndim_simplecalc2(
    (method) jit_scalebias_calculate_ndim,
    x, dimcount, dim, planecount,
    &in_minfo, in_bp, &out_minfo, out_bp,
    0, 0 );
```

Important Note: If you aren't sure if your object is a pointwise operator, or don't fully understand how to make your algorithm parallelizable, you shouldn't use the parallel utility functions in your object. You should simply call the function directly.

22.13 Processing N-Dimensional Matrices

In the Matrix Operator Quick Start chapter, we discussed how to define a recursive function to process N-dimensional data in 2D slices, using the jit.scalebias object as an example. This example was restricted to processing four plane char data, but many Jitter objects work with any type of data and any plane count. In order to support all types and plane counts, there needs to be some case handling to know how to step through the data, and what type data to interpret as so that you can perform the appropriate operations. There are a number of ways to approach this logic, and decisions to make with respect to optimization. All this case handling can be a bit cumbersome, so when initially developing objects, it probably makes sense for you to focus on a single type and plane count, and only after you've adequately defined your operation, attempt to make your code robust to process any type of data and consider optimization of certain cases. The use of C macros, or C++ templates might be useful things to explore for better code re-use. As for code optimization, typically a decent atomic element to try and optimize is the "innermost" loop, avoiding branch conditions where possible.

This function is at the heart of the logic you will add in your own custom object. Since there is no "right way" to process this data, we won't cover any more code listings for the recursive N-dimensional processing function. However, the SDK projects that are good examples include: jit.clip, which performs a planar independent, pointwise operation (limiting numbers to some specified range); jit.rgb2luma, which performs a planar dependent, pointwise operation (converting RGB color to luminance); and jit.transpose, which performs a planar independent, spatial operation (rows become columns). For more ideas about N-dimensional matrix processing, we would recommend reading one of the several books available on 2D signal processing and/or image processing. Most of these concepts are easily generalized to higher dimensions.

22.14 Defining the MOP Max Wrapper Class

MOP Max wrapper classes typically have a large amount of default behavior, as setup through the max_jit_classex_mop_wrap function, based on the jit_mop Jitter class adornment, and user specified flags. You can either override all of the default behavior or just specific features. If you wish to override all of the

default behavior, you can use the flag [MAX_JIT_MOP_FLAGS_OWN_ALL](#), when calling the [max_jit_classex_mop_wrap\(\)](#) function. If you need to make use of the jit_mop adornment(), the jit_mop can be looked up by calling the [jit_class_adornment_get\(\)](#) method on the Jitter class. The jit_mop_io inputs and outputs can be queried and their attributes inspected, similar to how they were set in the MOP Jitter class definition, described earlier in this chapter. Here is an example of how to look up the jit_mop adornment of the jit.scalebias object:

```
// look up jitter class by name
jclass = jit_class_findbyname(gensym("jit_scalebias"));
// look up jit_mop adornment
mop = jit_class_adornment_get(jclass,_jit_sym_jit_mop);
```

22.15 Overriding the jit_matrix Method

By default, a jit_matrix method is added which automatically manages matrix copying and calculation based on the incoming data. Most typical MOPs simply use the default jit_matrix method. However there are instances where it is necessary to override the default MOP method to get special behavior, such as recording which matrix input data is being input to as is the case for the jit.op SDK example, or to do something other than standard copying and adaptation as is the case for the jit.pack or jit.str.op SDK examples, or to prevent any jit_matrix method at all, as is the case for the jit.noise SDK example. To prevent the default jit_matrix method from being defined, you can use the flag [MAX_JIT_MOP_FLAGS_OWN_JIT_MATRIX](#), when calling the [max_jit_classex_mop_wrap\(\)](#) function. To define your own jit_matrix method, you can add an [A_GIMME](#) method bound to the symbol jit_matrix, in your main function. Here's an example from jit.op:

```
// add custom jit_matrix method in main()
addmess((method)max_jit_op_jit_matrix, "jit_matrix", A_GIMME, 0);

void max_jit_op_jit_matrix(t_max_jit_op **x, t_symbol *s, short argc,
                           t_atom *argv)
{
    if (max_jit_obex_inletnumber_get(x))
    {
        // if matrix is received in right input,
        // record to override float or int input
        x->last = OP_LAST_MATRIX;
    }

    // now pass on to the default jit_matrix method
    max_jit_mop_jit_matrix(x,s,argc,argv);
}
```

The jit.pack and jit.str.op examples are a bit more involved and also better illustrate the kinds of tasks the default jit_matrix method performs.

22.16 Overriding the bang and outputmatrix Methods

A MOP Max wrapper class typically has a bang and outputmatrix method. These two methods are typically equivalent, and by default, both send out the most recently calculated matrix output. Certain objects that don't have a matrix output, like the jit.3m SDK example, typically override these messages with their own bang and sometimes outputmatrix method. These methods can be overridden by using the [MAX_JIT_MOP_FLAGS_OWN_BANG](#) and [MAX_JIT_MOP_FLAGS_OWN_OUTPUTMATRIX](#) flags when calling the [max_jit_classex_mop_wrap\(\)](#) function. These flags are typically both passed in together.

22.17 Overriding the name, type, dim, and planecount Attributes

For each input and output, other than the leftmost input, there is, by default, an attribute added to query and set that input or output's matrix attributes, including name, type, dim, and planecount. While overriding the default attribute behavior is conceivably necessary to perform very specialized behavior, it is not used by any of the SDK examples. To prevent the addition of the default attributes for name, type, dim, and planecount, you can use the [MAX_JIT_MOP_FLAGS_OWN_NAME](#), [MAX_JIT_MOP_FLAGS_OWN_TYPE](#), [MAX_JIT_MOP_FLAGS_OWN_DIM](#), and [MAX_JIT_MOP_FLAGS_OWN_PLANECOUNT](#) flags when calling the [max_jit_classex_mop_wrap\(\)](#) function. To define your own attributes, you would follow the same means of defining any attributes for a Max wrapper class with the appropriate attribute name you wish to override.

22.18 Overriding the clear and notify Methods

By default, a clear and a notify method are added. The default clear method clears each of the input and output matrices. The default notify method, [max_jit_mop_notify\(\)](#), is called whenever any of the matrices maintained by the MOP are changed. If it is necessary to respond to additional notifications, it is important to call the [max_jit_mop_notify](#) function so that the MOP can perform any necessary maintenance with respect to input and output matrices, as demonstrated by the jit.notify SDK example. These methods can be overridden using the [MAX_JIT_MOP_FLAGS_OWN_CLEAR](#) and [MAX_JIT_MOP_FLAGS_OWN_NOTIFY](#) flags, respectively, when calling the [max_jit_classex_mop_wrap\(\)](#) function. Object registration and notification is covered in detail in a future chapter, but the jit.notify notify method is provided as an example.

```
// s is the servername, msg is the message, ob is the server object pointer,
// and data is extra data the server might provide for a given message
void max_jit_notify_notify(
    t_max_jit_notify *x, t_symbol *s, t_symbol *msg, void *ob, void *data)
{
    if (msg==gensym("splat")) {
        post("notify: server=%s message=%s", s->s_name, msg->s_name);
        if (!data) {
            error("splat message NULL pointer");
            return;
        }
        // here's where we output using the rightmost outlet
        // we just happen to know that "data" points to a t_atom[3]
        max_jit_obex_dumpout(x,msg,3,(t_atom *)data);
    } else {
        // pass on to the default Max MOP notification method
        max_jit_mop_notify(x,s,msg);
    }
}
```

22.19 Overriding the adapt and outputmode Attributes

By default, adapt and outputmode attributes are added to the MOP Max Wrapper. These attributes determine whether or not to adapt to incoming matrix attributes, and whether or not the output should calculate a new output matrix, output the last calculated matrix (freeze), pass on the input matrix (bypass). To prevent the addition of the default attributes for adapt and outputmode, you can use the [MAX_JIT_MOP_FLAGS_OWN_ADAPT](#), and [MAX_JIT_MOP_FLAGS_OWN_OUTPUTMODE](#) flags when calling the [max_jit_classex_mop_wrap\(\)](#) function. To define your own attributes, you would follow the same means of defining any attributes for a Max wrapper class with the appropriate attribute name you wish to override.

22.20 Defining an mproc Method

For many types of operations, it's not required to fully override the default jit_matrix method and any adaptation. If your object simply needs to override the way in which the Jitter class' matrix_calc method and outlet functions are called, you can do so by defining an mproc method, which will be called instead of the default behavior. The jit.3m SDK project is an example where after it calls the Jitter class' matrix_calc method, it queries the Jitter class' attributes and outputs max messages rather than the default jit_matrix message output.

```
void max_jit_3m_mproc(t_max_jit_3m **x, void *mop)
{
    t_jit_err err;

    // call internal Jitter object's matrix_calc method
    if (err=(t_jit_err) jit_object_method(
        max_jit_obex_jitob_get(x),
        _jit_sym_matrix_calc,
        jit_object_method(mop,_jit_sym_getinputlist),
        jit_object_method(mop,_jit_sym_getoutputlist)))
    {
        // report error if present
        jit_error_code(x,err);
    } else {
        // query Jitter class and makes outlet calls
        max_jit_3m_bang(x);
    }
}
```

22.21 The Max Class Constructor/Destructor

As we discussed in the Matrix Operator Quick Start, inside your Max class' constructor you need to allocate the matrices necessary for the MOP inputs and outputs, the corresponding matrix inlets and outlets, process matrix arguments and other MOP setup. And in your destructor, you need to free oup MOP resources. Typically you would accomplish this all with the standard [max_jit_mop_setup_simple\(\)](#) and [max_jit_mop_free\(\)](#) functions, however there are some instances where you may need to introduce custom behavior.

22.21.1 Variable Inputs/Outputs

The [max_jit_mop_setup_simple\(\)](#) function calls [max_jit_mop_inputs\(\)](#) and [max_jit_mop_outputs\(\)](#) to define any necessary proxy inlets, outlets, and internal matrices. The listing for these functions are provided below to illustrate the default behavior, and a few SDK projects we recommend investigating further are jit.scissors, jit.glue, jit.pack, and jit.unpack.

```
t_jit_err max_jit_mop_inputs(void **x)
{
    void *mop,*p,*m;
    long i,incount;
    t_jit_matrix_info info;
    t_symbol *name;

    // look up object's MOP adornment
    if (*x&&(mop=max_jit_obex_adornment_get(x,_jit_sym_jit_mop)))
    {
        incount = jit_attr_getlong(mop,_jit_sym_inputcount);

        // add proxy inlet and internal matrix for
        // all inputs except leftmost inlet
```

```

        for (i=2;i<=incount;i++) {
            max_jit_obex_proxy_new(x,(incount+1)-i); // right to left
            if (p= jit_object_method(mop,_jit_sym_getinput,i)) {
                jit_matrix_info_default(&info);
                max_jit_mop_restrict_info(x,p,&info);
                name = jit_symbol_unique();
                m = jit_object_new(_jit_sym_jit_matrix,&info);
                m = jit_object_register(m,name);
                jit_attr_setsym(p,_jit_sym_matrixname,name);
                jit_object_method(p,_jit_sym_matrix,m);
                jit_object_attach(name, x);
            }
        }
        return JIT_ERR_NONE;
    }
    return JIT_ERR_INVALID_PTR;
}

t_jit_err max_jit_mop_outputs(void *x)
{
    void *mop,*p,*m;
    long i,outcount;
    t_jit_matrix_info info;
    t_symbol *name;

    if (x&&(mop=max_jit_obex_adornment_get(x,_jit_sym_jit_mop)))
    {
        outcount = jit_attr_getlong(mop,_jit_sym_outputcount);

        // add outlet and internal matrix for all outputs
        for (i=1;i<=outcount;i++) {
            max_jit_mop_matrixout_new(x,(outcount)-i);// right to left
            if (p= jit_object_method(mop,_jit_sym_getoutput,i)) {
                jit_matrix_info_default(&info);
                max_jit_mop_restrict_info(x,p,&info);
                name = jit_symbol_unique();
                m = jit_object_new(_jit_sym_jit_matrix,&info);
                m = jit_object_register(m,name);
                jit_attr_setsym(p,_jit_sym_matrixname,name);
                jit_object_method(p,_jit_sym_matrix,m);
                jit_object_attach(name, x);
            }
        }
        return JIT_ERR_NONE;
    }
    return JIT_ERR_INVALID_PTR;
}

```

22.21.2 Matrix Arguments

The [max_jit_mop_setup_simple\(\)](#) function calls [max_jit_mop_matrix_args\(\)](#) to read any matrix arguments, and if present send them to any linked inputs/outputs and disable the adapt attribute. The listing is provided below to illustrate the default behavior.

```

t_jit_err max_jit_mop_matrix_args(void *x, long argc, t_atom *argv)
{
    void *mop,*p,*m;
    long incount,outcount,attrstart,i,j;
    t_jit_matrix_info info,info2;

    if (! (mop=max_jit_obex_adornment_get(x,_jit_sym_jit_mop)))
        return JIT_ERR_GENERIC;

```

```

incount = jit_attr_getlong(mop,_jit_sym_inputcount);
outcount = jit_attr_getlong(mop,_jit_sym_outputcount);

jit_matrix_info_default(&info);

attrstart = max_jit_attr_args_offset(argc,argv);
if (attrstart&&argv) {
    jit_atom_arg_getlong(&info.planecount, 0, attrstart, argv);
    jit_atom_arg_getsym(&info.type, 1, attrstart, argv);
    i=2; j=0;
    while (i<attrstart) { //dimensions
        jit_atom_arg_getlong(&(info.dim[j]), i, attrstart, argv);
        i++; j++;
    }
    if (j) info.dimcount=j;

    jit_attr_setlong(mop,_jit_sym_adapt,0); //adapt off
}

jit_attr_setlong(mop,_jit_sym_outputmode,1);

for (i=2;i<=incount;i++) {
    if ((p= jit_object_method(mop,_jit_sym_getinput,i)) &&
        (m= jit_object_method(p,_jit_sym_getmatrix)))
    {
        jit_object_method(m,_jit_sym_getinfo,&info2);
        if (jit_attr_getlong(p,_jit_sym_typealink)) {
            info2.type = info.type;
        }
        if (jit_attr_getlong(p,_jit_sym_planelink)) {
            info2.planecount = info.planecount;
        }
        if (jit_attr_getlong(p,_jit_sym_dimlink)) {
            info2.dimcount = info.dimcount;
            for (j=0;j<info2.dimcount;j++) {
                info2.dim[j] = info.dim[j];
            }
        }
        max_jit_mop_restrict_info(x,p,&info2);
        jit_object_method(m,_jit_sym_setinfo,&info2);
    }
}

for (i=1;i<=outcount;i++) {
    if ((p= jit_object_method(mop,_jit_sym_getoutput,i)) &&
        (m= jit_object_method(p,_jit_sym_getmatrix)))
    {
        jit_object_method(m,_jit_sym_getinfo,&info2);
        if (jit_attr_getlong(p,_jit_sym_typealink)) {
            info2.type = info.type;
        }
        if (jit_attr_getlong(p,_jit_sym_planelink)) {
            info2.planecount = info.planecount;
        }
        if (jit_attr_getlong(p,_jit_sym_dimlink)) {
            info2.dimcount = info.dimcount;
            for (j=0;j<info2.dimcount;j++) {
                info2.dim[j] = info.dim[j];
            }
        }
        max_jit_mop_restrict_info(x,p,&info2);
        jit_object_method(m,_jit_sym_setinfo,&info2);
    }
}

return JIT_ERR_NONE;
}

```

Chapter 23

OB3D QuickStart

The purpose of this chapter is to give a quick and high level overview of how to develop a simple Jitter OpenGL object which draws geometry within a named rendering context - we refer to such an object as an OB3D.

For this task, we will use the jit.gl.simple SDK example. More details such as how to make an OpenGL object which deals with resources such as display lists and textures, wishes to support matrix input/output, or needs greater access to OpenGL state will appear in the following chapter. This chapter assumes familiarity with Jitter's OpenGL object suite used from the Max patcher, as discussed in the Jitter Tutorial, and the preceding chapters on the Jitter object model and Max wrapper classes.

23.1 Defining the OB3D Jitter Class

Jitter OB3Ds typically are defined to have all or most of the common OB3D attributes and methods discussed in the Group-OB3D section of the Jitter HTML object reference. These include attributes and methods to set the rendering destination name, object name, color, lighting, texturing, modelview transform, depth buffering, polygon mode, and several other common tasks. These common attributes and methods are added by the call to the [jit_ob3d_setup\(\)](#) function in your Jitter class definition, after calling [jit_class_new](#), but typically prior to defining other methods and attributes. For an OB3D, Jitter needs to store additional information in your object. This information is stored in an opaque pointer in your object struct, typically named ob3d. The byte offset to your OB3D data pointer is passed into [jit_ob3d_setup\(\)](#). You can override any default attributes and methods added by [jit_ob3d_setup\(\)](#) with the following flags:

```
#define JIT_OB3D_NO_ROTATION_SCALE      1 << 0
#define JIT_OB3D_NO_POLY_VARS           1 << 1
#define JIT_OB3D_NO_BLEND              1 << 2
#define JIT_OB3D_NO_TEXTURE             1 << 3
#define JIT_OB3D_NO_MATRIXOUTPUT       1 << 4
#define JIT_OB3D_AUTO_ONLY             1 << 5
#define JIT_OB3D_DOES_UI               1 << 6
#define JIT_OB3D_NO_DEPTH              1 << 7
#define JIT_OB3D_NO_ANTIALIAS          1 << 8
#define JIT_OB3D_NO_FOG                1 << 9
#define JIT_OB3D_NO_LIGHTING_MATERIAL  1 << 10
#define JIT_OB3D_HAS_LIGHTS            1 << 11
#define JIT_OB3D_HAS_CAMERA             1 << 12
#define JIT_OB3D_IS_RENDERER           1 << 13
#define JIT_OB3D_NO_COLOR              1 << 14
```

Aside from the attributes and methods added to your class by [jit_ob3d_setup\(\)](#), you need to define a private, untyped method bound to the symbol ob3d_draw. This method is where your object does all its drawing. It is called by the standard OB3D draw and drawraw methods. The OB3D draw method sets up all of the OpenGL state associated with the common OB3D attributes before calling your private ob3d_draw method. The drawraw method simply sets the context before calling your private ob3d_draw method. Because OB3Ds support being named for use within jit.gl.sketch*'s drawobject command, you must also add a private, untyped "register" method associated with the [jit_object_register\(\)](#) function. Let's examine the *jit.gl.simple SDK project as an example:

```
t_jit_err jit_gl_simple_init(void)
{
    long ob3d_flags = JIT_OB3D_NO_MATRIXOUTPUT; // no matrix output
    void *ob3d;

    _jit_gl_simple_class = jit_class_new("jit_gl_simple",
        (method) jit_gl_simple_new, (method) jit_gl_simple_free,
        sizeof(t_jit_gl_simple), 0L);

    // set up object extension for 3d object, customized with flags
```

```

ob3d = jit_ob3d_setup(_jit_gl_simple_class,
                      calcoffset(t_jit_gl_simple, ob3d),
                      ob3d_flags);

// define the OB3D draw method. called in automatic mode by
// jit.gl.render or otherwise through ob3d when banged. this
// method is A_CANT because our draw setup needs to happen
// in the ob3d beforehand to initialize OpenGL state
jit_class_addmethod(_jit_gl_simple_class,
                    (method) jit_gl_simple_draw, "ob3d_draw", A_CANT, 0L);

// define the dest_closing and dest_changed methods.
// these methods are called by jit.gl.render when the
// destination context closes or changes: for example, when
// the user moves the window from one monitor to another. Any
// resources your object keeps in the OpenGL machine
// (e.g. textures, display lists, vertex shaders, etc.)
// will need to be freed when closing, and rebuilt when it has
// changed. In this object, these functions do nothing, and
// could be omitted.
jit_class_addmethod(_jit_gl_simple_class,
                    (method) jit_gl_simple_dest_closing, "dest_closing", A_CANT, 0L);
jit_class_addmethod(_jit_gl_simple_class,
                    (method) jit_gl_simple_dest_changed, "dest_changed", A_CANT, 0L);

// must register for ob3d use
jit_class_addmethod(_jit_gl_simple_class,
                    (method) jit_object_register, "register", A_CANT, 0L);

jit_class_register(_jit_gl_simple_class);

return JIT_ERR_NONE;
}

```

23.2 The Jitter Class Constructor/Destructor

In your OB3D Jitter Class constructor, you need to pass in your rendering destination name as the first argument. You should call the [jit_ob3d_new\(\)](#) function with your destination name argument to initialize the OB3D data pointer, associating it with your rendering destination. In your destructor, you need to free your OB3D data pointer with [jit_ob3d_free\(\)](#). The jit.gl.simple constructor and destructors are below as an example.

```

t_jit_gl_simple *jit_gl_simple_new(t_symbol *dest_name)
{
    t_jit_gl_simple *x;

    // make jit object
    if (x = (t_jit_gl_simple *)jit_object_alloc(_jit_gl_simple_class))
    {
        // create and attach ob3d
        jit_ob3d_new(x, dest_name);
    }
    else
    {
        x = NULL;
    }
    return x;
}

void jit_gl_simple_free(t_jit_gl_simple *x)
{
    // free ob3d data

```

```

    jit_ob3d_free(x);
}

```

23.3 The OB3D draw Method

Your OB3D draw method, bound to the ob3d_draw symbol, is where all of your drawing code takes place. It is called automatically when your associated jit.gl.render object receives a bang, if your automatic and enabled attributes are turned on, as they are by default. It is also called if your Max wrapper object receives a bang, or the draw or drawraw messages. With the exception of the drawraw message, all of the standard OB3D object state is setup prior to calling your ob3d_draw method, so you needn't setup things like the modelview transform, color, lighting properties, texture information, if your object doesn't have special needs. The following example from jit.gl.simple, just draws a simple quadrilateral.

```

t_jit_err jit_gl_simple_draw(t_jit_gl_simple *x)
{
    t_jit_err result = JIT_ERR_NONE;

    // draw our OpenGL geometry.
    glBegin(GL_QUADS);
    glVertex3f(-1,-1,0);
    glVertex3f(-1,1,0);
    glVertex3f(1,1,0);
    glVertex3f(1,-1,0);
    glEnd();

    return result;
}

```

Since this example is meant only to show a minimal object which draws geometry with standard OpenGL calls, there is no texture information or vertex normals specified. However, all standard OpenGL calls should work within the ob3d_draw method. This example also doesn't show matrix output, as accomplished by [jit_ob3d_draw_chunk\(\)](#), which will be discussed in the following chapter on OB3D details.

23.4 Defining the OB3D Max Wrapper Class

For OB3Ds, the Max wrapper class has less extra work than for MOPs. In your Max wrapper class definition, you need only add a call to the [max_ob3d_setup\(\)](#) function to add your standard drawing methods, and the [max_jit_ob3d_assist\(\)](#) function as your assist method, unless you wish to define your own custom assist method. Everything else is similar to the standard technique of wrapping a Jitter Class demonstrated in the Max Wrapper Class chapter.

```

void main(void)
{
    void *classex, *jitclass;

    // initialize Jitter class
    jit_gl_simple_init();

    // create Max class
    setup((t_messlist **) &max_jit_gl_simple_class,
          (method) max_jit_gl_simple_new, (method) max_jit_gl_simple_free,
          (short) sizeof(t_max_jit_gl_simple), 0L, A_GIMME, 0);

    // specify a byte offset to keep additional information about our object
    classex = max_jit_classex_setup(calcoffset(t_max_jit_gl_simple, obex));
}

```

```

// look up Jitter class in the class registry
jitclass = jit_class_findbyname(gensym("jit_gl_simple"));

// wrap Jitter class with the standard methods for Jitter objects
max_jit_classex_standard_wrap(classex, jitclass, 0);

// use standard ob3d assist method
addmess((method)max_jit_ob3d_assist, "assist", A_CANT, 0);

// add methods for 3d drawing
max_ob3d_setup();
}

```

23.5 The Max Class Constructor/Destructor

Your Max class' constructor should be similar to the standard Max wrapper constructor, but the differences worth noting are that you should pass your first normal argument, which is the rendering destination, on to your Jitter OB3D constructor, and create a second outlet for matrix output, attached to your object's OB3D data. For your destructor, there is nothing additional you need to do for OB3D. The jit.gl.simple Max class' constructor and destructor are provided as examples.

```

void *max_jit_gl_simple_new(t_symbol *s, long argc, t_atom *argv)
{
    t_max_jit_gl_simple *x;
    void *jit_ob;
    long attrstart;
    t_symbol *dest_name_sym = _jit_sym_nothing;

    if (x = (t_max_jit_gl_simple *) max_jit_obex_new(
        max_jit_gl_simple_class, gensym("jit_gl_simple")))
    {
        // get first normal arg, the destination name
        attrstart = max_jit_attr_args_offset(argc, argv);
        if (attrstart&&argc)
        {
            jit_atom_arg_getsym(&dest_name_sym, 0, attrstart, argv);
        }

        // instantiate Jitter object with dest_name arg
        if (jit_ob = jit_object_new(
            gensym("jit_gl_simple"), dest_name_sym))
        {
            // set internal jitter object instance
            max_jit_obex_jitob_set(x, jit_ob);

            // add a general purpose outlet (rightmost)
            max_jit_obex_dumpout_set(x, outlet_new(x, NULL));

            // process attribute arguments
            max_jit_attr_args(x, argc, argv);

            // attach the jit object's ob3d to a new outlet
            // this outlet is used in matrixoutput mode
            max_jit_ob3d_attach(x, jit_ob, outlet_new(x, "jit_matrix"));
        }
        else
        {
            error("jit.gl.simple: could not allocate object");
            freeobject((t_object *)x);
            x = NULL;
        }
    }
    return (x);
}

```

```
}
```

```
void max_jit_gl_simple_free(t_max_jit_gl_simple *x)
```

```
{
```

```
    // lookup our internal Jitter object instance and free
```

```
    jit_object_free(max_jit_obex_jitob_get(x));
```

```
    // free resources associated with our obex entry
```

```
    max_jit_obex_free(x);
```

```
}
```

Chapter 24

OB3D Details

The purpose of this chapter is to fill in additional details of Jitter OpenGL, which we refer to as OB3Ds.

We will show how to disable and/or override default OB3D attributes and methods, how to support matrix input and output, and manage resources such as textures, display lists, and shaders. This chapter assumes familiarity with the OpenGL API and the OB3D Quick Start chapter. It is out of the scope of our documentation to cover the OpenGL API, so for information on the OpenGL API we recommend consulting the OpenGL Red Book and the many online tutorials.

24.1 Defining the OB3D Jitter Class

As covered in the OB3D Quick Start, Jitter OB3Ds have a large number of default attributes and methods, and require some specific methods to be defined. This section seeks to clarify these common attributes and methods and how to achieve custom behavior where necessary.

24.2 Declaring a Draw Method

All Jitter OB3Ds must define a method bound to the symbol `ob3d_draw`. This method takes no arguments in addition to the object struct, and should be defined with the private [A_CANT](#) type signature. The private `ob3d_draw` method will be called by the standard `draw`, and `drawraw` methods that are added to every OB3D. The `draw` method will set up OpenGL state associated with the default OB3D attributes before calling `ob3d_draw`, while the `drawraw` method will not.

24.3 Declaring Destination and Geometry Related Methods

It is possible for attributes of a Jitter OB3D or your render destination to change, requiring resources to be freed or rebuilt. There are three methods used to communicate to an OB3D which such events happen so that the OB3D can manage resources accordingly. They are: `dest_closing`, which informs an OB3D that the destination is being freed, and any context dependent resources such as textures, display lists, and shaders should be freed; `dest_changed`, which informs an OB3D that the destination has been rebuilt, and new resources can be allocated; and `rebuild_geometry`, which informs an OB3D of a change in texture units or some other attribute which affects `jit_gl_drawinfo_setup()` and other `t_jit_gl_drawinfo` related functions, such as `jit_gl_texcoord`, requiring geometry that uses such functions to be rebuilt. These methods take no arguments in addition to the object struct. The `dest_closing` and `dest_changed` methods should be defined with the private [A_CANT](#) type signature, and the `rebuild_geometry` method is typically defined as typed, but without arguments, so that users have the ability to explicitly call, if deemed necessary. The `jit.gl.gridshape` SDK project is a good example of these methods as it needs to free and allocate a display list as the render destination changes, and also makes use of `jit_gl_texcoord` to support multi-texturing, requiring geometry to be rebuilt as the number of texture units or other attributes change.

24.4 Declaring a Register Method

Since all Jitter OB3D objects are named to support reference by name in `jit.gl.sketch`, and other objects, it is necessary to add the default registration method, [`jit_object_register\(\)`](#). Object registration and notification are covered in detail in a future chapter.

24.5 Overriding Rotation and Scale Related Attributes

By default, each Jitter OB3D has rotate, rotatexyz, scale, and viewalign attributes added to the class by [jit_ob3d_setup\(\)](#), and these attributes are used in the `ob3d_draw_preamble()` function to set up OpenGL state prior to calling your object's draw method. These attributes can be disabled by using the [JIT_OB3D_NO_ROTATION_SCALE](#) flag. You can override these attributes by defining your own attributes of the same name, however, you will need to manage any necessary OpenGL state inside of your own draw method with the appropriate calls to `glMatrixMode`, `glTranslate`, `glRotate`, and `glScale`.

24.6 Overriding Color Related Attributes

By default, each Jitter OB3D has color, aux_color, and smooth_shading attributes added to the class by [jit_ob3d_setup\(\)](#), and these attributes are used in the `ob3d_draw_preamble` function prior to calling your object's draw method. These attributes can be disabled by using the [JIT_OB3D_NO_COLOR](#) flag. You can override these attributes by defining your own attributes of the same name, however, you will need to manage any necessary OpenGL state inside of your own draw method with the appropriate calls to `glColor` and `glShadeModel`.

24.7 Overriding Texture Related Attributes

By default, each Jitter OB3D has texture, capture, tex_map, tex_plane_s, and tex_plane_t attributes added to the class by [jit_ob3d_setup\(\)](#), and these attributes are used in the `ob3d_draw_preamble()` function prior to calling your object's draw method. These attributes can be disabled by using the [JIT_OB3D_NO_TEXTURE](#) flag. You can override these attributes by defining your own attributes of the same name, however, you will need to manage any necessary OpenGL state inside of your own draw method with the appropriate calls to `glEnable`, `glTexGen`, `jit_gl_bindtexture`, `jit_gl_unbindtexture`, `jit_gl_begincapture`, and `jit_gl_endcapture`.

24.8 Overriding Lighting and Material Related Attributes

By default, each Jitter OB3D has lighting_enable, auto_material, shininess, mat_ambient, mat_diffuse, mat_specular, and mat_emission attributes added to the class by [jit_ob3d_setup\(\)](#), and these attributes are used in the `ob3d_draw_preamble` function prior to calling your object's draw method. These attributes can be disabled by using the [JIT_OB3D_NO_LIGHTING_MATERIAL](#) flag. You can override these attributes by defining your own attributes of the same name, however, you will need to manage any necessary OpenGL state inside of your own draw method with the appropriate calls to `glEnable`, `glLight`, `glLightModel`, and `glMaterial`.

24.9 Overriding Fog Related Attributes

By default, each Jitter OB3D has fog and fog_params attributes added to the class by [jit_ob3d_setup\(\)](#), and these attributes are used in the `ob3d_draw_preamble` function prior to calling your object's draw method. These attributes can be disabled by using the [JIT_OB3D_NO_FOG](#) flag. You can override these attributes by defining your own attributes of the same name, however, you will need to manage any necessary OpenGL state inside of your own draw method with the appropriate calls to `glEnable`, `glHint`, and `glFog`.

24.10 Overriding Polygon Variable Related Attributes

By default, each Jitter OB3D has poly_mode, cull_face, point_size, and line_width attributes added to the class by [jit_ob3d_setup\(\)](#), and these attributes are used in the ob3d_draw_preamble function prior to calling your object's draw method. These attributes can be disabled by using the [JIT_OB3D_NO_POLY_VARS](#) flag. You can override these attributes by defining your own attributes of the same name, however, you will need to manage any necessary OpenGL state inside of your own draw method with the appropriate calls to glPolygonMode, glEnable, glCullFace, glPointSize, and glLineWidth.

24.11 Overriding Blending Related Attributes

By default, each Jitter OB3D has blend_mode and blend_enable attributes added to the class by [jit_ob3d_setup\(\)](#), and these attributes are used in the ob3d_draw_preamble function prior to calling your object's draw method. These attributes can be disabled by using the [JIT_OB3D_NO_BLEND](#) flag. You can override these attributes by defining your own attributes of the same name, however, you will need to manage any necessary OpenGL state inside of your own draw method with the appropriate calls to glEnable and glBindFunc.

24.12 Overriding Depth Buffer and Antialiasing Related Attributes

By default, each Jitter OB3D has depth_enable and antialias attributes added to the class by [jit_ob3d_setup\(\)](#), and these attributes are used in your ob3d_draw_preamble function prior to calling your object's draw method. These attributes can be disabled by using the [JIT_OB3D_NO_DEPTH](#) and [JIT_OB3D_NO_ANTIALIAS](#) flags, respectively. You can override these attributes by defining your own attributes of the same name, however, you will need to manage any necessary OpenGL state inside of your own draw method with the appropriate calls to glEnable and glHint.

24.13 Overriding Matrixoutput and Automatic Attributes

By default, each Jitter OB3D has matrixoutput and automatic attributes added to the class by [jit_ob3d_setup\(\)](#), and these attributes are used in the ob3d_draw_preamble function prior to calling your object's draw method. These attributes can be disabled by using the [JIT_OB3D_NO_MATRIXOUTPUT](#) and [JIT_OB3D_AUTO_ONLY](#) flags, respectively. You can override these attributes by defining your own attributes of the same name.

24.14 Declaring a User Interface Object

It is possible to declare a user interface OB3D, such as jit.gl.handle. To do so, you must use the [JIT_OB3D_DOES_UI](#) flag to [jit_ob3d_setup\(\)](#), and define a method bound to the symbol ob3d_ui, with the private [A_CANT](#) type signature and prototype similar to the following example from jit.gl.handle:

```
t_jit_err jit_gl_handle_ui(t_jit_gl_handle **x,
    t_line_3d *p_line, t_wind_mouse_info *p_mouse);
```

24.15 The Jitter Class Constructor and Destructor

Inside your Jitter class constructor, you must call `jit_ob3d_new()` with a pointer to your newly allocated object, and your render destination name. The `jit_ob3d_new()` function allocates an opaque structure that stores the standard OB3D attributes and some additional OB3D state, initializing them to default values, and then setting the pointer at the byte offset specified when calling the `jit_ob3d_setup()` function in your class definition. If your object supports matrix output or simply uses the `t_jit_glchunk` structure when drawing, you should typically allocate your initial `t_jit_glchunk` in your constructor using the `jit_glchunk_new()` or `jit_glchunk_grid_new()` functions. Use of the `t_jit_glchunk` structure and matrix output is described later in this chapter. Similarly, your OB3D Jitter class destructor must call `jit_ob3d_free()` to free the opaque structure used for common OB3D state, free any allocated instances of `t_jit_glchunk` with `jit_glchunk_free()`, and free any other resources allocated such as display lists or textures.

24.16 The OB3D Draw Method

The `ob3d_draw` method is where all the drawing in your object should take place. It is also where you should typically allocate context dependent resources or query the context state, since you know that your context is valid and has been set. For the most part, the drawing you will perform in your `ob3d_draw` method will be pure and simple OpenGL, though there are a few caveats which we will cover.

24.17 The `t_jit_glchunk` Structure and Matrix Output

Since Jitter is a general purpose matrix processing framework, it makes sense that you would have the ability to pass geometry information through a Jitter network as matrices if your geometry is well suited to a matrix representation. The cells of your matrix can hold vertex information such as position, texture coordinates, normal vectors, color, and edge flags, and are documented in the "Geometry Under The Hood" Jitter Tutorial. You also have the option of specifying a connections matrix to reference the connectivity of the vertices if it is not implicit in the matrix representation, and a drawing primitive to use when drawing the vertices.

All this information, and whether or not the geometry matrix should be rendered immediately or sent through the Jitter network is managed with the `t_jit_glchunk`. An SDK example which demonstrates the use of `t_jit_glchunk` is `jit.gl.gridshape`. The `t_jit_glchunk` structure along with the vertex matrix it contains is allocated by the `jit_glchunk_new()` or `jit_glchunk_grid_new()` functions, freed with the `jit_glchunk_delete()` function, and drawn with the `jit_ob3d_draw_chunk()` function. For reference, the `t_jit_glchunk` structure and relevant chunk flags are provided below:

```
// jit_glchunk is a public structure to store one
// gl-command's-worth of data, in a format which
// can be passed easily to glDrawRangeElements.

typedef struct _jit_glchunk
{
    t_symbol    *prim;           // GL_TRI_STRIP, GL_TRIANGLES, etc.
    t_jit_object *m_vertex;     // jit_matrix of xyzst... data.
    t_symbol    *m_vertex_name;  // vertex matrix name
    t_jit_object *m_index;      // optional 1d connection matrix
    t_symbol    *m_index_name;   // connection matrix name
    unsigned long m_flags;      // special flags
    void        *next_chunk;    // singly linked list, typically NULL
} t_jit_glchunk;

// flags for chunk creation
#define JIT_GL_CHUNK_IGNORE_TEXTURES      1 << 0
```

```
#define JIT_GL_CHUNK_IGNORE_NORMALS      1 << 1
#define JIT_GL_CHUNK_IGNORE_COLORS       1 << 2
#define JIT_GL_CHUNK_IGNORE_EDGES        1 << 3
```

24.18 OB3D OpenGL Caveats

While you can use any standard OpenGL calls inside of your ob3d_draw method. There are a few things worth noting to follow Jitter conventions. The first of which is the binding of texture coordinates. Since Jitter OB3Ds support multi-texturing by default, it is not necessarily satisfactory to submit only one texture coordinate with glTexCoord. Jitter provides some utility routines to set the texture coordinates for as many texture units which are bound, jit_gl_texcoord(1/2/3)(f/fv). Determining how many texture units have been bound by the default OB3D attributes requires some overhead, so rather than perform this overhead with every jit_gl_texcoord call, the jit_gl_texcoord functions take a [t_jit_gl_drawinfo](#) struct as an argument. This struct can be setup once before rendering many vertices with the jit_gl_drawinfo_setup function. Example use of jit_gl_texcoord and jit_gl_drawinfo_setup is in the jit.gl.videoplane SDK project. Another Jitter specific mechanism is the means to bind textures using named instances of jit.gl.texture. It is possible to create and bind your own textures in an OB3D, but you must then perform all maintenance instead of relying on jit.gl.texture to handle this work for you. To bind and unbind an instance of jit.gl.texture, you should call the jit_gl_bindtexture and jit_gl_unbindtexture functions, which take a [t_jit_gl_drawinfo](#) argument, a symbol with the name of the jit.gl.texture instance, and an integer for which texture unit to bind. Unlike binding ordinary textures in OpenGL, it is important to unbind instances of jit.gl.texture, or else problems may arise.

24.19 Getting Information About the OB3D Attributes

Though the default OB3D attributes are typically relevant to the code which is automatically handled for your object prior to calling the ob3d_draw method, it is sometimes necessary to access these values. Since the default OB3D attributes are stored in an opaque ob3d struct member, they are not accessible by your object with a simple struct pointer dereference. Instead, you need to use the jit_attr_get* functions to access these attributes. You should pass in your object struct as the first argument to these functions rather than your ob3d struct member. For example:

```
float pos[3];
jit_attr_getfloat_array(x, gensym("position"), 3, pos);
```

Note that if you are acquiring this value often, it is preferable to generate the symbol in advance rather than generate the symbol for every call.

24.20 Getting Information About the Context

From within the ob3d_draw, dest_closing, and dest_changed methods, the rendering context has always been set, and you can get a handle to the native context using either the aglGetCurrentContext or wglGetCurrentContext functions. One can also in these methods use standard OpenGL glGet* functions to determine the context's OpenGL state, such as the viewport, transformation matrix. It is not recommended to try and acquire the native context from other methods, or query the OpenGL state as it may not be valid.

24.21 Playing Well with Others

It is important to recognize that OpenGL state is persistent, and that there may be objects which rely on OpenGL state that are drawn after your object draws itself. If your object makes any changes to OpenGL state that might affect objects that follow, you should restore the OpenGL state to whatever it was before your routine was called. For example, if your object changes the texture transformation matrix, you should push and pop the texture transformation matrix with `glMatrixMode`, `glPushMatrix`, and `glPopMatrix`, to prevent any problems with other objects.

24.22 Defining the OB3D Max Wrapper Class

As mentioned in the OB3D Quick Start, in your Max wrapper class definition, you need only add a call to the `max_ob3d_setup()` function to add your standard drawing methods, and the `max_jit_ob3d_assist()` function as your assist method, unless you wish to define your own custom assist method. Everything else is similar to the standard technique of wrapping a Jitter Class demonstrated in the Max Wrapper Class chapter. Please consult the OB3D Quick Start chapter and the `jit.gl.simple` SDK project for all necessary information related to the OB3D Max wrapper class.

24.23 Matrix Input

Sometimes it is desirable for an OB3D also support incoming matrices as is the case with `jit.gl.videoplane` or `jit.gl.mesh`. It is not recommended to mix and match OB3Ds with MOPs. Conflicts arise with respect to arguments, standard inlets and outlets. Instead, if you wish to support matrix input in your OB3D, you should simply add to your Jitter class a method bound to the symbol `jit_matrix`, and handle the incoming matrix data according to your needs - for example as texture data in the case of `jit.gl.videoplane`, or geometry data in the case of `jit.gl.mesh`. The `jit.gl.videoplane` SDK project provides an example of an OB3D which also supports matrix input. When it is necessary to have multiple input matrices, this is typically managed by either declaring alternately named methods for each input, or exposing an attribute that specifies which input the `jit_matrix` method assumes it is being called with. Note that this requires additional logic within the Max wrapper class to map to inlets, as it is not handled automatically.

Chapter 25

Scheduler and Low Priority Queue Issues

In Max, there are a few threads of execution.

The details of these threads are highlighted in the Max documentation and the article, "Event Priority in Max (Scheduler vs. Queue)". In this chapter, we won't cover all these details and restrict our discussion to the scheduler (which when overdrive is on runs in a separate and high priority thread) and the low priority queue (which always runs in the main application thread). As far as Jitter is concerned, we won't consider the real time audio thread or the case of scheduler in audio interrupt, where the scheduler runs in this real time audio thread.

By default, Jitter performs all drawing and matrix processing in the main application thread, with events serviced from the low priority queue. The reason for this low priority processing is to prevent high timing events such as note triggering or audio DSP from suffering timing problems due to visual processing. Jitter also exploits the low priority queue as a mechanism for graceful temporal downsampling of the visual stream in the instance that the processing requested is too demanding to be calculated in real-time. This results in dropped frames in the output when the demands can't be met. With audio, it's not sufficient to just drop frames of samples, since there will be an audible click, but with images, the last image will persist if a new one isn't generated at some fixed sampling rate.

25.1 Defer and Usurp

The mechanisms which enforce execution of Jitter drawing and matrix processing from within the low priority queue we will call "defer" and "usurp". The defer mechanism will take any high priority events and create a corresponding low priority event at the end of the low priority queue. The defer mechanism ensures that the events will not be executed from the high priority scheduler thread, but does not prevent scheduler backlog with the temporal downsampling mentioned above. To accomplish this, the usurp mechanism must be used. The usurp mechanism will use no more than one low priority queue element for the task requested (either a method call or attribute setter). The way usurp works is that if there is no pending event for the method or attribute call, a new event is placed at the end of the low priority queue. If there is already an event pending, the usurp mechanism will not place a new event on the end of the low priority queue, but rather "usurp" the arguments for the event waiting to be passed to the method or attribute call. This way, if a high priority metronome is rapidly sending values to set an attribute, while the initial low priority event is waiting to be processed, the value to be set is constantly being updated ("usurped") and only the value at the time of servicing the event will be used.

It is important to note that the defer and usurp mechanisms only work as called from within the Max patcher. For any methods which are called from a text based programming language, such as C, Java, or JavaScript, the defer and usurp mechanisms are bypassed. This may be something you need to pay attention to and handle yourself if you are making such calls from a text based programming language and need the defer or usurp behavior.

25.2 Using Defer and Usurp in Jitter Object Methods

When defining a method in Jitter, there is the possibility to define a type signature for the method just as one would do in MaxMSP. Typical type signatures include typical atom elements such as `A_LONG`, `A_FLOAT`, and `A_SYM`; or the corresponding default value versions `A_DEFLONG`, `A_DEFLOAT`, `A_DEFSYM`; or the variable argument version `A_GIMME` which provides a list of atoms and the number of atoms provided; or the private and untyped status of `A_CANT` used for methods which are not exposed to the patcher and require additional C function prototype information in order to call. While these type signatures can be used within Jitter objects, most methods exposed to the patcher interface make use of either the defer or usurp mechanism as defined by two new type signatures `A_DEFER_LOW` or `A_USURP_LOW`. Methods defined with the `A_DEFER_LOW`, or `A_USURP_LOW` type signatures should conform to the same variable argument prototype as `A_GIMME` methods, but behind the scenes, Jitter will make use of

the defer and usurp mechanism to enforce the appropriate behavior.

An example of two methods from jit.gl.videoplane which use these mechanisms is below:

```
// add a usurping jit_matrix method
jit_class_addmethod(_jit_gl_videoplane_class, (method) jit_gl_videoplane_jit_matrix,
                    "jit_matrix", A_USURP_LOW, 0);

// add a deferred sendtexture method
jit_class_addmethod(_jit_gl_videoplane_class, (method) jit_gl_videoplane_sendtexture,
                    "sendtexture", A_DEFER_LOW, 0);
```

The implementation of these methods is below:

```
void jit_gl_videoplane_jit_matrix(t_jit_gl_videoplane *x, t_symbol *s, int argc,
                                   t_atom *argv)
{
    t_symbol *name;
    void *m;
    t_jit_matrix_info info;
    long dim[2];

    if ((name= jit_atom_getsym(argv)) != _jit_sym_nothing) {
        m = jit_object_findregistered(name);
        if (!m) {
            error("jit.gl.videoplane: couldn't get matrix object!");
            return;
        }
    }

    if (x->texture) {
        jit_object_method(m, _jit_sym_getinfo, &info);
        jit_attr_getlong_array(x->texture, _jit_sym_dim, 2, dim);
        jit_object_method(x->texture, s, s, argc, argv);
        jit_attr_setsym(x, ps_texture, x->texturename);
    }
}

void jit_gl_videoplane_sendtexture(t_jit_gl_videoplane *x, t_symbol *s, int argc,
                                   t_atom *argv)
{
    if (x->texture) {
        s = jit_atom_getsym(argv);
        argc--;
        if (argc)
            argv++;
        else
            argv = NULL;
        object_method_typed(x->texture, s, argc, argv, NULL);
    }
}
```

From inspecting the header files, you may note that there are also [A_DEFER](#) and [A_USURP](#) type signatures, but these should be considered obsolete, as they make use of the problematic deferral strategy of placing the event at the front of the low priority queue and have the potential of reversing message sequencing.

25.3 Using Defer and Usurp in Jitter Object Attributes

Unlike methods, attributes do not make use of type signatures for their getter and setter accessor methods. Instead they should always be prototyped similar to [A_GIMME](#), but with an attribute object being passed in place of the traditional method symbol pointer of the [A_GIMME](#) signature. So the way you can specify

to use the defer and usurp mechanisms for attribute accessors are through the attribute flags argument to the attribute constructor. For the getter accessor method, you can use `JIT_ATTR_GET_DEFER_LOW` or `JIT_ATTR_GET_USURP_LOW` flags. For the setter accessor method, you can use `JIT_ATTR_SET_DEFER_LOW` or `JIT_ATTR_SET_USURP_LOW` flags.

An example attribute definition from jit.gl.videoplane is below:

```
attrflags = JIT_ATTR_GET_DEFER_LOW | JIT_ATTR_SET_USURP_LOW;
attr = jit_object_new(_jit_sym_jit_attr_offset,"displaylist",_jit_sym_char,attrfl
    ags,
    (method) 0L, (method) jit_gl_videoplane_displaylist,calcoffset(t_jit_gl_videoplan
        e, displaylist));
jit_class_addattr(_jit_gl_videoplane_class,attr);
```

You may have noticed that like previous code example, all Jitter object attributes which are not private have been defined with getter accessors which use the defer mechanism (`JIT_ATTR_GET_DEFER_LOW`) and setter accessors which use the usurp mechanism (`JIT_ATTR_SET_USURP_LOW`). This is the recommended style of exposing Jitter object attributes to the patcher, since there are many cases where at high priority an attribute is set repeatedly and we want both the latest high priority value when the next calculation is made at low priority and no low priority queue backlog from generating more events at high priority than can be processed at low priority. The defer mechanism is used for getter accessor methods so that every attribute query results in a corresponding output message out the dump outlet. Otherwise certain patcher logic could easily become confused. If a different behavior is required by the Max programmer, they can make use of the jit.qball object to force either the defer or usurp mechanisms to be used for their message stream.

25.4 Using Defer and Usurp in the Max Wrapper Object

Most of the above is also true when declaring methods and attributes in the Max wrapper object, however the function calls which are used are slightly different. You must use the special max object function calls `max_addmethod_defer_low()` and `max_addmethod_usurp_low()` for methods, and `max_jit_classex_addattr()` for attributes. Below are examples from jit.matrixset. Note that there is no type signature provided for either `max_addmethod_defer_low()` or `max_addmethod_usurp_low()`.

```
// add a deferred "exportmovie" method
max_addmethod_defer_low((method)max_jit_matrixset_export_movie, "exportmovie");

// add a usurped outputmatrix method
max_addmethod_usurp_low((method)max_jit_matrixset_outputmatrix,      "outputmatrix"
);

// add index attribute
attrflags = JIT_ATTR_GET_DEFER_LOW | JIT_ATTR_SET_USURP_LOW ;
attr = jit_object_new(_jit_sym_jit_attr_offset,"index",_jit_sym_long,attrflags,
    (method) 0L,(method) 0L,calcoffset(t_max_jit_matrixset,index));
max_jit_classex_addattr(p,attr);
```

25.5 When Not to Use the Usurp Mechanism

The bang method for Jitter MOP objects uses the usurp mechanism to drop frames when the number of bang messages cannot be handled in real time. However, jit.gl.render's bang method does not behave this way, and instead uses the defer mechanism. At first this might seem counterintuitive, however, because rendering in OpenGL with jit.gl.render uses a group of messages to perform erasing, any non automatic drawing of objects, and then a drawing of automatic clients and a swap to the screen with the bang method,

it is not an atomic action (i.e. requires a sequence of different events rather than a single event). Since the usurp mechanism is method or attribute specific with regard to the events which are being usurped, it only works for atomic actions. For this reason, it is important for users to perform some drop framing behavior before triggering the message sequence, typically accomplished with qmetro or jit.qball. If your object has some operation which requires a sequence of events in a similar fashion as jit.gl.render, then it would be best to use the defer mechanism rather than the usurp mechanism for relevant methods.

25.6 Overriding Defer and Usurp

There are instances where the user does not wish to be limited to processing Jitter matrices at low priority, such as when Jitter matrices are used for tasks other than realtime image processing—for example, parameter interpolation or matrices containing audio data. For these tasks, the jit.qfaker object is provided for advanced users which are aware of the potential problems involved in bypassing these low priority mechanisms. As mentioned above, when programming in a text based language, these mechanisms aren't used and all method and attribute accessor calls are synchronous. Therefore there typically isn't a need to consider overriding this behavior from a text based language. However, for certain externals which wish to simulate the jit.qfaker behavior, we expose the max_jit_queuestate() function to override Jitter's detection of queue state for the defer and usurp mechanisms. It is also possible to query what jitter believes the queue state to be with the max_jit_getqueuestate() function. This is the function employed by the defer and usurp mechanisms. The source code for these functions is below for reference.

```
long max_jit_queuestate(long state)
{
    long rv=_max_jit_queuestate;
    _max_jit_queuestate = (state!=0);
    return rv;
}

long max_jit_getqueuestate(void)
{
    // always return true if faking
    if (_max_jit_queuestate) return 1;
    return !sched_isinpoll();
}
```


Chapter 26

Jitter Object Registration and Notification

In Jitter, matrices are passed around as named references between Max objects.

This named reference is created since Jitter registers these matrices with the corresponding name using the [jit_object_register\(\)](#) function. Object registration is useful for a few reasons. First, registered matrices can be resolved by name using the [jit_object_findregistered\(\)](#) function. Secondly, registered objects can send event notification to clients who have attached to them using [jit_object_attach\(\)](#). Lastly, under certain circumstances, the object registration process can be used to have multiple external references to a single instance of an object as is the case with jit.matrix.

26.1 Registering Named Objects

To register an object, one can use the [jit_object_register\(\)](#) function, which is equivalent to the Max [object_register\(\)](#) function in the namespace associated with gensym("jitter"). Traditionally in Jitter, we bind [jit_object_register\(\)](#) to the "register" method for an object and use [jit_object_method\(\)](#) to call this method. For example, from the jit.notify SDK example:

```
// allocate the Jitter object
if (o=jit_object_new(gensym("jit_notify"))) {
    ...
    // generate a unique name
    x->servername = jit_symbol_unique();

    // register the object with the given name
    jit_object_method(o,_jit_sym_register,x->servername);
    ...
}
```

If not using a specific name, it is good to use the [jit_symbol_unique\(\)](#) function as above to generate a unique name which is slated for re-use once a registered object is freed. This prevents excess memory usage by the symbol table as associated with these unique names.

If you wish the object to have multiple references to a single instance with some name, as is common with the jit.matrix object, it is essential to use the return value of [jit_object_register\(\)](#) in any instance where the object pointer is saved after registration. This is because if the registered object with the same class already exists, the object attempting to be registered will be freed, and the already registered object of the same class will be returned, its reference count having been incremented. This is not typically an issue outside of registering jit.matrix objects, although you may have a need for this type of implementation in other situations. Most other situations in which object registration is used within Jitter only expects and/or permits a single instance to be registered. In the above example, we know that this is safe to do, as we are using [jit_symbol_unique\(\)](#) to generate a unique name.

It is also possible to unregister named objects, with the [jit_object_unregister\(\)](#) function, but typically this is handled for you when your object is freed, or if your object is registered again with a different name. This is not often used in the Jitter code base except within these contexts.

26.2 Looking Up an Object by Name

Registered objects can be found by name using the [jit_object_findregistered\(\)](#) function. For example named matrices are resolved using this function. Most Matrix Operator objects have this done for them by the default MOP code, but for example any MOP which has its own jit_matrix method, such as the jit.pack SDK example will make use of [jit_object_findregistered\(\)](#) inside its jit_matrix method:

```
// get our matrix name from the atom arguments provided
matrixname = jit_atom_getsym(argv);
```

```

// look up based on name
matrix = jit_object_findregistered(matrixname);

// make sure that it is a valid pointer and has a "class_jit_matrix" method which
// returns 1
if (matrix&&jit_object_method(matrix, _jit_sym_class_jit_matrix)) {
    ...
}

```

26.3 Attaching to Named Objects

Once an object has been registered, it can be considered a server to which clients attach to be notified of various events. To attach to a named object, use the the [jit_object_attach\(\)](#) function. Similarly to detach from a named object, use the [jit_object_detach\(\)](#) function. It is typical to detach from a server in your object's destructor, or any time your object is switching which server it is attached to. For your client object to receive any notification from the server object, it is important for your object to have defined a "notify" method which will receive the notification from all objects it is attached to.

Below is the jit.notify SDK example's max wrapper object's notify method, which receives some atom values from its internal Jitter object instance. Since this object is a Matrix Operator, it is important in the following example that jit.notify calls the [max_jit_classex_mop_wrap\(\)](#) function with the [MAX_JIT_MOP_FLAGS_OWN_NOTIFY](#) flag to override the default MOP notify method, and that we pass on all other messages to the standard [max_jit_mop_notify\(\)](#) method so that the default MOP code is informed of any changes to the input and output matrices.

```

// s is the servername, msg is the message, ob is the server object pointer,
// and data is extra data the server might provide for a given message
void max_jit_notify_notify(t_max_jit_notify *x, t_symbol *s, t_symbol *msg, void
    *ob, void *data)
{
    if (msg==gensym("splat")) {
        post("notify: server=%s message=%s", s->s_name, msg->s_name);
        if (!data) {
            error("splat message NULL pointer");
            return;
        }
        // here's where we output using the rightmost outlet
        // we just happen to know that "data" points to a t_atom[3]
        // alternately you could use max_jit_obex_dumpout_get just to get
        // the outlet pointer
        max_jit_obex_dumpout(x, msg, 3, (t_atom *)data);
    } else {
        // since we are a MOP, we are also attached to all the matrices for each in
        // put/output
        // so we need to deal with this by calling the default mop notify method
        // (this is how mops handle their matrices getting new names/freed/modified
        )
        max_jit_mop_notify(x, s, msg);
    }
}

```

26.4 Notifying Clients

If you are making an object which is to be registered, and wish to send custom notification to clients in addition to the default notification that attributes send to all clients when the attribute is modified, and the default object free notification, then you will want to use the [jit_object_notify\(\)](#) function. This function lets you determine a message name to use for notification and optionally specify additional, but untyped data

to all clients. If you choose to send additional data to clients, it is necessary for all client code to know how to unpack this information. Below is the example from the jit.notify SDK example which uses the notification mechanism to send some data to its max wrapper object:

```
t_atom foo[3];  
  
jit_atom_setlong(&foo[0],1);  
jit_atom_setlong(&foo[1],2);  
jit_atom_setlong(&foo[2],3);  
jit_object_notify(x,gensym("splat"), foo);
```

Chapter 27

Using Jitter Objects in C

When developing for Jitter in C, the functionality of pre-existing Jitter objects can be used.

In this chapter, we'll briefly examine instantiation and incorporation of the features of the jit.qt.movie and jit.qt.record objects from your C code.

27.1 Example 1: the t_jit_qt_movie object

Using an object like t_jit_qt_movie from your own code is fairly straightforward. Since it's a standard Jitter object, we can use `jit_object_new()` and `jit_object_free()` for instantiation and freeing, `jit_object_method()` for sending messages, and `jit_attr_get...` and `jit_attr_set...` for getting and setting attributes.

For instance, in the following code snippet, we'll create a t_jit_qt_movie object, read a pre-specified movie from disk, and decompress its first frame into a matrix, set to the native size of the movie.

```
void jit_foo_read_first_movie_frame(
    t_jit_foo *x, t_symbol *s, long ac, t_atom *av)
{
    void *qtmovie;

    // create the t_jit_qt_movie object, sized to 1x1
    qtmovie = jit_object_new(gensym("jit_qt_movie"), 1, 1);
    if (qtmovie) {
        t_atom rv; // will contain rvarr, with any return values
                   // from our "read" call
        t_object *rvarr; // the t_atomarray with the actual
                         // return values

        // turn off autostart
        jit_attr_setlong(qtmovie, gensym("autostart"), 0);
        // read the movie, just pass in the args to our function
        object_method_typed(qtmovie, gensym("read"), ac, av, &rv);

        // check the return value & verify that the movie loaded
        if (rvarr = jit_atom_getobj(&rv)) {
            long rvac = 0;
            t_atom *rvav = NULL;

            object_getvalueof(rvarr, &rvac, &rvav);
            if (rvac && rvav) {
                // just as in Max, we get a list: "filename success";
                // success of 1 means the read was successful
                if (rvac > 1 && jit_atom_getlong(rvav + 1)) {
                    long dim[2];
                    void *matrix;
                    t_jit_matrix_info info;

                    // get our movie's native dims
                    jit_attr_getlong_array(qtmovie, gensym("movie_dim"),
                        2, dim);
                    // set the t_jit_qt_movie's dim to match
                    jit_object_method(qtmovie, _jit_sym_dim, dim[0], dim[1]);
                    // set our matrix up to match
                    jit_matrix_info_default(&info);
                    info.type = _jit_sym_char;
                    info.planecount = 4;
                    info.dimcount = 2;
                    info.dim[0] = dim[0];
                    info.dim[1] = dim[1];
                    matrix = jit_object_new(_jit_sym_jit_matrix, &info);
                    if (matrix) {
                        // call the t_jit_qt_movie's matrix_calc method
                        // with our matrix as an argument
                        err = (t_jit_err) jit_object_method(qtmovie,
```

```
    _jit_sym_matrix_calc, NULL, matrix);
    if (err != JIT_ERR_NONE) {
        error("something went wrong");
    }
    // do something with the matrix

    // free the matrix
    jit_object_free(matrix);
}
}
freebytes(rvav, sizeof(t_atom) * rvac);
}
freeobject(rvarr);
}
jit_object_free(qtmovie);
}
}
```

Naturally, we could also set the `t_jit_qt_movie` object's time attribute, or call its `or` frame method, to recall an arbitrary point in time. In fact, nearly every documented method and attribute of the `jit.qt.movie` object, as it functions in the Max interface, is available from C. The exceptions are those functions implemented in the Max wrapper object, such as `framedump`.

Chapter 28

JXF File Specification

The Jitter File Format (JXF) stores matrix data in a binary (not human-readable) form.

When using Jitter you can create JXF files by sending the write message to a jit.matrix object. Conversely you can read JXF files from disk using the read message. This section will cover first the API functions that one can use from C to read and write JXF files. Then it will break down the file format at the bit level.

28.1 The Binary JXF API

Most Jitter users do not need or want to know about the internal binary format of a JXF-file. Even users who want to read and write JXF-files from C do not need to know the internal details if they use the functions of the Jitter API for the binary interface. Not only is the API more convenient, but using the functions provided by Cycling '74 may protect your code from having to be altered in the future in the event of a specification change.

There are two primary functions one should use to read data from a JXF file. [jit_bin_read_header\(\)](#) reads the version number and the size of the file from the header, and has the following signature:

```
t_jit_err jit_bin_read_header(t_filehandle fh, ulong *version, long *filesize)
```

[jit_bin_read_matrix\(\)](#) imports matrix data from a file to a matrix, resizing the matrix if necessary, and has the following signature:

```
t_jit_err jit_bin_read_matrix(t_filehandle fh, void *matrix)
```

Here's a chunk of code that shows how to read a matrix from disk:

```
if (!(err=path_opensysfile(filename, path, &fh, READ_PERM))) {
    //all is well
} else {
    error("jit.matrix: can't open file %s",name->s_name);
    goto out;
}
if (jit_bin_read_header(fh,&version,&filesize)) {
    error("jit.matrix: improper file format %s",name->s_name);
    sysfile_close(fh);
    goto out;
}
if (jit_bin_read_matrix(fh,matrix)) {
    error("jit.matrix: improper file format %s",name->s_name);
    sysfile_close(fh);
    goto out;
}
sysfile_close(fh);
```

Similarly there are two functions one should use when writing data to a JXF file. [jit_bin_write_header\(\)](#) writes a header to a file, and has the following signature:

```
t_jit_err jit_bin_write_header(t_filehandle fh, long filesize)
```

[jit_bin_write_matrix\(\)](#) writes a matrix to a file, and has the following signature:

```
t_jit_err jit_bin_write_matrix(t_filehandle fh, void *matrix)
```

Here's a section of code that shows how you might write a file with one matrix. Note that the initial filesize argument to [jit_bin_write_header\(\)](#) is bogus, but that the header is written again at the end of the operation when the filesize can be determined from the file position after writing the matrix.

```

if (err=path_createsysfile(filename, path, type, &fh)) {
    error("jit.matrix: could not create file %s",name->s_name);
    goto out;
}
if (jit_bin_write_header(fh,0)) {
    error("jit.matrix: could not write header %s", matrixName->s_name);
    sysfile_close(fh);
    goto out;
}
if (jit_bin_write_matrix(fh,pointerToMatrix)) {
    error("jit.matrix: could not write matrix %s", matrixName->s_name);
    sysfile_close(fh);
    goto out;
}
sysfile_getpos(fh, &position);
sysfile_seteof(fh, position);
if (jit_bin_write_header(fh,position)) {
    error("jit.matrix: could not write header %s",
matrixName->s_name);
    sysfile_close(fh);
    goto out;
}
sysfile_close(fh);

```

28.2 Specification of the JXF Format

The internal format of JXF-files is based on the Interchange File Format (IFF) (http://en.wikipedia.org/wiki/Interchange_File_Format). An IFF file is built up from chunks. All data in IFF files is big-endian. Several convenience macros defined in jit.byteorder.h are available to help convert numbers to the proper format before and after they're written to and read from a JXF file: BE_I32() can be called on 32-bit integers, BE_F32() on 32-bit floats, and BE_F64() on 64-bit doubles.

Each chunk in an IFF file begins with a four character Type ID. This is followed by a 32-bit unsigned integer specifying the size of the chunk content in bytes. In a JXF file, the 32-bit integer part of the first chunk tells us the size of the file, and all the subsequent chunks, which begin immediately after the first chunk, contain matrices. In the future chunks may also be used to store other kinds of data.

Here is a tabular overview of an example minimal JXF file.

Container Chunk

groupID	JIT_BIN_CHUNK_CONTAINER ('FORM')
File size	32-bit int
IFF Type	JIT_BIN_FORMAT ('JIT!')
Format Chunk	
chunkID	JIT_BIN_CHUNK_FORMAT_VERSION ('FVER')
Chunk size	12 bytes
Version	JIT_BIN_VERSION_1 (0x3C93DC80)
Matrix Chunk	
chunk ID	JIT_BIN_CHUNK_MATRIX ('MTRX')
chunk size	32-bit int
offset	32-bit int
type	4-char
planecount	32-bit int
dimcount	32-bit int
dim	Array of 32-bit ints that contain the dimensions
data	

The data offset of the matrix chunk represents the offset, in bytes, from the beginning of the chunk to the beginning of the data portion of the chunk. The type is one of CHAR, LONG, FL32 and FL64. The dim array contains dimcount elements, each of which is a 32-bit int. The data portion consists of the cells of the matrix written out one at a time in row-major order. Planar data is multiplexed in each cell. For example, a 3-plane 2 by 2 matrix would be written out in the following order:

Plane	Dim 0	Dim 1
0	0	0
1	0	0
2	0	0
0	1	0
1	1	0
2	1	0
0	0	1
1	0	1
2	0	1
0	1	1
1	1	1
2	1	1

The various chunks discussed above can be represented by the C structs listed below:

```

typedef struct _jit_bin_chunk_container
{
    ulong    ckid;        //'FORM'
    long     cksize;      //filesize
    ulong    formtype;   //'JIT!'
} t_jit_bin_chunk_container;

typedef struct _jit_bin_chunk_format_version
{
    ulong    ckid;        //'FVER'
    long     cksize;      //12
    ulong    vers;        //timestamp
} t_jit_bin_chunk_format_version;

typedef struct _jit_bin_chunk_matrix
{
    ulong    ckid;        //'MTRX'
    long     cksize;      //varies(should be equal to

```

```
//24+(4*dimcount)+(typesize*planecount*totalpoints))
long    offset;           //data offset(should be equal to 24+(4*dimcount))
ulong   type;            //'CHAR','LONG','FL32','FL64'
long    planecount;
long    dimcount;
long    dim[1];
} t_jit_bin_chunk_matrix;
```


Chapter 29

Jitter Networking Specification

This appendix describes the format of the data sent by a jit.net.send object.

The object attempts to form a TCP connection with a host at the IP and port specified by the object's attributes. Any program wishing to receive data will therefore have to set itself up as a host and listen for incoming TCP connections.

Once a connection is formed, data can be sent. Data is sent as a stream of chunks. The first thing received will be a chunk header. It consists of a 32-bit chunk ID and a 32-bit int representing the size of the next chunk to come. The chunk ID can be one of the following 4-char symbols, depending on what kind of packet it is:

```
#define JIT_MATRIX_PACKET_ID    'JMTX'
#define JIT_MATRIX_LATENCY_PACKET_ID   'JMLP'
#define JIT_MESSAGE_PACKET_ID    'JMMP'
```

This chunk header could be represented in C by the following struct:

```
typedef struct _jit_net_packet_header
{
    long id;
    long size; //size of packet to come
} t_jit_net_packet_header;
```

If the chunk is a matrix packet, the next data received will be a header of 288 bytes with the following contents:

id	'JMTX'
Size	288 (32-bit int, size of this header)
PlaneCount	32-bit int
Type	32-bit int, 0 for char, 1 for long, 2 for float32, 3 for float64
DimCount	32-bit int
Dim	Array of 32 32-bit ints
DimStride	Array of 32 32-bit ints
DataSize	32-bit int, size of the data buffer to come
Time	64-bit double precision float

This chunk could be represented with the following C struct:

```
typedef struct _jit_net_packet_matrix
{
    long    id;
    long    size;
    long    planeCount;
    long    type;           //0=char,1=long,2=float32,3=float64
    long    dimCount;
    long    dim[JIT_MATRIX_MAX_DIMCOUNT];
    long    dimStride[JIT_MATRIX_MAX_DIMCOUNT];
    long    dataSize;
    double  time;

} t_jit_net_packet_matrix;
```

Following this header the next data received will be the matrix data, the size of which was passed in the above header. When using the data, please note the dimstrides transmitted in the header.

The time field in the above header will be set to the time of transmission from the sending computer. jit.net.send expects the server to respond by sending back timing data of its own – it uses this data to estimate the transmission latency. The exact data in the latency chunk that jit.net.send expects to receive is the following:

id	'JMLP'
client_time_original	64-bit double, the time value received in the matrix header packet
server_time_before_data	64-bit double, the time on the server when the packet header is received
server_time_after_data	64-bit double, the time on the server after the packet has been processed and is in use

This chunk can be represented by the following C struct:

```
typedef struct _jit_net_packet_latency
{
    long id;
    double client_time_original;
    double server_time_before_data;
    double server_time_after_data;
} t_jit_net_packet_latency;
```

The difference between the server time before and server time after processing the data represents the time it takes the server to mobilize the data after it has been received. jit.net.send will send and expects to receive time in milliseconds. When this timing information is received by the transmitting computer, it notes its current time, calculates the round trip time and then estimates the latency as half the round trip time plus half of the server processing time. This estimate is accurate if the time of flight from A to B is the same as the time of flight from B to A, but network topology can be very complicated, and often the route from A to B is not the reverse of the route from B to A. In simple situations, such as a direct connection between two computers or a small LAN, the estimate should be reasonably accurate.

Finally, the last type of packet that can be sent is the message packet. The size of the message packet is sent in the initial header packet. Standard [A_GIMME](#) messages (`t_symbol *s, long ac, t_atom *av`) are serialized starting with a 32-bit integer that contains the size of the serialized message in bytes. Following that another 32-bit integer gives the argument count for the atoms. Following that comes the message atoms themselves, starting with the leading symbol if it exists. Each atom is represented in memory first with a char that indicates what type of atom it is: 's' for symbol, 'l' for long, and 'f' for float. For long and float atoms, the next 4 bytes contain the value of the atom; for symbol atoms a null terminated character string follows. Below is a C function that will deserialize a message passed in as a data pointer.

```
void gimme_deserialize(char *data, t_symbol **s, long *ac, t_atom **av)
{
    char *curr = data;
    float *currf;
    long *currL, i;
    long datasize = BE_I32(*((long *)curr));
    curr += sizeof(long);
    *ac = BE_I32(*((long *)curr));
    curr += sizeof(long);
    *av = (t_atom *)sysmem_newptr(sizeof(t_atom) * (*ac));

    if (*curr == ATOM_SERIALIZATION_SYMBOL_CODE)
    {
        curr++;
        *s = gensym(curr);
        while (*(++curr) != '\0') ;
        curr++;
    }
    else
        *s = 0L;
    for (i=0;i<*ac;i++)
        switch (*curr++)
    {
        case ATOM_SERIALIZATION_SYMBOL_CODE:
```

```
(*av)[i].a_type = A_SYM;
(*av)[i].a_w.w_sym = gensym(curr);
while (*(++curr) != '\0') ;
curr++;
break;
case ATOM_SERIALIZATION_FLOAT_CODE:
    (*av)[i].a_type = A_FLOAT;
    (*av)[i].a_w.w_float = BE_F32(*((float *)curr));
    curr += sizeof(float);
    break;
case ATOM_SERIALIZATION_LONG_CODE:
    (*av)[i].a_type = A_LONG;
    (*av)[i].a_w.w_long = BE_I32(*((long *)curr));
    curr += sizeof(long);
    break;
}
}
```

Chapter 30

Appendix: Messages sent to Objects

When writing objects for Max, you typically think of creating methods which are called when a message is sent to your object through the object's inlet.

However, your object may receive messages directly from Max rather than using the inlet.

One common example is the "assist" message, which is sent to your object when a user's mouse cursor hovers over one of your object's inlets or outlets. If your object binds a method to the "assist" message then you will be able to customize the message that is shown.

This appendix serves as a quick reference for messages that are commonly sent to objects by Max, should they be implemented by the given object. Where possible, the prototypes given are actual prototypes from example objects in the SDK rather than abstractions to assist in finding the context for these calls.

30.1 Messages for All Objects

acceptsdrag_locked	long pictmeter_acceptsdrag_unlocked(t_pictmeter *x, t_object *drag, t_object *view);	
acceptsdrag_unlocked	long pictmeter_acceptsdrag_unlocked(t_pictmeter *x, t_object *drag, t_object *view);	
assist	void pictmeter_assist(t_pictmeter *x, void *b, long m, long a, char *s);	
dumpout		bind this message to object_obex_dumpout() rather than defining your own method.
inletinfo	void my_obj(t_object *x, void *b, long a, char *t)	you may bind to stdinletinfo() or define your own inletinfo method. The 'b' parameter can be ignored, the 'a' parameter is the inlet number, and 1 or 0 should set the value of '*t' upon return.
notify	t_max_err dbviewer_notify(t_dbviewer *x, t_symbol *s, t_symbol *msg, void *sender, void *data);	
quickref		obsolete, this is provided automatically now

30.2 Messages for Non-UI Objects

dblclick	void scripto_dblclick(t_scripto *x);	
----------	---	--

30.3 Messages for User Interface Objects

getdrawparams	void uisimp_getdrawparams(t_ -uisimp *x, t_object *patcherview, t_jboxdrawparams *params);	
mousedown	void scripto_ui_mousedown(t_ -scripto_ui *x, t_object *patcherview, t_pt pt, long modifiers);	
mouseup	void uisimp_mouseup(t_uisimp *x, t_object *patcherview, t_pt pt, long modifiers);	
mousedown	void scripto_ui_mousedown(t_ -scripto_ui *x, t_object *patcherview, t_pt pt, long modifiers);	
mouseenter	void uisimp_mouseenter(t_uisimp *x, t_object *patcherview, t_pt pt, long modifiers);	
mouseleave	void uisimp_mouseleave(t_uisimp *x, t_object *patcherview, t_pt pt, long modifiers);	
mousemove	void uisimp_mousemove(t_uisimp *x, t_object *patcherview, t_pt pt, long modifiers);	
paint	void pictmeter_paint(t_pictmeter *x, t_object *patcherview);	

30.4 Message for Audio Objects

dsp	void plus_dsp(t_plus *x, t_signal **sp, short *count);	
dspstate	plus_dspstate(t_plus *x, long n);	

30.5 Messages for Objects Containing Text Fields

key	long uitextfield_key(t_uittextfield *x, t_object *patcherview, long keycode, long modifiers, long textcharacter);	
keyfilter	long uitextfield_keyfilter(t_- uittextfield *x, t_object *patcherview, long *keycode, long *modifiers, long *textcharacter);	
enter	void uitextfield_enter(t_uittextfield *x);	
select	void uitextfield_select(t_uittextfield *x);	

30.6 Messages for Objects with Text Editor Windows

edclose	void simpletext_edclose(t_- simpletext *x, char **text, long size);	
---------	---	--

30.7 Messages for Dataview Client Objects

getcelltext	void dbviewer_getcelltext(t_- dbviewer *x, t_symbol *colname, long index, char *text, long maxlen);	
newpatcherview	void dbviewer_- newpatcherview(t_dbviewer *x, t_object *patcherview);	
freepatcherview	void dbviewer_- freepatcherview(t_dbviewer *x, t_object *patcherview);	

Chapter 31

Appendix: Providing Icons for UI Objects

If you are writing user interface objects for Max, it is recommended that you provide an icon for your object.

Providing an icon will allow users to create an instance of your class from the object palette, and improve the user's experience in other interactions with Max including the Object Defaults inspector.

31.1 Object SVG Icon

To see the icons provided by Cycling '74 for objects included in Max, look in the **Cycling '74/object-palettes** folder installed by Max. You will find a variety of SVG (scalable vector graphics) files for the objects. The files are named with the same name of the class (as it is defined in your `main()` function) with which they are associated. You will need to place your svg in this folder for it to be found by Max.

SVG files can be edited in a variety of software applications such as InkScape or Adobe Illustrator. You can also export SVG files from OmniGraffle on the Mac, which is how the Max's object icons were created.

31.2 Object Palette Definition

Adding the svg file will make the icon available to Max for use in some ways. To make your icon appear in the new object palette, however, you must create a palette containing your SVG file. If you look in the **Cycling '74/object-palettes** folder (where you placed your SVG file), you should notice some files with names like "palette1.json", "palette2.json", and "palette3.json". For your object, you should create a new palette file.

For the following example we will assume you have created an object called 'littleuifoo'. For this object we will create a palette called 'littleuifoo-palette.json'. The contents of this file will look like this:

```
{
  "patcher": {
    "rect": [ 0.000000, 0.000000, 1000.000000, 1000.000000 ],
    "bgcolor": [ 1.000000, 1.000000, 1.000000, 1.000000 ],
    "bglocked": 0,
    "defrect": [ 10.000000, 59.000000, 1176.000000, 668.000000 ],
    "boxes": [
      {
        "box": {
          "maxclass": "fpic",
          "boxalpha": 1.000000,
          "presentation": 0,
          "destrect": [ 0.000000, 0.000000, 0.000000, 0.000000 ],
          "patching_rect": [ 241.000000, 244.000000, 100.000000, 50.000000 ],
          "autofit": 0,
          "id": "obj-1",
          "ignoreclick": 0,
          "hidden": 0,
          "fontname": "Courier",
          "pic": "littleuifoo.svg",
          "xoffset": 0.000000,
          "yoffset": 0.000000,
          "background": 0,
          "presentation_rect": [ 0.000000, 0.000000, 0.000000, 0.000000 ],
          "fontsize": 12.000000,
          "instance_attributes": {
            "palette_category": [ "Images", "Interface" ],
            "palette_action": "littleuifoo"
          }
        }
      }
    ]
  }
}
```

```
}

}

],
"lines" : [ ]
}
}
```

Most of this palette file will be the same for any given object. The astute reader might notice that this is a JSON representation of a [t_dictionary](#) representing a patcher that includes an **fpic** object. We care about three lines in this dictionary:

1. The 'pic' attribute of the fpic object determines what image will be displayed in the new-object palette.
2. The 'palette_category' instance attribute will determine what categories/tabs your icon will appear under in the new-object palette.
3. The 'palette_action' instance attribute will determine what object class is instantiated by when user chooses your icon.

Chapter 32

Appendix: Additional Resources

While it is out of the scope of this document to cover many topics related to Jitter development, we suggest the following resources to better inform your development.

The C Programming Language:

- "The C Programming Language", Kernighan and Ritchie (Prentice Hall, 1988). ISBN: 0131103709
- "A Book on C", Kelly and Pohl (Addison Wesley, 1997). ISBN: 0201183994
- [Wikipedia's C programming language resources](#)

Object Oriented Programming:

- [Wikipedia's Object Oriented Programming resources](#)
- [Sun's Object Oriented Programming Concepts Tutorial](#)
- [Object Oriented Programming in C](#)

Digital Image Processing:

- "Handbook of Image and Video Processing", A. Bovik et al. (Academic Press, 2000). ISBN: 0121197921
- "Digital Image Processing", W. K. Pratt (John Wiley and Sons, 2001). ISBN: 0471857661
- "Principles of Digital Image Synthesis", A. S. Glassner (Morgan Kaufmann, 1995). ISBN: 1558602763
- [Wikipedia's digital image processing resources](#)

Open GL:

- [The official OpenGL web portal](#)

Apple and QuickTime:

- [Apple's developer resources](#)

Microsoft:

- [Micrcosoft's developer resources](#)

Chapter 33

Module Documentation

33.1 Attributes

An attribute of an object is a setting or property that tells the object how to do its job.

Data Structures

- struct `t_attr`

Common attr struct.

Defines

- #define `CLASS_ATTR_CHAR`(c, attrname, flags, structname, structmember) class_-
addattr((c),attr_offset_new(attrname,USESYM(char),(flags),(method)0L,(method)0L,calcoffset(structname,structmembe

Create a char attribute and add it to a Max class.

- #define `CLASS_ATTR_LONG`(c, attrname, flags, structname, structmember) class_-
addattr((c),attr_offset_new(attrname,USESYM(long),(flags),(method)0L,(method)0L,calcoffset(structname,structmembe

Create a long integer attribute and add it to a Max class.

- #define `CLASS_ATTR_FLOAT`(c, attrname, flags, structname, structmember) class_-
addattr((c),attr_offset_new(attrname,USESYM(float32),(flags),(method)0L,(method)0L,calcoffset(structname,structmembe

Create a 32-bit float attribute and add it to a Max class.

- #define `CLASS_ATTR_DOUBLE`(c, attrname, flags, structname, structmember) class_-
addattr((c),attr_offset_new(attrname,USESYM(float64),(flags),(method)0L,(method)0L,calcoffset(structname,structmembe

Create a 64-bit float attribute and add it to a Max class.

- #define `CLASS_ATTR_SYM`(c, attrname, flags, structname, structmember) class_addattr((c),attr_-
offset_new(attrname,USESYM(symbol),(flags),(method)0L,(method)0L,calcoffset(structname,structmember)))

Create a `t_symbol*` attribute and add it to a Max class.

- `#define CLASS_ATTR_ATOM(c, attrname, flags, structname, structmember) class_addattr((c),attr_offset_new(attrname,USESYM(atom),(flags),(method)0L,(method)0L,calcoffset(structname,structmember)))`

Create a `t_atom` attribute and add it to a Max class.

- `#define CLASS_ATTR_OBJ(c, attrname, flags, structname, structmember) class_addattr((c),attr_offset_new(attrname,USESYM(object),(flags),(method)0L,(method)0L,calcoffset(structname,structmember)))`

Create a `t_object*` attribute and add it to a Max class.

- `#define CLASS_ATTR_CHAR_ARRAY(c, attrname, flags, structname, structmember, size) class_addattr((c),attr_offset_array_new(attrname,USESYM(char),(size),(flags),(method)0L,(method)0L,0/*fix*/,calcoffset(structname,structmember)))`

Create an array-of-chars attribute of fixed length, and add it to a Max class.

- `#define CLASS_ATTR_LONG_ARRAY(c, attrname, flags, structname, structmember, size) class_addattr((c),attr_offset_array_new(attrname,USESYM(long),(size),(flags),(method)0L,(method)0L,0/*fix*/,calcoffset(structname,structmember)))`

Create an array-of-long-integers attribute of fixed length, and add it to a Max class.

- `#define CLASS_ATTR_FLOAT_ARRAY(c, attrname, flags, structname, structmember, size) class_addattr((c),attr_offset_array_new(attrname,USESYM(float32),(size),(flags),(method)0L,(method)0L,0/*fix*/,calcoffset(structname,structmember)))`

Create an array-of-32bit-floats attribute of fixed length, and add it to a Max class.

- `#define CLASS_ATTR_DOUBLE_ARRAY(c, attrname, flags, structname, structmember, size) class_addattr((c),attr_offset_array_new(attrname,USESYM(float64),(size),(flags),(method)0L,(method)0L,0/*fix*/,calcoffset(structname,structmember)))`

Create an array-of-64bit-floats attribute of fixed length, and add it to a Max class.

- `#define CLASS_ATTR_SYM_ARRAY(c, attrname, flags, structname, structmember, size) class_addattr((c),attr_offset_array_new(attrname,USESYM(symbol),(size),(flags),(method)0L,(method)0L,0/*fix*/,calcoffset(structname,structmember)))`

Create an array-of-symbols attribute of fixed length, and add it to a Max class.

- `#define CLASS_ATTR_ATOM_ARRAY(c, attrname, flags, structname, structmember, size) class_addattr((c),attr_offset_array_new(attrname,USESYM(atom),(size),(flags),(method)0L,(method)0L,0/*fix*/,calcoffset(structname,structmember)))`

Create an array-of-atoms attribute of fixed length, and add it to a Max class.

- `#define CLASS_ATTR_OBJ_ARRAY(c, attrname, flags, structname, structmember, size) class_addattr((c),attr_offset_array_new(attrname,USESYM(object),(size),(flags),(method)0L,(method)0L,0/*fix*/,calcoffset(structname,structmember)))`

Create an array-of-objects attribute of fixed length, and add it to a Max class.

- #define **CLASS_ATTR_CHAR_VARSIZE**(c, attrname, flags, structname, structmember, sizemember, maxsize) class_addattr((c),attr_offset_array_-new(attrname,USESYM(char),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizemember),calcoffset(sizemember,flags,structname,structmember,method)))

Create an array-of-chars attribute of variable length, and add it to a Max class.

- #define **CLASS_ATTR_LONG_VARSIZE**(c, attrname, flags, structname, structmember, sizemember, maxsize) class_addattr((c),attr_offset_array_-new(attrname,USESYM(long),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizemember),calcoffset(sizemember,flags,structname,structmember,method)))

Create an array-of-long-integers attribute of variable length, and add it to a Max class.

- #define **CLASS_ATTR_FLOAT_VARSIZE**(c, attrname, flags, structname, structmember, sizemember, maxsize) class_addattr((c),attr_offset_array_-new(attrname,USESYM(float32),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizemember),calcoffset(sizemember,flags,structname,structmember,method)))

Create an array-of-32bit-floats attribute of variable length, and add it to a Max class.

- #define **CLASS_ATTR_DOUBLE_VARSIZE**(c, attrname, flags, structname, structmember, sizemember, maxsize) class_addattr((c),attr_offset_array_-new(attrname,USESYM(float64),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizemember),calcoffset(sizemember,flags,structname,structmember,method)))

Create an array-of-64bit-floats attribute of variable length, and add it to a Max class.

- #define **CLASS_ATTR_SYM_VARSIZE**(c, attrname, flags, structname, structmember, sizemember, maxsize) class_addattr((c),attr_offset_array_-new(attrname,USESYM(symbol),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizemember),calcoffset(sizemember,flags,structname,structmember,method)))

Create an array-of-symbols attribute of variable length, and add it to a Max class.

- #define **CLASS_ATTR_ATOM_VARSIZE**(c, attrname, flags, structname, structmember, sizemember, maxsize) class_addattr((c),attr_offset_array_-new(attrname,USESYM(atom),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizemember),calcoffset(sizemember,flags,structname,structmember,method)))

Create an array-of-atoms attribute of variable length, and add it to a Max class.

- #define **CLASS_ATTR_OBJ_VARSIZE**(c, attrname, flags, structname, structmember, sizemember, maxsize) class_addattr((c),attr_offset_array_-new(attrname,USESYM(object),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizemember),calcoffset(sizemember,flags,structname,structmember,method)))

Create an array-of-objects attribute of variable length, and add it to a Max class.

- #define **STRUCT_ATTR_CHAR**(c, flags, structname, structmember) **CLASS_ATTR_CHAR**(c,#structmember,flags,structname,structmember)

Create a char attribute and add it to a Max class.

- #define **STRUCT_ATTR_LONG**(c, flags, structname, structmember) **CLASS_ATTR_LONG**(c,#structmember,flags,structname,structmember)

Create a long integer attribute and add it to a Max class.

- #define **STRUCT_ATTR_FLOAT**(c, flags, structname, structmember) CLASS_ATTR_FLOAT(c,#structmember,flags,structname,structmember)

Create a 32bit float attribute and add it to a Max class.
- #define **STRUCT_ATTR_DOUBLE**(c, flags, structname, structmember) CLASS_ATTR_DOUBLE(c,#structmember,flags,structname,structmember)

Create a 64bit float attribute and add it to a Max class.
- #define **STRUCT_ATTR_SYM**(c, flags, structname, structmember) CLASS_ATTR_SYM(c,#structmember,flags,structname,structmember)

*Create a *t_symbol** attribute and add it to a Max class.*
- #define **STRUCT_ATTR_ATOM**(c, flags, structname, structmember) CLASS_ATTR_ATOM(c,#structmember,flags,structname,structmember)

*Create a *t_atom* attribute and add it to a Max class.*
- #define **STRUCT_ATTR_OBJ**(c, flags, structname, structmember) CLASS_ATTR_OBJ(c,#structmember,flags,structname,structmember)

*Create a *t_object** attribute and add it to a Max class.*
- #define **STRUCT_ATTR_CHAR_ARRAY**(c, flags, structname, structmember, size) CLASS_ATTR_CHAR_ARRAY(c,#structmember,flags,structname,structmember,size)

Create an array-of-chars attribute of fixed length, and add it to a Max class.
- #define **STRUCT_ATTR_LONG_ARRAY**(c, flags, structname, structmember, size) CLASS_ATTR_LONG_ARRAY(c,#structmember,flags,structname,structmember,size)

Create an array-of-long-integers attribute of fixed length, and add it to a Max class.
- #define **STRUCT_ATTR_FLOAT_ARRAY**(c, flags, structname, structmember, size) CLASS_ATTR_FLOAT_ARRAY(c,#structmember,flags,structname,structmember,size)

Create an array-of-32bit-floats attribute of fixed length, and add it to a Max class.
- #define **STRUCT_ATTR_DOUBLE_ARRAY**(c, flags, structname, structmember, size) CLASS_ATTR_DOUBLE_ARRAY(c,#structmember,flags,structname,structmember,size)

Create an array-of-64bit-floats attribute of fixed length, and add it to a Max class.
- #define **STRUCT_ATTR_SYM_ARRAY**(c, flags, structname, structmember, size) CLASS_ATTR_SYM_ARRAY(c,#structmember,flags,structname,structmember,size)

Create an array-of-symbols attribute of fixed length, and add it to a Max class.
- #define **STRUCT_ATTR_ATOM_ARRAY**(c, flags, structname, structmember, size) CLASS_ATTR_ATOM_ARRAY(c,#structmember,flags,structname,structmember,size)

Create an array-of-atoms attribute of fixed length, and add it to a Max class.
- #define **STRUCT_ATTR_OBJ_ARRAY**(c, flags, structname, structmember, size) CLASS_ATTR_OBJ_ARRAY(c,#structmember,flags,structname,structmember,size)

Create an array-of-objects attribute of fixed length, and add it to a Max class.
- #define **STRUCT_ATTR_CHAR_VARSIZE**(c, flags, structname, structmember, sizemember, maxsize) CLASS_ATTR_CHAR_VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)

Create an array-of-chars attribute of variable length, and add it to a Max class.

Create an array-of-chars attribute of variable length, and add it to a Max class.

- #define **STRUCT_ATTR_LONG_VARSIZE**(c, flags, structname, structmember, sizemember, maxsize) CLASS_ATTR_LONG_- VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)

Create an array-of-long-integers attribute of variable length, and add it to a Max class.

- #define **STRUCT_ATTR_FLOAT_VARSIZE**(c, flags, structname, structmember, sizemember, maxsize) CLASS_ATTR_FLOAT_- VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)

Create an array-of-32bit-floats attribute of variable length, and add it to a Max class.

- #define **STRUCT_ATTR_DOUBLE_VARSIZE**(c, flags, structname, structmember, sizemember, maxsize) CLASS_ATTR_DOUBLE_- VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)

Create an array-of-64bit-floats attribute of variable length, and add it to a Max class.

- #define **STRUCT_ATTR_SYM_VARSIZE**(c, flags, structname, structmember, sizemember, maxsize) CLASS_ATTR_SYM_- VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)

Create an array-of-symbols attribute of variable length, and add it to a Max class.

- #define **STRUCT_ATTR_ATOM_VARSIZE**(c, flags, structname, structmember, sizemember, maxsize) CLASS_ATTR_ATOM_- VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)

Create an array-of-atoms attribute of variable length, and add it to a Max class.

- #define **STRUCT_ATTR_OBJ_VARSIZE**(c, flags, structname, structmember, sizemember, maxsize) CLASS_ATTR_OBJ_VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)

Create an array-of-objects attribute of variable length, and add it to a Max class.

- #define **STATIC_ATTR_CHAR**(c, attrname, flags, val) STATIC_ATTR_- FORMAT(c,attrname,USESYM(char),flags,"c",val)

Create a shared (static/global) char attribute and add it to a Max class.

- #define **STATIC_ATTR_LONG**(c, attrname, flags, val) STATIC_ATTR_- FORMAT(c,attrname,USESYM(long),flags,"l",val)

Create a shared (static/global) long integer attribute and add it to a Max class.

- #define **STATIC_ATTR_FLOAT**(c, attrname, flags, val) STATIC_ATTR_- FORMAT(c,attrname,USESYM(float32),flags,"f",val)

Create a shared (static/global) 32bit float attribute and add it to a Max class.

- #define **STATIC_ATTR_DOUBLE**(c, attrname, flags, val) STATIC_ATTR_- FORMAT(c,attrname,USESYM(float64),flags,"d",val)

Create a shared (static/global) 64bit float attribute and add it to a Max class.

- #define **STATIC_ATTR_SYM**(c, attrname, flags, val) STATIC_ATTR_- FORMAT(c,attrname,USESYM(symbol),flags,"s",val)

*Create a shared (static/global) *t_symbol** attribute and add it to a Max class.*

- #define **STATIC_ATTR_ATOM**(c, attrname, flags, val) STATIC_ATTR_FORMAT(c,attrname,USESYM(atom),flags,"a",val)

Create a shared (static/global) `t_atom` attribute and add it to a Max class.
- #define **STATIC_ATTR_OBJ**(c, attrname, flags, val) STATIC_ATTR_FORMAT(c,attrname,USESYM(object),flags,"o",val)

Create a shared (static/global) `t_object` attribute and add it to a Max class.*
- #define **STATIC_ATTR_CHAR_ARRAY**(c, attrname, flags, count, vals) STATIC_ATTR_FORMAT(c,attrname,USESYM(char),flags,"C",count,vals)

Create a shared (static/global) array-of-chars attribute of fixed length, and add it to a Max class.
- #define **STATIC_ATTR_LONG_ARRAY**(c, attrname, flags, count, vals) STATIC_ATTR_FORMAT(c,attrname,USESYM(long),flags,"L",count,vals)

Create a shared (static/global) array-of-long-integers attribute of fixed length, and add it to a Max class.
- #define **STATIC_ATTR_FLOAT_ARRAY**(c, attrname, flags, count, vals) STATIC_ATTR_FORMAT(c,attrname,USESYM(float32),flags,"F",count,vals)

Create a shared (static/global) array-of-32bit-floats attribute of fixed length, and add it to a Max class.
- #define **STATIC_ATTR_DOUBLE_ARRAY**(c, attrname, flags, count, vals) STATIC_ATTR_FORMAT(c,attrname,USESYM(float64),flags,"D",count,vals)

Create a shared (static/global) array-of-64bit-floats attribute of fixed length, and add it to a Max class.
- #define **STATIC_ATTR_SYM_ARRAY**(c, attrname, flags, count, vals) STATIC_ATTR_FORMAT(c,attrname,USESYM(symbol),flags,"S",count,vals)

Create a shared (static/global) array-of-symbols attribute of fixed length, and add it to a Max class.
- #define **STATIC_ATTR_ATOM_ARRAY** STATIC_ATTR_ATOMS

Create a shared (static/global) array-of-atoms attribute of fixed length, and add it to a Max class.
- #define **STATIC_ATTR_OBJ_ARRAY**(c, attrname, flags, count, vals) STATIC_ATTR_FORMAT(c,attrname,USESYM(object),flags,"O",count,vals)

Create a shared (static/global) array-of-objects attribute of fixed length, and add it to a Max class.
- #define **OBJ_ATTR_CHAR**(x, attrname, flags, val) OBJ_ATTR_FORMAT(x,attrname,USESYM(char),flags,"c",val)

Create an instance-local char attribute and add it to a Max class.
- #define **OBJ_ATTR_LONG**(x, attrname, flags, val) OBJ_ATTR_FORMAT(x,attrname,USESYM(long),flags,"l",val)

Create an instance-local long integer attribute and add it to a Max class.
- #define **OBJ_ATTR_FLOAT**(x, attrname, flags, val) OBJ_ATTR_FORMAT(x,attrname,USESYM(float32),flags,"f",val)

Create an instance-local 32bit float attribute and add it to a Max class.
- #define **OBJ_ATTR_DOUBLE**(x, attrname, flags, val) OBJ_ATTR_FORMAT(x,attrname,USESYM(float64),flags,"d",val)

Create an instance-local 64bit float attribute and add it to a Max class.

- #define **OBJ_ATTR_SYM**(x, attrname, flags, val) OBJ_ATTR_-
FORMAT(x,attrname,USESYM(symbol),flags,"s",val)
Create an instance-local `t_symbol` attribute and add it to a Max class.*
- #define **OBJ_ATTR_ATOM**(x, attrname, flags, val) OBJ_ATTR_-
FORMAT(x,attrname,USESYM(atom),flags,"a",val)
Create an instance-local `t_atom` attribute and add it to a Max class.
- #define **OBJ_ATTR_OBJ**(x, attrname, flags, val) OBJ_ATTR_-
FORMAT(x,attrname,USESYM(object),flags,"o",val)
Create an instance-local `t_object` attribute and add it to a Max class.*
- #define **OBJ_ATTR_CHAR_ARRAY**(x, attrname, flags, count, vals) OBJ_ATTR_-
FORMAT(x,attrname,USESYM(char),flags,"C",count,vals)
Create an instance-local array-of-chars attribute of fixed length, and add it to the object.
- #define **OBJ_ATTR_LONG_ARRAY**(x, attrname, flags, count, vals) OBJ_ATTR_-
FORMAT(x,attrname,USESYM(long),flags,"L",count,vals)
Create an instance-local array-of-long-integers attribute of fixed length, and add it to the object.
- #define **OBJ_ATTR_FLOAT_ARRAY**(x, attrname, flags, count, vals) OBJ_ATTR_-
FORMAT(x,attrname,USESYM(float32),flags,"F",count,vals)
Create an instance-local array-of-32bit-floats attribute of fixed length, and add it to the object.
- #define **OBJ_ATTR_DOUBLE_ARRAY**(x, attrname, flags, count, vals) OBJ_ATTR_-
FORMAT(x,attrname,USESYM(float64),flags,"D",count,vals)
Create an instance-local array-of-64bit-floats attribute of fixed length, and add it to the object.
- #define **OBJ_ATTR_SYM_ARRAY**(x, attrname, flags, count, vals) OBJ_ATTR_-
FORMAT(x,attrname,USESYM(symbol),flags,"S",count,vals)
Create an instance-local array-of-symbols attribute of fixed length, and add it to the object.
- #define **OBJ_ATTR_ATOM_ARRAY** OBJ_ATTR_ATOMS
Create an instance-local array-of-atoms attribute of fixed length, and add it to the object.
- #define **OBJ_ATTR_OBJ_ARRAY**(x, attrname, flags, count, vals) OBJ_ATTR_-
FORMAT(x,attrname,USESYM(object),flags,"O",count,vals)
Create an instance-local array-of-objects attribute of fixed length, and add it to the object.
- #define **CLASS_ATTR_ACCESSORS**(c, attrname, getter, setter)
Specify custom accessor methods for an attribute.
- #define **CLASS_ATTR_ADD_FLAGS**(c, attrname, flags)
Add flags to an attribute.
- #define **CLASS_ATTR_REMOVE_FLAGS**(c, attrname, flags)
Remove flags from an attribute.
- #define **CLASS_ATTR_FILTER_MIN**(c, attrname, minval)

Add a filter to the attribute to limit the lower bound of a value.

- `#define CLASS_ATTR_FILTER_MAX(c, attrname, maxval)`

Add a filter to the attribute to limit the upper bound of a value.

- `#define CLASS_ATTR_FILTER_CLIP(c, attrname, minval, maxval)`

Add a filter to the attribute to limit both the lower and upper bounds of a value.

- `#define CLASS_ATTR_ALIAS(c, attrname, aliasname)`

Create a new attribute that is an alias of an existing attribute.

- `#define CLASS_ATTR_DEFAULT(c, attrname, flags, parsestr) { t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname)); CLASS_ATTR_ATTR_PARSE(c,attrname,"default",(t_symbol *)object_method(theattr,USESYM(gettype)),flags,parsestr); }`

Add a new attribute to the specified attribute to specify a default value.

- `#define CLASS_ATTR_SAVE(c, attrname, flags) CLASS_ATTR_ATTRPARSE(c,attrname,"save",USESYM(long),flags,"1")`

Add a new attribute to the specified attribute to indicate that the specified attribute should be saved with the patcher.

- `#define CLASS_ATTR_DEFAULT_SAVE(c, attrname, flags, parsestr) { CLASS_ATTR_DEFAULT(c,attrname,flags,parsestr); CLASS_ATTR_SAVE(c,attrname,flags); }`

A convenience wrapper for both `CLASS_ATTR_DEFAULT` and `CLASS_ATTR_SAVE`.

- `#define CLASS_ATTR_DEFAULTNAME(c, attrname, flags, parsestr) { t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname)); CLASS_ATTR_ATTRPARSE(c,attrname,"defaultname",(t_symbol *)object_method(theattr,USESYM(gettype)),flags,parsestr); }`

Add a new attribute to the specified attribute to specify a default value, based on Max's Object Defaults.

- `#define CLASS_ATTR_DEFAULTNAME_SAVE(c, attrname, flags, parsestr) { CLASS_ATTR_DEFAULTNAME(c,attrname,flags,parsestr); CLASS_ATTR_SAVE(c,attrname,flags); }`

A convenience wrapper for both `CLASS_ATTR_DEFAULTNAME` and `CLASS_ATTR_SAVE`.

- `#define CLASS_ATTR_MIN(c, attrname, flags, parsestr) { t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname)); CLASS_ATTR_ATTRPARSE(c,attrname,"min",(t_symbol *)object_method(theattr,USESYM(gettype)),flags,parsestr); }`

Add a new attribute to the specified attribute to specify a lower range.

- `#define CLASS_ATTR_MAX(c, attrname, flags, parsestr) { t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname)); CLASS_ATTR_ATTRPARSE(c,attrname,"max",(t_symbol *)object_method(theattr,USESYM(gettype)),flags,parsestr); }`

Add a new attribute to the specified attribute to specify an upper range.

- `#define CLASS_ATTR_PAINT(c, attrname, flags) CLASS_ATTR_ATTRPARSE(c,attrname,"paint",USESYM(long),flags,"1")`

Add a new attribute indicating that any changes to the specified attribute will trigger a call to the object's paint method.

- `#define CLASS_ATTR_DEFAULT_PAINT(c, attrname, flags, parsestr) { CLASS_ATTR_DEFAULT(c,attrname,flags,parsestr); CLASS_ATTR_PAINT(c,attrname,flags); }`

A convenience wrapper for both `CLASS_ATTR_DEFAULT` and `CLASS_ATTR_PAINT`.

- `#define CLASS_ATTR_DEFAULT_SAVE_PAINT(c, attrname, flags, parsestr) { CLASS_ATTR_DEFAULT(c,attrname,flags,parsestr); CLASS_ATTR_SAVE(c,attrname,flags); CLASS_ATTR_PAINT(c,attrname,flags); }`

A convenience wrapper for `CLASS_ATTR_DEFAULT`, `CLASS_ATTR_SAVE`, and `CLASS_ATTR_PAINT`.

- `#define CLASS_ATTR_DEFAULTNAME_PAINT(c, attrname, flags, parsestr) { CLASS_ATTR_DEFAULTNAME(c,attrname,flags,parsestr); CLASS_ATTR_PAINT(c,attrname,flags); }`

A convenience wrapper for `CLASS_ATTR_DEFAULTNAME`, `CLASS_ATTR_SAVE`, and `CLASS_ATTR_PAINT`.

- `#define CLASS_ATTR_DEFAULTNAME_SAVE_PAINT(c, attrname, flags, parsestr) { CLASS_ATTR_DEFAULTNAME(c,attrname,flags,parsestr); CLASS_ATTR_SAVE(c,attrname,flags); CLASS_ATTR_PAINT(c,attrname,flags); }`

A convenience wrapper for `CLASS_ATTR_DEFAULTNAME`, `CLASS_ATTR_SAVE`, and `CLASS_ATTR_PAINT`.

- `#define CLASS_ATTR_STYLE(c, attrname, flags, parsestr) CLASS_ATTR_ATTRIBUTEPARSE(c,attrname,"style",USESYM(symbol),flags,parsestr)`

Add a new attribute to the specified attribute to specify an editor style for the Max inspector.

- `#define CLASS_ATTR_LABEL(c, attrname, flags, labelstr) CLASS_ATTR_ATTRIBUTEFORMAT(c,attrname,"label",USESYM(symbol),flags,"s",gensym(labelstr))`

Add a new attribute to the specified attribute to specify an a human-friendly label for the Max inspector.

- `#define CLASS_ATTR_ENUM(c, attrname, flags, parsestr) { CLASS_ATTR_STYLE(c,attrname,flags,"enum"); CLASS_ATTR_ATTRIBUTEPARSE(c,attrname,"enumvals",USESYM(atom),flags,parsestr); }`

Add a new attribute to the specified attribute to specify a list of choices to display in a menu for the Max inspector.

- `#define CLASS_ATTR_ENUMINDEX(c, attrname, flags, parsestr) { CLASS_ATTR_STYLE(c,attrname,flags,"enumindex"); CLASS_ATTR_ATTRIBUTEPARSE(c,attrname,"enumvals",USESYM(atom),flags,parsestr); }`

Add a new attribute to the specified attribute to specify a list of choices to display in a menu for the Max inspector.

- `#define CLASS_ATTR_CATEGORY(c, attrname, flags, parsestr) CLASS_ATTR_ATTRIBUTEPARSE(c,attrname,"category",USESYM(symbol),flags,parsestr)`

Add a new attribute to the specified attribute to specify a category to which the attribute is assigned in the Max inspector.

- `#define CLASS_ATTR_STYLE_LABEL(c, attrname, flags, stylestr, labelstr) { CLASS_ATTR_ATTRIBUTEPARSE(c,attrname,"style",USESYM(symbol),flags,stylestr); CLASS_ATTR_ATTRIBUTEFORMAT(c,attrname,"label",USESYM(symbol),flags,"s",gensym(labelstr)); }`

A convenience wrapper for `CLASS_ATTR_STYLE`, and `CLASS_ATTR_LABEL`.

- `#define CLASS_ATTR_INVISIBLE(c, attrname, flags) CLASS_ATTR_ATTRIBUTEPARSE(c,attrname,"invisible",USESYM(long),flags,"1")`

Add a new attribute to the specified attribute to flag an attribute as invisible to the Max inspector.

- #define **CLASS_ATTR_ORDER**(c, attrname, flags, parsestr) CLASS_ATTR_ATTRPARSE(c,attrname,"order",USESYM(long),flags,parsestr)

Add a new attribute to the specified attribute to specify a default order in which to list attributes.
- #define **CLASS_METHOD_ATTR_PARSE**(c, methodname, attrname, type, flags, parsestring)

Define and add attributes to class methods.
- #define **OBJ_ATTR_DEFAULT**(x, attrname, flags, parsestr) { t_object *theattr=(t_object *)object_attr_get(x,gensym(attrname)); OBJ_ATTR_ATTRPARSE(x,attrname,"default",(t_symbol *)object_method(theattr,USESYM(gettype)),flags,parsestr); }

*An instance-attribute version of **CLASS_ATTR_DEFAULT**.*
- #define **OBJ_ATTR_SAVE**(x, attrname, flags) OBJ_ATTR_ATTRPARSE(x,attrname,"save",USESYM(long),flags,"1")

*An instance-attribute version of **CLASS_ATTR_SAVE**.*
- #define **OBJ_ATTR_DEFAULT_SAVE**(x, attrname, flags, parsestr) { OBJ_ATTR_DEFAULT(x,attrname,flags,parsestr); OBJ_ATTR_SAVE(x,attrname,flags); }

*An instance-attribute version of **CLASS_ATTR_DEFAULT_SAVE**.*
- #define **CLASS_STICKY_ATTR**(c, name, flags, parsestr) { t_object *attr = attribute_new_parse(name,NULL,flags,parsestr); class_sticky(c,gensym("sticky_attr"),gensym(name),attr); }

Create an attribute, and add it to all following attribute declarations.
- #define **CLASS_STICKY_ATTR_CLEAR**(c, name) class_sticky_clear(c,gensym("sticky_attr"),name?gensym(name):NULL)

*Close a **CLASS_STICKY_ATTR** block.*
- #define **CLASS_STICKY_METHOD**(c, name, flags, parsestr) { t_object *attr = attribute_new_parse(name,NULL,flags,parsestr); class_sticky(c,gensym("sticky_method"),gensym(name),attr); }

Create an attribute, and add it to all following method declarations.
- #define **CLASS_STICKY_METHOD_CLEAR**(c, name) class_sticky_clear(c,gensym("sticky_method"),name?gensym(name):clear)

*Close a **CLASS_STICKY_METHOD** block.*
- #define **CLASS_ATTR_RGBA**(c, attrname, flags, structname, structmember)

*Create a color (*t_jrgba*) attribute and add it to a Max class.*

Enumerations

- enum **e_max_attrflags** {

ATTR_FLAGS_NONE = 0x0000000,

ATTR_GET_OPAQUE = 0x00000001,

ATTR_SET_OPAQUE = 0x00000002,

ATTR_GET_OPAQUE_USER = 0x00000100,

ATTR_SET_OPAQUE_USER = 0x00000200
 }

Attribute flags.

Functions

- `void * object_attr_get (void *x, t_symbol *attrname)`
Returns the pointer to an attribute, given its name.
- `method object_attr_method (void *x, t_symbol *methodname, void **attr, long *get)`
Returns the method of an attribute's get or set function, as well as a pointer to the attribute itself, from a message name.
- `long object_attr_usercanset (void *x, t_symbol *s)`
Determines if an object's attribute can be set from the Max interface (i.e.
- `long object_attr_usercanget (void *x, t_symbol *s)`
Determines if the value of an object's attribute can be queried from the Max interface (i.e.
- `void object_attr_getdump (void *x, t_symbol *s, long argc, t_atom *argv)`
Forces a specified object's attribute to send its value from the object's dumpout outlet in the Max interface.
- `t_max_err object_attr_setvalueof (void *x, t_symbol *s, long argc, t_atom *argv)`
Sets the value of an object's attribute.
- `t_max_err object_addattr (void *x, t_object *attr)`
Attaches an attribute directly to an object.
- `t_max_err object_deleteattr (void *x, t_symbol *attrsym)`
Detach an attribute from an object that was previously attached with `object_addattr()`.
- `t_max_err object_chuckattr (void *x, t_symbol *attrsym)`
Detach an attribute from an object that was previously attached with `object_addattr()`.
- `long attr_args_offset (short ac, t_atom *av)`
Determines the point in an atom list where attribute arguments begin.
- `void attr_args_process (void *x, short ac, t_atom *av)`
Takes an atom list and properly set any attributes described within.
- `t_object * attribute_new (C74_CONST char *name, t_symbol *type, long flags, method mget, method mset)`
Create a new attribute.
- `t_object * attr_offset_new (C74_CONST char *name, C74_CONST t_symbol *type, long flags, C74_CONST method mget, C74_CONST method mset, long offset)`
Create a new attribute.
- `t_object * attr_offset_array_new (C74_CONST char *name, t_symbol *type, long size, long flags, method mget, method mset, long offsetcount, long offset)`
Create a new attribute.
- `long object_attr_getlong (void *x, t_symbol *s)`
Retrieves the value of an attribute, given its parent object and name.

- **t_max_err object_attr_setlong** (void *x, **t_symbol** *s, long c)
Sets the value of an attribute, given its parent object and name.
- float **object_attr_getfloat** (void *x, **t_symbol** *s)
Retrieves the value of an attribute, given its parent object and name.
- **t_max_err object_attr_setfloat** (void *x, **t_symbol** *s, float c)
Sets the value of an attribute, given its parent object and name.
- **t_symbol * object_attr_getsym** (void *x, **t_symbol** *s)
Retrieves the value of an attribute, given its parent object and name.
- **t_max_err object_attr_setsym** (void *x, **t_symbol** *s, **t_symbol** *c)
Sets the value of an attribute, given its parent object and name.
- long **object_attr_getlong_array** (void *x, **t_symbol** *s, long max, long *vals)
Retrieves the value of an attribute, given its parent object and name.
- **t_max_err object_attr_setlong_array** (void *x, **t_symbol** *s, long count, long *vals)
Sets the value of an attribute, given its parent object and name.
- long **object_attr_getchar_array** (void *x, **t_symbol** *s, long max, uchar *vals)
Retrieves the value of an attribute, given its parent object and name.
- **t_max_err object_attr_setchar_array** (void *x, **t_symbol** *s, long count, C74_CONST uchar *vals)
Sets the value of an attribute, given its parent object and name.
- long **object_attr_getfloat_array** (void *x, **t_symbol** *s, long max, float *vals)
Retrieves the value of an attribute, given its parent object and name.
- **t_max_err object_attr_setfloat_array** (void *x, **t_symbol** *s, long count, float *vals)
Sets the value of an attribute, given its parent object and name.
- long **object_attr_getdouble_array** (void *x, **t_symbol** *s, long max, double *vals)
Retrieves the value of an attribute, given its parent object and name.
- **t_max_err object_attr_setdouble_array** (void *x, **t_symbol** *s, long count, double *vals)
Sets the value of an attribute, given its parent object and name.
- long **object_attr_getsym_array** (void *x, **t_symbol** *s, long max, **t_symbol** **vals)
Retrieves the value of an attribute, given its parent object and name.
- **t_max_err object_attr_setsym_array** (void *x, **t_symbol** *s, long count, **t_symbol** **vals)
Sets the value of an attribute, given its parent object and name.
- **t_max_err attr_addfilterset_clip** (void *x, double min, double max, long usemin, long usemax)
Attaches a clip filter to an attribute.

- `t_max_err attr_addfilterset_clip_scale` (void *x, double scale, double min, double max, long usemin, long usemax)
Attaches a clip/scale filter to an attribute.
- `t_max_err attr_addfilterget_clip` (void *x, double min, double max, long usemin, long usemax)
Attaches a clip filter to an attribute.
- `t_max_err attr_addfilterget_clip_scale` (void *x, double scale, double min, double max, long usemin, long usemax)
Attaches a clip/scale filter to an attribute.
- `t_max_err attr_addfilter_clip` (void *x, double min, double max, long usemin, long usemax)
Attaches a clip filter to an attribute.
- `t_max_err attr_addfilter_clip_scale` (void *x, double scale, double min, double max, long usemin, long usemax)
Attaches a clip/scale filter to an attribute.
- `t_max_err attr_addfilterset_proc` (void *x, `method` proc)
Attaches a custom filter method to an attribute.
- `t_max_err attr_addfilterget_proc` (void *x, `method` proc)
Attaches a custom filter method to an attribute.
- `void attr_args_dictionary (t_dictionary *x, short ac, t_atom *av)`
Create a dictionary of attribute-name, attribute-value pairs from an array of atoms containing an attribute definition list.
- `void attr_dictionary_process (void *x, t_dictionary *d)`
Set attributes for an object that are defined in a dictionary.
- `t_max_err object_attr_setparse (t_object *x, t_symbol *s, C74_CONST char *parsestr)`
Set an attribute value with one or more atoms parsed from a C-string.
- `void * object_new_parse (t_symbol *name_space, t_symbol *classname, C74_CONST char *parsestr)`
Create a new object with one or more atoms parsed from a C-string.
- `t_max_err object_attr_getjrgba (void *ob, t_symbol *s, t_jrgba *c)`
Retrieves the value of a color attribute, given its parent object and name.
- `t_max_err object_attr_setjrgba (void *ob, t_symbol *s, t_jrgba *c)`
Sets the value of a color attribute, given its parent object and name.
- `t_max_err object_attr_get_rect (t_object *o, t_symbol *name, t_rect *rect)`
Gets the value of a `t_rect` attribute, given its parent object and name.
- `t_max_err object_attr_set_rect (t_object *o, t_symbol *name, t_rect *rect)`
Sets the value of a `t_rect` attribute, given its parent object and name.

- [t_max_err object_attr_getpt \(t_object *o, t_symbol *name, t_pt *pt\)](#)
Gets the value of a [t_pt](#) attribute, given its parent object and name.
- [t_max_err object_attr_setpt \(t_object *o, t_symbol *name, t_pt *pt\)](#)
Sets the value of a [t_pt](#) attribute, given its parent object and name.
- [t_max_err object_attr_getsize \(t_object *o, t_symbol *name, t_size *size\)](#)
Gets the value of a [t_size](#) attribute, given its parent object and name.
- [t_max_err object_attr_setsize \(t_object *o, t_symbol *name, t_size *size\)](#)
Sets the value of a [t_size](#) attribute, given its parent object and name.
- [t_max_err object_attr_getcolor \(t_object *b, t_symbol *attrname, t_jrgba *prgba\)](#)
Gets the value of a [t_jrgba](#) attribute, given its parent object and name.
- [t_max_err object_attr_setcolor \(t_object *b, t_symbol *attrname, t_jrgba *prgba\)](#)
Sets the value of a [t_jrgba](#) attribute, given its parent object and name.

33.1.1 Detailed Description

An attribute of an object is a setting or property that tells the object how to do its job. For example, the metro object has an interval attribute that tells it how fast to run.

Attributes are similar to methods, except that the attributes have a state. Attributes are themselves objects, and they share a common interface for getting and setting values.

An attribute is most typically added to the class definition of another object during it's class initialization or main() function. Most typically, this attribute's value will be stored in an instance's struct, and thus it will serve as a property of that instance of the object.

Attributes can, however, be declared as 'class static'. This means that the property is shared by all instances of the class, and the value is stored as a shared (static) variable.

Additionally, Max 5 has introduced the notion of 'instance attributes' (also called 'object attributes'). Instance attributes are the creation of an attribute object, and then adding it to one specific instance of another class.

Finally, because attributes themselves are Max objects they too can possess attributes. These 'attributes of attributes' are used in Max to do things like specify a range of values for an attribute, give an attribute human friendly caption, or determine to what category an attribute should belong in the inspector.

The easiest and most common way of working with attributes is to use the provided macros. These macros simplify the process of creating a new attribute object, setting any attributes of the attribute, and binding it to an object class or an object instance.

33.1.2 Setting and Getting Attribute Values

By default, Max provides standard attribute accessors. These are the functions the get or set the attribute value in the object's struct. If you need to define a custom accessor, you can specify this information using the [CLASS_ATTR_ACCESSORS](#) macro.

33.1.2.1 Writing a custom Attribute Getter

If you need to define a custom accessor, it should have a prototype and form comparable to the following custom getter:

```
t_max_err foo_myval_get(t_foo *x, void *attr, long *ac, t_atom **av)
{
    if ((*ac)&&(*av)) {
        //memory passed in, use it
    } else {
        //otherwise allocate memory
        *ac = 1;
        if (!(*av = getbytes(sizeof(t_atom) * (*ac)))) {
            *ac = 0;
            return MAX_ERR_OUT_OF_MEM;
        }
    }
    atom_setfloat(*av,x->myval);

    return MAX_ERR_NONE;
}
```

Note that getters require memory to be allocated, if there is not memory passed into the getter. Also the attr argument is the class' attribute object and can be queried using object_method for things like the attribute flags, names, filters, etc..

33.1.2.2 Writing a custom Attribute Getter

If you need to define a custom accessor, it should have a prototype and form comparable to the following custom setter:

```
t_max_err foo_myval_set(t_foo *x, void *attr, long ac, t_atom *av)
{
    if (ac&&av) {
        x->myval = atom_getfloat(av);
    } else {
        // no args, set to zero
        x->myval = 0;
    }
    return MAX_ERR_NONE;
}
```

33.1.3 Attribute Notificaton

Although the subject of object registration and notification is covered elsewhere, it bears noting that attributes of all types will, if registered, automatically send notifications to all attached client objects each time the attribute's value is set.

33.1.4 Define Documentation

33.1.4.1 #define CLASS_ATTR_ACCESSORS(c, attrname, getter, setter)

Value:

```
{ t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname)); \
    object_method(theattr,gensym("setmethod"),USESYM(get),getter); \
    object_method(theattr,gensym("setmethod"),USESYM(set),setter); }
```

Specify custom accessor methods for an attribute.

If you specify a non-NULL value for the setter or getter, then the function you specify will be called to set or get the attribute's value rather than using the built-in accessor.

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

getter An appropriate getter method as discussed in [Setting and Getting Attribute Values](#), or NULL to use the default getter.

setter An appropriate setter method as discussed in [Setting and Getting Attribute Values](#), or NULL to use the default setter.

33.1.4.2 #define CLASS_ATTR_ADD_FLAGS(*c*, *attrname*, *flags*)

Value:

```
{ t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname)); \
    long oldflags = object_method(theattr,gensym("getflags")); \
    object_method(theattr,gensym("setflags"),oldflags|(flags)); }
```

Add flags to an attribute.

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to add to this attribute, as defined in [e_max_attrflags](#).

33.1.4.3 #define CLASS_ATTR_ALIAS(*c*, *attrname*, *aliasname*)

Value:

```
{ t_object *thealias; \
    t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname)); \
    thealias = object_clone(theattr); \
    object_method(thealias,USESYM(setname),gensym(aliasname)); \
    class_addattr(c,thearias); \
    CLASS_ATTR_ATTR_PARSE(c,aliasname,"alias",USESYM(symbol),0,attrname); }
```

Create a new attribute that is an alias of an existing attribute.

Parameters

c The class pointer.

attrname The name of the actual attribute as a C-string.

aliasname The name of the new alias attribute.

```
33.1.4.4 #define CLASS_ATTR_ATOM(c, attrname, flags,
structname, structmember) class_addattr((c),attr_offset_-
new(attrname,USESYM(atom),(flags),(method)0L,(method)0L,calcoffset(structname,structmember)))
```

Create a `t_atom` attribute and add it to a Max class.

Parameters

- c* The class pointer.
- attrname* The name of this attribute as a C-string.
- flags* Any flags you wish to declare for this attribute, as defined in `e_max_attrflags`.
- structname* The C identifier for the struct (containing a valid `t_object` header) representing an instance of this class.
- structmember* The C identifier of the member in the struct that holds the value of this attribute.

```
33.1.4.5 #define CLASS_ATTR_ATOM_ARRAY(c, attrname, flags,
structname, structmember, size) class_addattr((c),attr_offset_array_-
new(attrname,USESYM(atom),(size),(flags),(method)0L,(method)0L,/*fix*/,calcoffset(structname,structmem
```

Create an array-of-atoms attribute of fixed length, and add it to a Max class.

Parameters

- c* The class pointer.
- attrname* The name of this attribute as a C-string.
- flags* Any flags you wish to declare for this attribute, as defined in `e_max_attrflags`.
- structname* The C identifier for the struct (containing a valid `t_object` header) representing an instance of this class.
- structmember* The C identifier of the member in the struct that holds the value of this attribute.
- size* The number of items in the `t_atom` array.

```
33.1.4.6 #define CLASS_ATTR_ATOM_VARSIZE(c, attrname, flags, structname,
structmember, sizemember, maxsize) class_addattr((c),attr_offset_array_-
new(attrname,USESYM(atom),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizemember),
```

Create an array-of-atoms attribute of variable length, and add it to a Max class.

Parameters

- c* The class pointer.
- attrname* The name of this attribute as a C-string.
- flags* Any flags you wish to declare for this attribute, as defined in `e_max_attrflags`.
- structname* The C identifier for the struct (containing a valid `t_object` header) representing an instance of this class.
- structmember* The C identifier of the member in the struct that holds the value of this attribute.
- sizemember* The actual number of items in the `t_atom` array at any given moment.
- maxsize* The maximum number of items in the `t_atom` array, i.e. the number of members allocated for the array in the struct.

**33.1.4.7 #define CLASS_ATTR_CATEGORY(c, attrname, flags, parsestr) CLASS_ATTR_-
ATTR_PARSE(c,attrname,"category",USESYM(symbol),flags,parsestr)**

Add a new attribute to the specified attribute to specify a category to which the attribute is assigned in the Max inspector.

Categories are represented in the inspector as tabs. If the specified category does not exist then it will be created.

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

**33.1.4.8 #define CLASS_ATTR_CHAR(c, attrname, flags,
structname, structmember) class_addattr((c),attr_offset_-
new(attrname,USESYM(char),(flags),(method)0L,(method)0L,calcoffset(structname,structmember)))**

Create a char attribute and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

**33.1.4.9 #define CLASS_ATTR_CHAR_ARRAY(c, attrname, flags,
structname, structmember, size) class_addattr((c),attr_offset_array_-
new(attrname,USESYM(char),(size),(flags),(method)0L,(method)0L,0/*fix*/,calcoffset(structname,structmem**

Create an array-of-chars attribute of fixed length, and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

size The number of chars in the array.

```
33.1.4.10 #define CLASS_ATTR_CHAR_VARSIZE(c, attrname, flags, structname,
structmember, sizemember, maxsize) class_addattr((c),attr_offset_array_-
new(attrname,USESYM(char),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizemember))
```

Create an array-of-chars attribute of variable length, and add it to a Max class.

Parameters

c The class pointer.
attrname The name of this attribute as a C-string.
flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).
structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.
structmember The C identifier of the member in the struct that holds the value of this attribute.
sizemember The actual number of items in the char array at any given moment.
maxsize The maximum number of items in the char array, i.e. the number of members allocated for the array in the struct.

```
33.1.4.11 #define CLASS_ATTR_DEFAULT(c, attrname, flags, parsestr) {
    t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname));
    CLASS_ATTR_ATTR_PARSE(c,attrname,"default",(t_symbol
    *)object_method(theattr,USESYM(gettype)),flags,parsestr); }
```

Add a new attribute to the specified attribute to specify a default value.

The default value will be automatically set when the object is created only if your object uses a dictionary constructor with the [CLASS_FLAG_NEWDICTIONARY](#) flag.

Parameters

c The class pointer.
attrname The name of the attribute as a C-string.
flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).
parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

```
33.1.4.12 #define CLASS_ATTR_DEFAULT_PAINT(c, attrname, flags, parsestr) { CLASS_-
ATTR_DEFAULT(c,attrname,flags,parsestr); CLASS_ATTR_PAINT(c,attrname,flags);
}
```

A convenience wrapper for both [CLASS_ATTR_DEFAULT](#) and [CLASS_ATTR_PAINT](#).

Parameters

c The class pointer.
attrname The name of the attribute as a C-string.
flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).
parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

See also

[CLASS_ATTR_DEFAULT](#)
[CLASS_ATTR_PAINT](#)

```
33.1.4.13 #define CLASS_ATTR_DEFAULT_SAVE(c, attrname, flags, parsestr) { CLASS_-
ATTR_DEFAULT(c,attrname,flags,parsestr); CLASS_ATTR_SAVE(c,attrname,flags);
}
```

A convenience wrapper for both [CLASS_ATTR_DEFAULT](#) and [CLASS_ATTR_SAVE](#).

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

See also

[CLASS_ATTR_DEFAULT](#)
[CLASS_ATTR_SAVE](#)

```
33.1.4.14 #define CLASS_ATTR_DEFAULT_SAVE_PAINT(c, attrname, flags,
parsestr) { CLASS_ATTR_DEFAULT(c,attrname,flags,parsestr);
CLASS_ATTR_SAVE(c,attrname,flags); CLASS_ATTR_PAINT(c,attrname,flags); }
```

A convenience wrapper for [CLASS_ATTR_DEFAULT](#), [CLASS_ATTR_SAVE](#), and [CLASS_ATTR_PAINT](#).

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

See also

[CLASS_ATTR_DEFAULT](#)
[CLASS_ATTR_PAINT](#)
[CLASS_ATTR_SAVE](#)

```
33.1.4.15 #define CLASS_ATTR_DEFAULTNAME(c, attrname, flags, parsestr) {
t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname));
CLASS_ATTR_ATTR_PARSE(c,attrname,"defaultname",(t_symbol
*)object_method(theattr,USESYM(gettype)),flags,parsestr); }
```

Add a new attribute to the specified attribute to specify a default value, based on Max's Object Defaults.

If a value is present in Max's Object Defaults, then that value will be used as the default value. Otherwise, use the default value specified here. The default value will be automatically set when the object is created only if your object uses a dictionary constructor with the [CLASS_FLAG_NEWDICTIONARY](#) flag.

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

```
33.1.4.16 #define CLASS_ATTR_DEFAULTNAME_PAINT(c, attrname, flags,  
parsestr) { CLASS_ATTR_DEFAULTNAME(c,attrname,flags,parsestr);  
CLASS_ATTR_PAINT(c,attrname,flags); }
```

A convenience wrapper for [CLASS_ATTR_DEFAULTNAME](#), [CLASS_ATTR_SAVE](#), and [CLASS_ATTR_PAINT](#).

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

See also

[CLASS_ATTR_DEFAULTNAME](#)

[CLASS_ATTR_PAINT](#)

[CLASS_ATTR_SAVE](#)

```
33.1.4.17 #define CLASS_ATTR_DEFAULTNAME_SAVE(c, attrname, flags,  
parsestr) { CLASS_ATTR_DEFAULTNAME(c,attrname,flags,parsestr);  
CLASS_ATTR_SAVE(c,attrname,flags); }
```

A convenience wrapper for both [CLASS_ATTR_DEFAULTNAME](#) and [CLASS_ATTR_SAVE](#).

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

See also

[CLASS_ATTR_DEFAULTNAME](#)

[CLASS_ATTR_SAVE](#)

```
33.1.4.18 #define CLASS_ATTR_DEFAULTNAME_SAVE_PAINT(c, attrname, flags,  
parsestr) { CLASS_ATTR_DEFAULTNAME(c,attrname,flags,parsestr);  
CLASS_ATTR_SAVE(c,attrname,flags); CLASS_ATTR_PAINT(c,attrname,flags); }
```

A convenience wrapper for [CLASS_ATTR_DEFAULTNAME](#), [CLASS_ATTR_SAVE](#), and [CLASS_ATTR_PAINT](#).

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

See also

[CLASS_ATTR_DEFAULTNAME](#)

[CLASS_ATTR_PAINT](#)

[CLASS_ATTR_SAVE](#)

```
33.1.4.19 #define CLASS_ATTR_DOUBLE(c, attrname, flags,
    structname, structmember) class_addattr((c),attr_offset_-
    new(attrname,USESYM(float64),(flags),(method)0L,(method)0L,calcoffset(structname,structmember)))
```

Create a 64-bit float attribute and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

```
33.1.4.20 #define CLASS_ATTR_DOUBLE_ARRAY(c, attrname, flags,
    structname, structmember, size) class_addattr((c),attr_offset_array_-
    new(attrname,USESYM(float64),(size),(flags),(method)0L,(method)0L,/*fix*/,calcoffset(structname,structm
```

Create an array-of-64bit-floats attribute of fixed length, and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

size The number of doubles in the array.

```
33.1.4.21 #define CLASS_ATTR_DOUBLE_VARSIZE(c, attrname, flags, structname,
structmember, sizemember, maxsize) class_addattr((c),attr_offset_array_-
new(attrname,USESYM(float64),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizememb
```

Create an array-of-64bit-floats attribute of variable length, and add it to a Max class.

Parameters

- c* The class pointer.
- attrname* The name of this attribute as a C-string.
- flags* Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).
- structname* The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.
- structmember* The C identifier of the member in the struct that holds the value of this attribute.
- sizemember* The actual number of items in the double array at any given moment.
- maxsize* The maximum number of items in the double array, i.e. the number of members allocated for the array in the struct.

```
33.1.4.22 #define CLASS_ATTR_ENUM(c, attrname, flags, parsestr) {
CLASS_ATTR_STYLE(c,attrname,flags,"enum"); CLASS_ATTR_-
ATTR_PARSE(c,attrname,"enumvals",USESYM(atom),flags,parsestr);
}
```

Add a new attribute to the specified attribute to specify a list of choices to display in a menu for the Max inspector.

Parameters

- c* The class pointer.
- attrname* The name of the attribute as a C-string.
- flags* Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).
- parsestr* A C-string, which will be parsed into an array of atoms to set the initial value.

Remarks

This macro automatically calls

```
CLASS_ATTR_STYLE(c,attrname,flags,"enum").
```

See also

[CLASS_ATTR_ENUMINDEX](#)

```
33.1.4.23 #define CLASS_ATTR_ENUMINDEX(c, attrname, flags, parsestr) {
CLASS_ATTR_STYLE(c,attrname,flags,"enumindex"); CLASS_ATTR_-
ATTR_PARSE(c,attrname,"enumvals",USESYM(atom),flags,parsestr);
}
```

Add a new attribute to the specified attribute to specify a list of choices to display in a menu for the Max inspector.

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

Remarks

This macro automatically calls

```
CLASS_ATTR_STYLE(c, attrname, flags, "enumindex") .
```

See also

[CLASS_ATTR_ENUM](#)

33.1.4.24 #define CLASS_ATTR_FILTER_CLIP(c, attrname, minval, maxval)**Value:**

```
{ t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname)); \
    attr_addfilter_clip(theattr,minval,maxval,1,1); }
```

Add a filter to the attribute to limit both the lower and upper bounds of a value.

The limiting will be performed by the default attribute accessor.

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

minval The maximum acceptable value to which the attribute will be limited.

maxval The maximum acceptable value to which the attribute will be limited.

See also**33.1.4.25 #define CLASS_ATTR_FILTER_MAX(c, attrname, maxval)****Value:**

```
{ t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname)); \
    attr_addfilter_clip(theattr,0,maxval,0,1); }
```

Add a filter to the attribute to limit the upper bound of a value.

The limiting will be performed by the default attribute accessor.

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

maxval The maximum acceptable value to which the attribute will be limited.

See also

[CLASS_ATTR_FILTER_MIN](#)
[CLASS_ATTR_FILTER_CLIP](#)
[CLASS_ATTR_MAX](#)

33.1.4.26 #define CLASS_ATTR_FILTER_MIN(*c*, *attrname*, *minval*)

Value:

```
{ t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname)); \
    attr_addfilter_clip(theattr,minval,0,1,0); }
```

Add a filter to the attribute to limit the lower bound of a value.

The limiting will be performed by the default attribute accessor.

Parameters

c The class pointer.
attrname The name of the attribute as a C-string.
minval The minimum acceptable value to which the attribute will be limited.

See also

[CLASS_ATTR_FILTER_MAX](#)
[CLASS_ATTR_FILTER_CLIP](#)
[CLASS_ATTR_MIN](#)

33.1.4.27 #define CLASS_ATTR_FLOAT(*c*, *attrname*, *flags*, *structname*, *structmember*) class_addattr((*c*),attr_offset_- new(attrname,USESYM(float32),(flags),(method)0L,(method)0L,calcoffset(structname,structmember)))

Create a 32-bit float attribute and add it to a Max class.

Parameters

c The class pointer.
attrname The name of this attribute as a C-string.
flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).
structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.
structmember The C identifier of the member in the struct that holds the value of this attribute.

33.1.4.28 #define CLASS_ATTR_FLOAT_ARRAY(*c*, *attrname*, *flags*, *structname*, *structmember*, *size*) class_addattr((*c*),attr_offset_array_- new(attrname,USESYM(float32),(size),(flags),(method)0L,(method)0L,0/*fix*/,calcoffset(structname,structm))

Create an array-of-32bit-floats attribute of fixed length, and add it to a Max class.

Parameters

c The class pointer.
attrname The name of this attribute as a C-string.
flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).
structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.
structmember The C identifier of the member in the struct that holds the value of this attribute.
size The number of floats in the array.

```
33.1.4.29 #define CLASS_ATTR_FLOAT_VARSIZE(c, attrname, flags, structname,
    structmember, sizemember, maxsize) class_addattr((c),attr_offset_array_-
    new(attrname,USESYM(float32),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizememb...
```

Create an array-of-32bit-floats attribute of variable length, and add it to a Max class.

Parameters

c The class pointer.
attrname The name of this attribute as a C-string.
flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).
structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.
structmember The C identifier of the member in the struct that holds the value of this attribute.
sizemember The actual number of items in the float array at any given moment.
maxsize The maximum number of items in the float array, i.e. the number of members allocated for the array in the struct.

```
33.1.4.30 #define CLASS_ATTR_INVISIBLE(c, attrname, flags) CLASS_ATTR_ATTR_-
    PARSE(c,attrname,"invisible",USESYM(long),flags,"1")
```

Add a new attribute to the specified attribute to flag an attribute as invisible to the Max inspector.

Parameters

c The class pointer.
attrname The name of the attribute as a C-string.
flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

```
33.1.4.31 #define CLASS_ATTR_LABEL(c, attrname, flags, labelstr) CLASS_ATTR_ATTR_-
    FORMAT(c,attrname,"label",USESYM(symbol),flags,"s",gensym(labelstr))
```

Add a new attribute to the specified attribute to specify an a human-friendly label for the Max inspector.

Parameters

c The class pointer.
attrname The name of the attribute as a C-string.
flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).
labelstr A C-string, which will be parsed into an array of atoms to set the initial value.

```
33.1.4.32 #define CLASS_ATTR_LONG(c, attrname, flags,
                           structname, structmember) class_addattr((c),attr_offset_-
                           new(attrname,USESYM(long),(flags),(method)0L,(method)0L,calcoffset(structname,structmember)))
```

Create a long integer attribute and add it to a Max class.

Parameters

- c* The class pointer.
- attrname* The name of this attribute as a C-string.
- flags* Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).
- structname* The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.
- structmember* The C identifier of the member in the struct that holds the value of this attribute.

```
33.1.4.33 #define CLASS_ATTR_LONG_ARRAY(c, attrname, flags,
                           structname, structmember, size) class_addattr((c),attr_offset_array_-
                           new(attrname,USESYM(long),(size),(flags),(method)0L,(method)0L,/*fix*/,calcoffset(structname,structmembe
```

Create an array-of-long-integers attribute of fixed length, and add it to a Max class.

Parameters

- c* The class pointer.
- attrname* The name of this attribute as a C-string.
- flags* Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).
- structname* The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.
- structmember* The C identifier of the member in the struct that holds the value of this attribute.
- size* The number of longs in the array.

```
33.1.4.34 #define CLASS_ATTR_LONG_VARSIZE(c, attrname, flags, structname,
                           structmember, sizemember, maxsize) class_addattr((c),attr_offset_array_-
                           new(attrname,USESYM(long),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizemember),
```

Create an array-of-long-integers attribute of variable length, and add it to a Max class.

Parameters

- c* The class pointer.
- attrname* The name of this attribute as a C-string.
- flags* Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).
- structname* The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.
- structmember* The C identifier of the member in the struct that holds the value of this attribute.
- sizemember* The actual number of items in the long array at any given moment.
- maxsize* The maximum number of items in the long array, i.e. the number of members allocated for the array in the struct.

```
33.1.4.35 #define CLASS_ATTR_MAX(c, attrname, flags, parsestr) {  
    t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname));  
    CLASS_ATTR_ATTRPARSE(c,attrname,"max",(t_symbol  
    *)object_method(theattr,USESYM(gettype)),flags,parsestr); }
```

Add a new attribute to the specified attribute to specify an upper range.

The values will not be automatically limited.

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

See also

[CLASS_ATTR_MIN](#)

[CLASS_ATTR_FILTER_MAX](#)

[CLASS_ATTR_FILTER_CLIP](#)

```
33.1.4.36 #define CLASS_ATTR_MIN(c, attrname, flags, parsestr) {  
    t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname));  
    CLASS_ATTR_ATTRPARSE(c,attrname,"min",(t_symbol  
    *)object_method(theattr,USESYM(gettype)),flags,parsestr); }
```

Add a new attribute to the specified attribute to specify a lower range.

The values will not be automatically limited.

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

See also

[CLASS_ATTR_MAX](#)

[CLASS_ATTR_FILTER_MAX](#)

[CLASS_ATTR_FILTER_CLIP](#)

```
33.1.4.37 #define CLASS_ATTR_OBJ(c, attrname, flags,  
    structname, structmember) class_addattr((c),attr_offset_-  
    new(attrname,USESYM(object),(flags),(method)0L,(method)0L,calcoffset(structname,structmember)))
```

Create a [t_object*](#) attribute and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

33.1.4.38 #define CLASS_ATTR_OBJ_ARRAY(c, attrname, flags, structname, structmember, size) class_addattr((c),attr_offset_array_-new(attrname,USESYM(object),(size),(flags),(method)0L,(method)0L,0/*fix*/,calcoffset(structname,structmembe

Create an array-of-objects attribute of fixed length, and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

size The number of items in the [t_object](#)* array.

33.1.4.39 #define CLASS_ATTR_OBJ_VARSIZE(c, attrname, flags, structname, structmember, sizemember, maxsize) class_addattr((c),attr_offset_array_-new(attrname,USESYM(object),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizemember

Create an array-of-objects attribute of variable length, and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

sizemember The actual number of items in the [t_object](#)* array at any given moment.

maxsize The maximum number of items in the [t_object](#)* array, i.e. the number of members allocated for the array in the struct.

33.1.4.40 #define CLASS_ATTR_ORDER(c, attrname, flags, parsestr) CLASS_ATTR_ATTR_-PARSE(c,attrname,"order",USESYM(long),flags,parsestr)

Add a new attribute to the specified attribute to specify a default order in which to list attributes.

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

Remarks

A value of zero indicates that there is no ordering. Ordering values begin at 1. For example:

```
CLASS_ATTR_ORDER(c, "firstattr", 0, "1");
CLASS_ATTR_ORDER(c, "secondattr", 0, "2");
CLASS_ATTR_ORDER(c, "thirdattr", 0, "3");
```

33.1.4.41 #define CLASS_ATTR_PAINT(*c*, *attrname*, *flags*) CLASS_ATTR_ATTR_- PARSE(*c,attrname,"paint",USESYM(long),flags,"1"*)

Add a new attribute indicating that any changes to the specified attribute will trigger a call to the object's paint method.

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

33.1.4.42 #define CLASS_ATTR_REMOVE_FLAGS(*c*, *attrname*, *flags*)

Value:

```
{ t_object *theattr=(t_object *)class_attr_get(c,gensym(attrname)); \
    long oldflags = object_method(theattr,gensym("getflags")); \
    object_method(theattr,gensym("setflags"),oldflags&(~(flags))); }
```

Remove flags from an attribute.

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to remove from this attribute, as defined in [e_max_attrflags](#).

33.1.4.43 #define CLASS_ATTR_RGBA(*c*, *attrname*, *flags*, *structname*, *structmember*)

Value:

```
{ CLASS_ATTR_DOUBLE_ARRAY(c,attrname,flags,structname,structmember,4); \
    CLASS_ATTR_ACCESSORS(c,attrname,NULL,jgraphics_attr_setrgba); \
    CLASS_ATTR_PAINT(c,attrname,0); }
```

Create a color ([t_jrgba](#)) attribute and add it to a Max class.

Parameters

- c* The class pointer.
- attrname* The name of this attribute as a C-string.
- flags* Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).
- structname* The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.
- structmember* The C identifier of the member in the struct that holds the value of this attribute.

```
33.1.4.44 #define CLASS_ATTR_SAVE(c, attrname, flags) CLASS_ATTR_ATTR_-  
PARSE(c,attrname,"save",USESYM(long),flags,"1")
```

Add a new attribute to the specified attribute to indicate that the specified attribute should be saved with the patcher.

Parameters

- c* The class pointer.
- attrname* The name of the attribute as a C-string.
- flags* Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

```
33.1.4.45 #define CLASS_ATTR_STYLE(c, attrname, flags, parsestr) CLASS_ATTR_ATTR_-  
PARSE(c,attrname,"style",USESYM(symbol),flags,parsestr)
```

Add a new attribute to the specified attribute to specify an editor style for the Max inspector.

Available styles include

- "text" : a text editor
- "onoff" : a toggle switch
- "rgba" : a color chooser
- "enum" : a menu of available choices, whose symbol will be passed upon selection
- "enumindex" : a menu of available choices, whose index will be passed upon selection
- "rect" : a style for displaying and editing [t_rect](#) values
- "font" : a font chooser
- "file" : a file chooser dialog

Parameters

- c* The class pointer.
- attrname* The name of the attribute as a C-string.
- flags* Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).
- parsestr* A C-string, which will be parsed into an array of atoms to set the initial value.

```
33.1.4.46 #define CLASS_ATTR_STYLE_LABEL(c, attrname,
    flags, stylestr, labelstr) { CLASS_ATTR_ATTR_-
    PARSE(c,attrname,"style",USESYM(symbol),flags,stylestr); CLASS_ATTR_-
    ATTR_FORMAT(c,attrname,"label",USESYM(symbol),flags,"s",gensym(labelstr));
}
```

A convenience wrapper for [CLASS_ATTR_STYLE](#), and [CLASS_ATTR_LABEL](#).

Parameters

c The class pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

stylestr A C-string that names the style for the attribute. See [CLASS_ATTR_STYLE](#) for the available styles.

labelstr A C-string that names the category to which the attribute is assigned in the inspector.

See also

[CLASS_ATTR_STYLE](#)

[CLASS_ATTR_LABEL](#)

```
33.1.4.47 #define CLASS_ATTR_SYM(c, attrname, flags,
    structname, structmember) class_addattr((c),attr_offset_-
    new(attrname,USESYM(symbol),(flags),(method)0L,(method)0L,calcoffset(structname,structmember)))
```

Create a [t_symbol*](#) attribute and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

```
33.1.4.48 #define CLASS_ATTR_SYM_ARRAY(c, attrname, flags,
    structname, structmember, size) class_addattr((c),attr_offset_array_-
    new(attrname,USESYM(symbol),(size),(flags),(method)0L,(method)0L,/*fix*/,calcoffset(structname,structmem))
```

Create an array-of-symbols attribute of fixed length, and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

size The number of items in the [t_symbol*](#) array.

33.1.4.49 #define CLASS_ATTR_SYM_VARSIZE(c, attrname, flags, structname, structmember, sizemember, maxsize) class_addattr((c),attr_offset_array - new(attrname,USESYM(symbol),(maxsize),(flags),(method)0L,(method)0L,calcoffset(structname,sizememb

Create an array-of-symbols attribute of variable length, and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

sizemember The actual number of items in the [t_symbol*](#) array at any given moment.

maxsize The maximum number of items in the [t_symbol*](#) array, i.e. the number of members allocated for the array in the struct.

33.1.4.50 #define CLASS_METHOD_ATTR_PARSE(c, methodname, attrname, type, flags, parsestring)

Value:

```
{ t_hashtab *methods=NULL; \
    t_object *m=NULL; \
    methods = (t_hashtab *)class_extra_lookup(c,gensym("methods")); \
    if (methods) { \
        hashtab_lookup(methods,gensym((methodname)),&m); \
        if (m) \
            object_addattr_parse(m,attrname,type,flags,parsestring); \
    } \
}
```

Define and add attributes to class methods.

Parameters

c The class pointer.

methodname The name of the existing method as a C-string.

attrname The name of the attribute to add as a C-string.

type The datatype of the attribute to be added.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestring A C-string, which will be parsed into an array of atoms to set the initial value.

Remarks

An example which makes a method invisible to users:

```
class_addmethod(c, (method)my_foo, "foo", 0);
CLASS_METHOD_ATTR_PARSE(c, "foo", "undocumented", gensym("long"), 0, "1");
```

**33.1.4.51 #define CLASS_STICKY_ATTR(c, name, flags, parsestr) {
 t_object *attr = attribute_new_parse(name,NULL,flags,parsestr);
 class_sticky(c,gensym("sticky_attr")),gensym(name),attr); }**

Create an attribute, and add it to all following attribute declarations.

The block is closed by a call to [CLASS_STICKY_ATTR_CLEAR](#).

Parameters

c The class pointer.

name The name of the new attribute to create as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

Remarks

The most common use of CLASS_STICKY_ATTR is for creating multiple attributes with the same category, as in this example:

```
CLASS_STICKY_ATTR(c, "category", 0, "Foo");

CLASS_ATTR_DOUBLE(c, "bar", 0, t_myobject, x_bar);
CLASS_ATTR_LABEL(c, "bar", 0, "A Bar");

CLASS_ATTR_CHAR(c, "switch", 0, t_myobject, x_switch);
CLASS_ATTR_STYLE_LABEL(c, "switch", 0, "onoff", "Bar Switch");

CLASS_ATTR_DOUBLE(c, "flow", 0, t_myobject, x_flow);
CLASS_ATTR_LABEL(c, "flow", 0, "Flow Amount");

CLASS_STICKY_ATTR_CLEAR(c, "category");
```

See also

[CLASS_STICKY_ATTR_CLEAR](#)

33.1.4.52 #define CLASS_STICKY_ATTR_CLEAR(c, name) class_sticky_clear(c,gensym("sticky_attr"),name?gensym(name):NULL)

Close a [CLASS_STICKY_ATTR](#) block.

Parameters

c The class pointer.

name The name of the sticky attribute as a C-string.

See also

[CLASS_STICKY_ATTR](#)

**33.1.4.53 #define CLASS_STICKY_METHOD(c, name, flags, parsestr) {
 t_object *attr = attribute_new_parse(name,NULL,flags,parsestr);
 class_sticky(c,gensym("sticky_method")),gensym(name),attr); }**

Create an attribute, and add it to all following method declarations.

The block is closed by a call to [CLASS_STICKY_METHOD_CLEAR](#).

Parameters

c The class pointer.

name The name of the new attribute to create as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

Remarks

The most common use of CLASS_STICKY_ATTR is for creating multiple attributes with the same category, as in this example:

```
CLASS_STICKY_METHOD(c, "undocumented", 0, "1");

// add some methods here with class_add_method()
// the undocumented attribute for methods means that the ref-page
// generator will ignore these methods.

CLASS_STICKY_METHOD_CLEAR(c, "undocumented");
```

See also

[CLASS_STICKY_METHOD_CLEAR](#)

33.1.4.54 #define CLASS_STICKY_METHOD_CLEAR(c, name) class_sticky_clear(c,gensym("sticky_method"),name?gensym(name):clear)

Close a [CLASS_STICKY_METHOD](#) block.

Parameters

c The class pointer.

name The name of the sticky attribute as a C-string.

See also

[CLASS_STICKY_METHOD](#)

33.1.4.55 #define OBJ_ATTR_ATOM(x, attrname, flags, val) OBJ_ATTR_FORMAT(x,attrname,USESYM(atom),flags,"a",val)

Create an instance-local [t_atom](#) attribute and add it to a Max class.

Parameters

x The object pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

val Pointer to the value.

33.1.4.56 #define OBJ_ATTR_ATOM_ARRAY OBJ_ATTR_ATOMS

Create an instance-local array-of-atoms attribute of fixed length, and add it to the object.

Parameters

- x* The object pointer.
- attrname* The name of this attribute as a C-string.
- flags* Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).
- count* The number of items in the [t_atom](#) array.
- vals* Pointer to the values.

33.1.4.57 #define OBJ_ATTR_CHAR(x, attrname, flags, val) OBJ_ATTR_FORMAT(x,attrname,USESYM(char),flags,"c",val)

Create an instance-local char attribute and add it to a Max class.

Parameters

- x* The object pointer.
- attrname* The name of this attribute as a C-string.
- flags* Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).
- val* Pointer to the value.

33.1.4.58 #define OBJ_ATTR_CHAR_ARRAY(x, attrname, flags, count, vals) OBJ_ATTR_FORMAT(x,attrname,USESYM(char),flags,"C",count,vals)

Create an instance-local array-of-chars attribute of fixed length, and add it to the object.

Parameters

- x* The object pointer.
- attrname* The name of this attribute as a C-string.
- flags* Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).
- count* The number of items in the char array.
- vals* Pointer to the values.

33.1.4.59 #define OBJ_ATTR_DEFAULT(x, attrname, flags, parsestr) { t_object *theattr=(t_object *)object_attr_get(x,gensym(attrname)); OBJ_ATTR_ATTR_PARSE(x,attrname,"default",(t_symbol *)object_method(theattr,USESYM(gettype)),flags,parsestr); }

An instance-attribute version of [CLASS_ATTR_DEFAULT](#).

Parameters

- x* The [t_object](#) instance pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

See also

[CLASS_ATTR_DEFAULT](#)

```
33.1.4.60 #define OBJ_ATTR_DEFAULT_SAVE(x, attrname, flags, parsestr) { OBJ_-  
ATTR_DEFAULT(x,attrname,flags,parsestr); OBJ_ATTR_SAVE(x,attrname,flags);  
}
```

An instance-attribute version of [CLASS_ATTR_DEFAULT_SAVE](#).

Parameters

x The [t_object](#) instance pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in [e_max_attrflags](#).

parsestr A C-string, which will be parsed into an array of atoms to set the initial value.

See also

[CLASS_ATTR_DEFAULT_SAVE](#)

```
33.1.4.61 #define OBJ_ATTR_DOUBLE(x, attrname, flags, val) OBJ_ATTR_-  
FORMAT(x,attrname,USESYM(float64),flags,"d",val)
```

Create an instance-local 64bit float attribute and add it to a Max class.

Parameters

x The object pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

val Pointer to the value.

```
33.1.4.62 #define OBJ_ATTR_DOUBLE_ARRAY(x, attrname, flags, count,  
vals) OBJ_ATTR_FORMAT(x,attrname,USESYM(float64),flags,"D",count,vals)
```

Create an instance-local array-of-64bit-floats attribute of fixed length, and add it to the object.

Parameters

x The object pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

count The number of items in the double array.

vals Pointer to the values.

33.1.4.63 #define OBJ_ATTR_FLOAT(x, attrname, flags, val) OBJ_ATTR_FORMAT(x,attrname,USESYM(float32),flags,"f",val)

Create an instance-local 32bit float attribute and add it to a Max class.

Parameters

x The object pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

val Pointer to the value.

33.1.4.64 #define OBJ_ATTR_FLOAT_ARRAY(x, attrname, flags, count, vals) OBJ_ATTR_FORMAT(x,attrname,USESYM(float32),flags,"F",count,vals)

Create an instance-local array-of-32bit-floats attribute of fixed length, and add it to the object.

Parameters

x The object pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

count The number of items in the float array.

vals Pointer to the values.

33.1.4.65 #define OBJ_ATTR_LONG(x, attrname, flags, val) OBJ_ATTR_FORMAT(x,attrname,USESYM(long),flags,"l",val)

Create an instance-local long integer attribute and add it to a Max class.

Parameters

x The object pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

val Pointer to the value.

33.1.4.66 #define OBJ_ATTR_LONG_ARRAY(x, attrname, flags, count, vals) OBJ_ATTR_FORMAT(x,attrname,USESYM(long),flags,"L",count,vals)

Create an instance-local array-of-long-integers attribute of fixed length, and add it to the object.

Parameters

x The object pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

count The number of items in the long array.

vals Pointer to the values.

33.1.4.67 #define OBJ_ATTR_OBJ(x, attrname, flags, val) OBJ_ATTR_FORMAT(x,attrname,USESYM(object),flags,"o",val)

Create an instance-local `t_object*` attribute and add it to a Max class.

Parameters

x The object pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in `e_max_attrflags`.

val Pointer to the value.

33.1.4.68 #define OBJ_ATTR_OBJ_ARRAY(x, attrname, flags, count, vals) OBJ_ATTR_FORMAT(x,attrname,USESYM(object),flags,"O",count,vals)

Create an instance-local array-of-objects attribute of fixed length, and add it to the object.

Parameters

x The object pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in `e_max_attrflags`.

count The number of items in the `t_object*` array.

vals Pointer to the values.

33.1.4.69 #define OBJ_ATTR_SAVE(x, attrname, flags) OBJ_ATTR_ATTR_PARSE(x,attrname,"save",USESYM(long),flags,"1")

An instance-attribute version of [CLASS_ATTR_SAVE](#).

Parameters

x The `t_object` instance pointer.

attrname The name of the attribute as a C-string.

flags Any flags you wish to declare for this new attribute, as defined in `e_max_attrflags`.

See also

[CLASS_ATTR_SAVE](#)

33.1.4.70 #define OBJ_ATTR_SYM(x, attrname, flags, val) OBJ_ATTR_FORMAT(x,attrname,USESYM(symbol),flags,"s",val)

Create an instance-local `t_symbol*` attribute and add it to a Max class.

Parameters

x The object pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in `e_max_attrflags`.

val Pointer to the value.

33.1.4.71 #define OBJ_ATTR_SYM_ARRAY(x, attrname, flags, count, vals) OBJ_ATTR_FORMAT(x,attrname,USESYM(symbol),flags,"S",count,vals)

Create an instance-local array-of-symbols attribute of fixed length, and add it to the object.

Parameters

x The object pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

count The number of items in the [t_symbol*](#) array.

vals Pointer to the values.

33.1.4.72 #define STATIC_ATTR_ATOM(c, attrname, flags, val) STATIC_ATTR_FORMAT(c,attrname,USESYM(atom),flags,"a",val)

Create a shared (static/global) [t_atom](#) attribute and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

val Pointer to the value.

33.1.4.73 #define STATIC_ATTR_ATOM_ARRAY STATIC_ATTR_ATOMS

Create a shared (static/global) array-of-atoms attribute of fixed length, and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

count The number of items in the [t_atom](#) array.

vals Pointer to the values.

33.1.4.74 #define STATIC_ATTR_CHAR(c, attrname, flags, val) STATIC_ATTR_FORMAT(c,attrname,USESYM(char),flags,"c",val)

Create a shared (static/global) char attribute and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

val Pointer to the value.

**33.1.4.75 #define STATIC_ATTR_CHAR_ARRAY(*c*, *attrname*, *flags*, *count*,
 vals) STATIC_ATTR_FORMAT(*c*,*attrname*,USESYM(char),*flags*,"C",*count*,*vals*)**

Create a shared (static/global) array-of-chars attribute of fixed length, and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

count The number of items in the char array.

vals Pointer to the values.

**33.1.4.76 #define STATIC_ATTR_DOUBLE(*c*, *attrname*, *flags*, *val*) STATIC_ATTR_-
 FORMAT(*c*,*attrname*,USESYM(float64),*flags*,"d",*val*)**

Create a shared (static/global) 64bit float attribute and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

val Pointer to the value.

**33.1.4.77 #define STATIC_ATTR_DOUBLE_ARRAY(*c*, *attrname*, *flags*, *count*,
 vals) STATIC_ATTR_FORMAT(*c*,*attrname*,USESYM(float64),*flags*,"D",*count*,*vals*)**

Create a shared (static/global) array-of-64bit-floats attribute of fixed length, and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

count The number of items in the double array.

vals Pointer to the values.

**33.1.4.78 #define STATIC_ATTR_FLOAT(*c*, *attrname*, *flags*, *val*) STATIC_ATTR_-
 FORMAT(*c*,*attrname*,USESYM(float32),*flags*,"f",*val*)**

Create a shared (static/global) 32bit float attribute and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

val Pointer to the value.

**33.1.4.79 #define STATIC_ATTR_FLOAT_ARRAY(*c*, *attrname*, *flags*, *count*,
 vals) STATIC_ATTR_FORMAT(*c*,*attrname*,USESYM(float32),*flags*,"F",*count*,*vals*)**

Create a shared (static/global) array-of-32bit-floats attribute of fixed length, and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

count The number of items in the float array.

vals Pointer to the values.

**33.1.4.80 #define STATIC_ATTR_LONG(*c*, *attrname*, *flags*, *val*) STATIC_ATTR_-
 FORMAT(*c*,*attrname*,USESYM(long),*flags*,"I",*val*)**

Create a shared (static/global) long integer attribute and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

val Pointer to the value.

**33.1.4.81 #define STATIC_ATTR_LONG_ARRAY(*c*, *attrname*, *flags*, *count*,
 vals) STATIC_ATTR_FORMAT(*c*,*attrname*,USESYM(long),*flags*,"L",*count*,*vals*)**

Create a shared (static/global) array-of-long-integers attribute of fixed length, and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

count The number of items in the long array.

vals Pointer to the values.

**33.1.4.82 #define STATIC_ATTR_OBJ(*c*, *attrname*, *flags*, *val*) STATIC_ATTR_-
 FORMAT(*c*,*attrname*,USESYM(object),*flags*,"o",*val*)**

Create a shared (static/global) [t_object](#)* attribute and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

val Pointer to the value.

**33.1.4.83 #define STATIC_ATTR_OBJ_ARRAY(*c*, *attrname*, *flags*, *count*,
 vals) STATIC_ATTR_FORMAT(*c*,*attrname*,USESYM(*object*),*flags*,"O",*count*,*vals*)**

Create a shared (static/global) array-of-objects attribute of fixed length, and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

count The number of items in the [t_object](#)* array.

vals Pointer to the values.

**33.1.4.84 #define STATIC_ATTR_SYM(*c*, *attrname*, *flags*, *val*) STATIC_ATTR_-
 FORMAT(*c*,*attrname*,USESYM(*symbol*),*flags*,"s",*val*)**

Create a shared (static/global) [t_symbol](#)* attribute and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

val Pointer to the value.

**33.1.4.85 #define STATIC_ATTR_SYM_ARRAY(*c*, *attrname*, *flags*, *count*,
 vals) STATIC_ATTR_FORMAT(*c*,*attrname*,USESYM(*symbol*),*flags*,"S",*count*,*vals*)**

Create a shared (static/global) array-of-symbols attribute of fixed length, and add it to a Max class.

Parameters

c The class pointer.

attrname The name of this attribute as a C-string.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

count The number of items in the [t_symbol](#)* array.

vals Pointer to the values.

**33.1.4.86 #define STRUCT_ATTR_ATOM(*c*, *flags*, *structname*, *structmember*) CLASS_ATTR_-
 ATOM(*c*,#*structmember*,*flags*,*structname*,*structmember*)**

Create a [t_atom](#) attribute and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

33.1.4.87 #define STRUCT_ATTR_ATOM_ARRAY(c, flags, structname, structmember, size) CLASS_ATTR_ATOM_ARRAY(c,#structmember,flags,structname,structmember,size)

Create an array-of-atoms attribute of fixed length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

size The number of items in the [t_atom](#) array.

33.1.4.88 #define STRUCT_ATTR_ATOM_VARSIZE(c, flags, structname, structmember, sizemember, maxsize) CLASS_ATTR_ATOM_VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)

Create an array-of-atoms attribute of variable length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

sizemember The actual number of items in the [t_atom](#) array at any given moment.

maxsize The maximum number of items in the [t_atom](#) array, i.e. the number of members allocated for the array in the struct.

33.1.4.89 #define STRUCT_ATTR_CHAR(c, flags, structname, structmember) CLASS_ATTR_CHAR(c,#structmember,flags,structname,structmember)

Create a char attribute and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

33.1.4.90 #define STRUCT_ATTR_CHAR_ARRAY(c, flags, structname, structmember, size) CLASS_ATTR_CHAR - ARRAY(c,#structmember,flags,structname,structmember,size)

Create an array-of-chars attribute of fixed length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

size The number of items in the char array.

33.1.4.91 #define STRUCT_ATTR_CHAR_VARSIZE(c, flags, structname, structmember, sizemember, maxsize) CLASS_ATTR_CHAR - VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)

Create an array-of-chars attribute of variable length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

sizemember The actual number of items in the char array at any given moment.

maxsize The maximum number of items in the char array, i.e. the number of members allocated for the array in the struct.

33.1.4.92 #define STRUCT_ATTR_DOUBLE(c, flags, structname, structmember) CLASS_ATTR_DOUBLE(c,#structmember,flags,structname,structmember)

Create a 64bit float attribute and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

33.1.4.93 #define STRUCT_ATTR_DOUBLE_ARRAY(c, flags, structname, structmember, size) CLASS_ATTR_DOUBLE_ARRAY(c,#structmember,flags,structname,structmember,size)

Create an array-of-64bit-floats attribute of fixed length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

size The number of items in the double array.

33.1.4.94 #define STRUCT_ATTR_DOUBLE_VARSIZE(c, flags, structname, structmember, sizemember, maxsize) CLASS_ATTR_DOUBLE_VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)

Create an array-of-64bit-floats attribute of variable length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

sizemember The actual number of items in the double array at any given moment.

maxsize The maximum number of items in the double array, i.e. the number of members allocated for the array in the struct.

33.1.4.95 #define STRUCT_ATTR_FLOAT(c, flags, structname, structmember) CLASS_ATTR_FLOAT(c,#structmember,flags,structname,structmember)

Create a 32bit float attribute and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

33.1.4.96 #define STRUCT_ATTR_FLOAT_ARRAY(c, flags, structname, structmember, size) CLASS_ATTR_FLOAT_ARRAY(c,#structmember,flags,structname,structmember,size)

Create an array-of-32bit-floats attribute of fixed length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

size The number of items in the floats array.

33.1.4.97 #define STRUCT_ATTR_FLOAT_VARSIZE(c, flags, structname, structmember, sizemember, maxsize) CLASS_ATTR_FLOAT_VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)

Create an array-of-32bit-floats attribute of variable length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

sizemember The actual number of items in the float array at any given moment.

maxsize The maximum number of items in the float array, i.e. the number of members allocated for the array in the struct.

**33.1.4.98 #define STRUCT_ATTR_LONG(c, flags, structname, structmember) CLASS_ATTR_-
LONG(c,#structmember,flags,structname,structmember)**

Create a long integer attribute and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

**33.1.4.99 #define STRUCT_ATTR_LONG_ARRAY(c, flags, struct-
name, structmember, size) CLASS_ATTR_LONG_-
ARRAY(c,#structmember,flags,structname,structmember,size)**

Create an array-of-long-integers attribute of fixed length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

size The number of items in the long array.

**33.1.4.100 #define STRUCT_ATTR_LONG_VARSIZE(c, flags, structname,
structmember, sizemember, maxsize) CLASS_ATTR_LONG_-
VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)**

Create an array-of-long-integers attribute of variable length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in [e_max_attrflags](#).

structname The C identifier for the struct (containing a valid [t_object](#) header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

sizemember The actual number of items in the long array at any given moment.

maxsize The maximum number of items in the long array, i.e. the number of members allocated for the array in the struct.

**33.1.4.101 #define STRUCT_ATTR_OBJ(c, flags, structname, structmember) CLASS_ATTR_-
OBJ(c,#structmember,flags,structname,structmember)**

Create a `t_object*` attribute and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in `e_max_attrflags`.

structname The C identifier for the struct (containing a valid `t_object` header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

**33.1.4.102 #define STRUCT_ATTR_OBJ_ARRAY(c, flags,
structname, structmember, size) CLASS_ATTR_OBJ_-
ARRAY(c,#structmember,flags,structname,structmember,size)**

Create an array-of-objects attribute of fixed length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in `e_max_attrflags`.

structname The C identifier for the struct (containing a valid `t_object` header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

size The number of items in the `t_object*` array.

**33.1.4.103 #define STRUCT_ATTR_OBJ_VARSIZE(c, flags, structname,
structmember, sizemember, maxsize) CLASS_ATTR_OBJ_-
VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)**

Create an array-of-objects attribute of variable length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in `e_max_attrflags`.

structname The C identifier for the struct (containing a valid `t_object` header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

sizemember The actual number of items in the `t_object*` array at any given moment.

maxsize The maximum number of items in the `t_object*` array, i.e. the number of members allocated for the array in the struct.

**33.1.4.104 #define STRUCT_ATTR_SYM(c, flags, structname, structmember) CLASS_ATTR_-
SYM(c,#structmember,flags,structname,structmember)**

Create a `t_symbol*` attribute and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in `e_max_attrflags`.

structname The C identifier for the struct (containing a valid `t_object` header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

**33.1.4.105 #define STRUCT_ATTR_SYM_ARRAY(c, flags,
structname, structmember, size) CLASS_ATTR_SYM_-
ARRAY(c,#structmember,flags,structname,structmember,size)**

Create an array-of-symbols attribute of fixed length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in `e_max_attrflags`.

structname The C identifier for the struct (containing a valid `t_object` header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

size The number of items in the `t_symbol*` array.

**33.1.4.106 #define STRUCT_ATTR_SYM_VARSIZE(c, flags, structname,
structmember, sizemember, maxsize) CLASS_ATTR_SYM_-
VARSIZE(c,#structmember,flags,structname,structmember,sizemember,maxsize)**

Create an array-of-symbols attribute of variable length, and add it to a Max class.

The name of the attribute is automatically determined by the name of the struct member.

Parameters

c The class pointer.

flags Any flags you wish to declare for this attribute, as defined in `e_max_attrflags`.

structname The C identifier for the struct (containing a valid `t_object` header) representing an instance of this class.

structmember The C identifier of the member in the struct that holds the value of this attribute.

sizemember The actual number of items in the `t_symbol*` array at any given moment.

maxsize The maximum number of items in the `t_symbol*` array, i.e. the number of members allocated for the array in the struct.

33.1.5 Enumeration Type Documentation

33.1.5.1 enum e_max_attrflags

Attribute flags.

Remarks

To create a readonly attribute, for example, you should pass ATTR_SET_OPAQUE or ATTR_SET_-OPAQUE_USER as a flag when you create your attribute.

Enumerator:

ATTR_FLAGS_NONE No flags.

ATTR_GET_OPAQUE The attribute cannot be queried by either max message when used inside of a CLASS_BOX object, nor from C code.

ATTR_SET_OPAQUE The attribute cannot be set by either max message when used inside of a CLASS_BOX object, nor from C code.

ATTR_GET_OPAQUE_USER The attribute cannot be queried by max message when used inside of a CLASS_BOX object, but *can* be queried from C code.

ATTR_SET_OPAQUE_USER The attribute cannot be set by max message when used inside of a CLASS_BOX object, but *can* be set from C code.

33.1.6 Function Documentation

33.1.6.1 t_max_err attr_addfilter_clip (void **x*, double *min*, double *max*, long *usemin*, long *usemax*)

Attaches a clip filter to an attribute.

The filter will clip any values sent to or retrieved from the attribute using the attribute's get and set functions.

Parameters

x Pointer to the attribute to receive the filter

min Minimum value for the clip filter

max Maximum value for the clip filter

usemin Sets this value to 0 if the minimum clip value should *not* be used. Otherwise, set the value to non-zero.

usemax Sets this value to 0 if the maximum clip value should *not* be used. Otherwise, set the value to non-zero.

Returns

This function returns the error code **MAX_ERR_NONE** if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.2 `t_max_err attr_addfilter_clip_scale (void *x, double scale, double min, double max, long usemin, long usemax)`

Attaches a clip/scale filter to an attribute.

The filter will clip and scale any values sent to or retrieved from the attribute using the attribute's get and set functions.

Parameters

x Pointer to the attribute to receive the filter

scale Scale value. Data sent to the attribute will be scaled by this amount. Data retrieved from the attribute will be scaled by its reciprocal. *Scaling occurs previous to clipping.*

min Minimum value for the clip filter

max Maximum value for the clip filter

usemin Sets this value to 0 if the minimum clip value should *not* be used. Otherwise, set the value to non-zero.

usemax Sets this value to 0 if the maximum clip value should *not* be used. Otherwise, set the value to non-zero.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.3 `t_max_err attr_addfilterget_clip (void *x, double min, double max, long usemin, long usemax)`

Attaches a clip filter to an attribute.

The filter will *only* clip values retrieved from the attribute using the attribute's get function.

Parameters

x Pointer to the attribute to receive the filter

min Minimum value for the clip filter

max Maximum value for the clip filter

usemin Sets this value to 0 if the minimum clip value should *not* be used. Otherwise, set the value to non-zero.

usemax Sets this value to 0 if the maximum clip value should *not* be used. Otherwise, set the value to non-zero.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.4 `t_max_err attr_addfilterget_clip_scale (void *x, double scale, double min, double max, long usemin, long usemax)`

Attaches a clip/scale filter to an attribute.

The filter will *only* clip and scale values retrieved from the attribute using the attribute's get function.

Parameters

- x* Pointer to the attribute to receive the filter
- scale* Scale value. Data retrieved from the attribute will be scaled by this amount. *Scaling occurs previous to clipping.*
- min* Minimum value for the clip filter
- max* Maximum value for the clip filter
- usemin* Sets this value to 0 if the minimum clip value should *not* be used. Otherwise, set the value to non-zero.
- usemax* Sets this value to 0 if the minimum clip value should *not* be used. Otherwise, set the value to non-zero.

Returns

This function returns the error code **MAX_ERR_NONE** if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.5 **t_max_err attr_addfilterget_proc (void *x, method proc)**

Attaches a custom filter method to an attribute.

The filter will *only* be called for values retrieved from the attribute using the attribute's `get` function.

Parameters

- x* Pointer to the attribute to receive the filter
- proc* A filter method

Returns

This function returns the error code **MAX_ERR_NONE** if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

Remarks

The filter method should be prototyped and implemented as described above for the [attr_addfilterset_proc\(\)](#) function.

33.1.6.6 **t_max_err attr_addfilterset_clip (void *x, double min, double max, long usemin, long usemax)**

Attaches a clip filter to an attribute.

The filter will *only* clip values sent to the attribute using the attribute's `set` function.

Parameters

- x* Pointer to the attribute to receive the filter
- min* Minimum value for the clip filter
- max* Maximum value for the clip filter
- usemin* Sets this value to 0 if the minimum clip value should *not* be used. Otherwise, set the value to non-zero.

usemax Sets this value to 0 if the minimum clip value should *not* be used. Otherwise, set the value to non-zero.

Returns

This function returns the error code **MAX_ERR_NONE** if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.7 **t_max_err attr_addfilterset_clip_scale (void *x, double scale, double min, double max, long usemin, long usemax)**

Attaches a clip/scale filter to an attribute.

The filter will *only* clip and scale values sent to the attribute using the attribute's `set` function.

Parameters

x Pointer to the attribute to receive the filter

scale Scale value. Data sent to the attribute will be scaled by this amount. *Scaling occurs previous to clipping.*

min Minimum value for the clip filter

max Maximum value for the clip filter

usemin Sets this value to 0 if the minimum clip value should *not* be used. Otherwise, set the value to non-zero.

usemax Sets this value to 0 if the minimum clip value should *not* be used. Otherwise, set the value to non-zero.

Returns

This function returns the error code **MAX_ERR_NONE** if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.8 **t_max_err attr_addfilterset_proc (void *x, method proc)**

Attaches a custom filter method to an attribute.

The filter will *only* be called for values retrieved from the attribute using the attribute's `set` function.

Parameters

x Pointer to the attribute to receive the filter

proc A filter method

Returns

This function returns the error code **MAX_ERR_NONE** if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

Remarks

The filter method should be prototyped and implemented as follows:

```
t_max_err myfiltermethod(void *parent, void *attr, long ac, t_atom *av);

t_max_err myfiltermethod(void *parent, void *attr, long ac, t_atom *av)
{
    long i;
    float temp;

    // this filter rounds off all values
    // assumes that the data is float
    for (i = 0; i < ac; i++) {
        temp = atom_getfloat(av + i);
        temp = (float)((long)(temp + 0.5));
        atom_setfloat(av + i, temp);
    }
    return MAX_ERR_NONE;
}
```

33.1.6.9 void attr_args_dictionary (t_dictionary *x, short ac, t_atom *av)

Create a dictionary of attribute-name, attribute-value pairs from an array of atoms containing an attribute definition list.

Parameters

- x** A dictionary instance pointer.
- ac** The number of atoms to parse in av.
- av** A pointer to the first of the array of atoms containing the attribute values.

Remarks

The code example below shows the creation of a list of atoms using [atom_setparse\(\)](#), and then uses that list of atoms to fill the dictionary with [attr_args_dictionary\(\)](#).

```
long ac = 0;
t_atom *av = NULL;
char parsebuf[4096];
t_dictionary *d = dictionary_new();
t_atom a;

sprintf(parsebuf, "@defrect %.6f %.6f %.6f %.6f @title Untitled @presentation 0
", r->x, r->y, r->width, r->height);
atom_setparse(&ac, &av, parsebuf);
attr_args_dictionary(d, ac, av);
atom_setobj(&a, d);
```

33.1.6.10 long attr_args_offset (short ac, t_atom *av)

Determines the point in an atom list where attribute arguments begin.

Developers can use this function to assist in the manual processing of attribute arguments, when [attr_args_process\(\)](#) doesn't provide the correct functionality for a particular purpose.

Parameters

- ac** The count of t_atoms in av
- av** An atom list

Returns

This function returns an offset into the atom list, where the first attribute argument occurs. For instance, the atom list `foo bar 3.0 @mode 6` would cause `attr_args_offset` to return 3 (the attribute `mode` appears at position 3 in the atom list).

33.1.6.11 void attr_args_process (void **x*, short *ac*, t_atom **av*)

Takes an atom list and properly set any attributes described within.

This function is typically used in an object's `new` method to conveniently process attribute arguments.

Parameters

- x* The object whose attributes will be processed
- ac* The count of `t_atoms` in *av*
- av* An atom list

Remarks

Here is a typical example of usage:

```
void *myobject_new(t_symbol *s, long ac, t_atom *av)
{
    t_myobject *x = NULL;

    if (x=(t_myobject *)object_alloc(myobject_class))
    {
        // initialize any data before processing
        // attributes to avoid overwriting
        // attribute argument-set values
        x->data = 0;

        // process attr args, if any
        attr_args_process(x, ac, av);
    }
    return x;
}
```

33.1.6.12 void attr_dictionary_process (void **x*, t_dictionary **d*)

Set attributes for an object that are defined in a dictionary.

Objects with dictionary constructors, such as UI objects, should call this method to set their attributes when an object is created.

Parameters

- x* The object instance pointer.
- d* The dictionary containing the attributes.

See also

[attr_args_process\(\)](#)

33.1.6.13 `t_object* attr_offset_array_new (C74_CONST char * name, t_symbol * type, long size, long flags, method mget, method mset, long offsetcount, long offset)`

Create a new attribute.

The attribute references an array of memory stored outside of itself, in the object's data structure. Attributes created using `attr_offset_array_new()` can be assigned either to classes (using the `class_addattr()` function) or to objects (using the `object_addattr()` function).

Parameters

`name` A name for the attribute, as a C-string

`type` A `t_symbol *` representing a valid attribute type. At the time of this writing, the valid type-symbols are: `_sym_char` (`char`), `_sym_long` (`long`), `_sym_float32` (32-bit float), `_sym_float64` (64-bit float), `_sym_atom` (Max `t_atom` pointer), `_sym_symbol` (Max `t_symbol` pointer), `_sym_pointer` (generic pointer) and `_sym_object` (Max `t_object` pointer).

`size` Maximum number of items that may be in the array.

`flags` Any attribute flags, expressed as a bitfield. Attribute flags are used to determine if an attribute is accessible for setting or querying. The available accessor flags are defined in `e_max_attrflags`.

`mget` The method to use for the attribute's get functionality. If `mget` is NULL, the default method is used. See the discussion under `attribute_new()`, for more information.

`mset` The method to use for the attribute's set functionality. If `mset` is NULL, the default method is used. See the discussion under `attribute_new()`, for more information.

`offsetcount` Byte offset into the object class's data structure of a long variable describing how many array elements (up to `size`) comprise the data to be referenced by the attribute. Typically, the `calcoffset` macro is used to calculate this offset.

`offset` Byte offset into the class data structure of the object which will "own" the attribute. The offset should point to the data to be referenced by the attribute. Typically, the `calcoffset` macro is used to calculate this offset.

Returns

This function returns the new attribute's object pointer if successful, or NULL if unsuccessful.

Remarks

For instance, to create a new attribute which references an array of 10 `t_atoms` (`atm`; the current number of "active" elements in the array is held in the variable `atmcount`) in an object class's data structure:

```
t_object *attr = attr_offset_array_new("myattrarray", _sym_atom / * matches da
ta size * /, 10 / * max * /, 0 / * no flags * /, (method)0L, (method)0L,
calcoffset(t_myobject, atmcount) / * count * /, calcoffset(t_myobject, atm) / * d
ata * /);
```

33.1.6.14 `t_object* attr_offset_new (C74_CONST char * name, C74_CONST t_symbol * type, long flags, C74_CONST method mget, C74_CONST method mset, long offset)`

Create a new attribute.

The attribute references memory stored outside of itself, in the object's data structure. Attributes created using `attr_offset_new()` can be assigned either to classes (using the `class_addattr()` function) or to objects (using the `object_addattr()` function).

Parameters

name A name for the attribute, as a C-string

type A `t_symbol *` representing a valid attribute type. At the time of this writing, the valid type-symbols are: `_sym_char` (char), `_sym_long` (long), `_sym_float32` (32-bit float), `_sym_float64` (64-bit float), `_sym_atom` (Max `t_atom` pointer), `_sym_symbol` (Max `t_symbol` pointer), `_sym_pointer` (generic pointer) and `_sym_object` (Max `t_object` pointer).

flags Any attribute flags, expressed as a bitfield. Attribute flags are used to determine if an attribute is accessible for setting or querying. The available accessor flags are defined in `e_max_attrflags`.

mget The method to use for the attribute's get functionality. If `mget` is NULL, the default method is used. See the discussion under `attribute_new()`, for more information.

mset The method to use for the attribute's set functionality. If `mset` is NULL, the default method is used. See the discussion under `attribute_new()`, for more information.

offset Byte offset into the class data structure of the object which will "own" the attribute. The offset should point to the data to be referenced by the attribute. Typically, the `calcoffset` macro (described above) is used to calculate this offset.

Returns

This function returns the new attribute's object pointer if successful, or NULL if unsuccessful.

Remarks

For instance, to create a new attribute which references the value of a double variable (`val`) in an object class's data structure:

```
t_object *attr = attr_offset_new("myattr", _sym_float64 / * matches data size
                                * /, 0 / * no flags * /, (method)0L, (method)0L, calcoffset(t_myobject, val));
```

33.1.6.15 `t_object* attribute_new (C74_CONST char * name, t_symbol * type, long flags, method mget, method mset)`

Create a new attribute.

The attribute will allocate memory and store its own data. Attributes created using `attribute_new()` can be assigned either to classes (using the `class_addattr()` function) or to objects (using the `object_addattr()` function).

Parameters

name A name for the attribute, as a C-string

type A `t_symbol *` representing a valid attribute type. At the time of this writing, the valid type-symbols are: `_sym_char` (char), `_sym_long` (long), `_sym_float32` (32-bit float), `_sym_float64` (64-bit float), `_sym_atom` (Max `t_atom` pointer), `_sym_symbol` (Max `t_symbol` pointer), `_sym_pointer` (generic pointer) and `_sym_object` (Max `t_object` pointer).

flags Any attribute flags, expressed as a bitfield. Attribute flags are used to determine if an attribute is accessible for setting or querying. The available accessor flags are defined in `e_max_attrflags`.

mget The method to use for the attribute's get functionality. If `mget` is NULL, the default method is used.

mset The method to use for the attribute's set functionality. If `mset` is NULL, the default method is used.

Returns

This function returns the new attribute's object pointer if successful, or NULL if unsuccessful.

Remarks

Developers wishing to define custom methods for get or set functionality need to prototype them as:

```
t_max_err myobject_myattr_get(t_myobject **x, void *attr, long *ac, t_atom **av
);
t_max_err myobject_myattr_set(t_myobject **x, void *attr, long ac, t_atom *av);
```

Implementation will vary, of course, but need to follow the following basic models. Note that, as with custom `getvalueof` and `setvalueof` methods for the object, assumptions are made throughout Max that `getbytes()` has been used for memory allocation. Developers are strongly urged to do the same:

```
t_max_err myobject_myattr_get(t_myobject **x, void *attr, long *ac, t_atom **av
)
{
    if (*ac && *av)
        // memory passed in; use it
    else {
        *ac = 1; // size of attr data
        *av = (t_atom *)getbytes(sizeof(t_atom) * (*ac));
        if (!(*av)) {
            *ac = 0;
            return MAX_ERR_OUT_OF_MEM;
        }
    }
    atom_setlong(*av, x->some_value);
    return MAX_ERR_NONE;
}

t_max_err myobject_myattr_set(t_myobject **x, void *attr, long ac, t_atom *av)
{
    if (ac && av) {
        x->some_value = atom_getlong(av);
    }
    return MAX_ERR_NONE;
}
```

33.1.6.16 `t_max_err object_addattr (void *x, t_object *attr)`

Attaches an attribute directly to an object.

Parameters

x An object to which the attribute should be attached

attr The attribute's pointer—this should be a pointer returned from `attribute_new()`, `attr_offset_new()` or `attr_offset_array_new()`.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.1.6.17 void* object_attr_get (void **x*, t_symbol **attrname*)

Returns the pointer to an attribute, given its name.

Parameters

- x* Pointer to the object whose attribute is of interest
- attrname* The attribute's name

Returns

This function returns a pointer to the attribute, if successful, or NULL, if unsuccessful.

33.1.6.18 t_max_err object_attr_get_rect (t_object **o*, t_symbol **name*, t_rect **rect*)

Gets the value of a [t_rect](#) attribute, given its parent object and name.

Do not use this on a [jbox](#) object -- use [jbox_get_rect_for_view\(\)](#) instead!

Parameters

- o* The attribute's parent object
- name* The attribute's name
- rect* The address of a valid [t_rect](#) whose values will be filled-in from the attribute.

Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.19 long object_attr_getchar_array (void **x*, t_symbol **s*, long *max*, uchar **vals*)

Retrieves the value of an attribute, given its parent object and name.

This function uses a developer-allocated array to copy data to. Developers wishing to retrieve the value of an attribute without pre-allocating memory should refer to the [object_attr_getvalueof\(\)](#) function.

Parameters

- x* The attribute's parent object
- s* The attribute's name
- max* The number of array elements in *vals*. The function will take care not to overwrite the bounds of the array.
- vals* Pointer to the first element of a pre-allocated array of unsigned char data.

Returns

This function returns the number of elements copied into *vals*.

Remarks

If the attribute is not of the type specified by the function, the function will attempt to coerce a valid value from the attribute.

33.1.6.20 `t_max_err object_attr_getcolor (t_object *b, t_symbol *attrname, t_jrgba *prgba)`

Gets the value of a `t_jrgba` attribute, given its parent object and name.

Parameters

b The attribute's parent object

attrname The attribute's name

prgba The address of a valid `t_jrgba` whose values will be filled-in from the attribute.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.1.6.21 `long object_attr_getdouble_array (void *x, t_symbol *s, long max, double *vals)`

Retrieves the value of an attribute, given its parent object and name.

This function uses a developer-allocated array to copy data to. Developers wishing to retrieve the value of an attribute without pre-allocating memory should refer to the `object_attr_getvalueof()` function.

Parameters

x The attribute's parent object

s The attribute's name

max The number of array elements in `vals`. The function will take care not to overwrite the bounds of the array.

vals Pointer to the first element of a pre-allocated array of double data.

Returns

This function returns the number of elements copied into `vals`.

Remarks

If the attribute is not of the type specified by the function, the function will attempt to coerce a valid value from the attribute.

33.1.6.22 `void object_attr_getdump (void *x, t_symbol *s, long argc, t_atom *argv)`

Forces a specified object's attribute to send its value from the object's dumpout outlet in the Max interface.

Parameters

x Pointer to the object whose attribute is of interest

s The attribute's name

argc Unused

argv Unused

33.1.6.23 float object_attr_getfloat (void **x*, t_symbol **s*)

Retrieves the value of an attribute, given its parent object and name.

Parameters

- x* The attribute's parent object
- s* The attribute's name

Returns

This function returns the value of the specified attribute, if successful, or 0, if unsuccessful.

Remarks

If the attribute is not of the type specified by the function, the function will attempt to coerce a valid value from the attribute.

33.1.6.24 long object_attr_getfloat_array (void **x*, t_symbol **s*, long *max*, float **vals*)

Retrieves the value of an attribute, given its parent object and name.

This function uses a developer-allocated array to copy data to. Developers wishing to retrieve the value of an attribute without pre-allocating memory should refer to the `object_attr_getvalueof()` function.

Parameters

- x* The attribute's parent object
- s* The attribute's name
- max* The number of array elements in *vals*. The function will take care not to overwrite the bounds of the array.
- vals* Pointer to the first element of a pre-allocated array of float data.

Returns

This function returns the number of elements copied into *vals*.

Remarks

If the attribute is not of the type specified by the function, the function will attempt to coerce a valid value from the attribute.

33.1.6.25 t_max_err object_attr_getjrgba (void **ob*, t_symbol **s*, t_jrgba **c*)

Retrieves the value of a color attribute, given its parent object and name.

Parameters

- ob* The attribute's parent object
- s* The attribute's name
- c* The address of a `t_jrgba` struct that will be filled with the attribute's color component values.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.1.6.26 long object_attr_getlong (void **x*, t_symbol **s*)

Retrieves the value of an attribute, given its parent object and name.

Parameters

- x* The attribute's parent object
- s* The attribute's name

Returns

This function returns the value of the specified attribute, if successful, or 0, if unsuccessful.

Remarks

If the attribute is not of the type specified by the function, the function will attempt to coerce a valid value from the attribute.

33.1.6.27 long object_attr_getlong_array (void **x*, t_symbol **s*, long *max*, long **vals*)

Retrieves the value of an attribute, given its parent object and name.

This function uses a developer-allocated array to copy data to. Developers wishing to retrieve the value of an attribute without pre-allocating memory should refer to the `object_attr_getvalueof()` function.

Parameters

- x* The attribute's parent object
- s* The attribute's name
- max* The number of array elements in *vals*. The function will take care not to overwrite the bounds of the array.
- vals* Pointer to the first element of a pre-allocated array of long data.

Returns

This function returns the number of elements copied into *vals*.

Remarks

If the attribute is not of the type specified by the function, the function will attempt to coerce a valid value from the attribute.

33.1.6.28 t_max_err object_attr_getpt (t_object **o*, t_symbol **name*, t_pt **pt*)

Gets the value of a `t_pt` attribute, given its parent object and name.

Parameters

- o* The attribute's parent object
- name* The attribute's name
- pt* The address of a valid `t_pt` whose values will be filled-in from the attribute.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.1.6.29 `t_max_err object_attr_getsize (t_object *o, t_symbol *name, t_size *size)`

Gets the value of a `t_size` attribute, given its parent object and name.

Parameters

- `o` The attribute's parent object
- `name` The attribute's name
- `size` The address of a valid `t_size` whose values will be filled-in from the attribute.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.1.6.30 `t_symbol* object_attr_getsym (void *x, t_symbol *s)`

Retrieves the value of an attribute, given its parent object and name.

Parameters

- `x` The attribute's parent object
- `s` The attribute's name

Returns

This function returns the value of the specified attribute, if successful, or the empty symbol (equivalent to `gensym("")` or `_sym_nothing`), if unsuccessful.

33.1.6.31 `long object_attr_getsym_array (void *x, t_symbol *s, long max, t_symbol **vals)`

Retrieves the value of an attribute, given its parent object and name.

This function uses a developer-allocated array to copy data to. Developers wishing to retrieve the value of an attribute without pre-allocating memory should refer to the `object_attr_getvalueof()` function.

Parameters

- `x` The attribute's parent object
- `s` The attribute's name
- `max` The number of array elements in `vals`. The function will take care not to overwrite the bounds of the array.
- `vals` Pointer to the first element of a pre-allocated array of `t_symbol *`s.

Returns

This function returns the number of elements copied into `vals`.

33.1.6.32 method object_attr_method (*void *x, t_symbol *methodname, void **attr, long *get*)

Returns the method of an attribute's get or set function, as well as a pointer to the attribute itself, from a message name.

Parameters

x Pointer to the object whose attribute is of interest

methodname The Max message used to call the attribute's get or set function. For example, gensym ("mode") or gensym ("getthresh").

attr A pointer to a void *, which will be set to the attribute pointer upon successful completion of the function

get A pointer to a long variable, which will be set to 1 upon successful completion of the function, if the queried method corresponds to the get function of the attribute.

Returns

This function returns the requested method, if successful, or NULL, if unsuccessful.

33.1.6.33 t_max_err object_attr_set_rect (*t_object *o, t_symbol *name, t_rect *rect*)

Sets the value of a *t_rect* attribute, given its parent object and name.

Do not use this on a jbox object -- use [jbox_get_rect_for_view\(\)](#) instead!

Parameters

o The attribute's parent object

name The attribute's name

rect The address of a valid *t_rect* whose values will be used to set the attribute.

Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.34 t_max_err object_attr_setchar_array (*void *x, t_symbol *s, long count, C74_CONST uchar *vals*)

Sets the value of an attribute, given its parent object and name.

The function will call the attribute's set method, using the data provided.

Parameters

x The attribute's parent object

s The attribute's name

count The number of array elements in vals

vals Pointer to the first element of an array of unsigned char data

Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.35 `t_max_err object_attr_setcolor (t_object *b, t_symbol *attrname, t_jrgba *prgba)`

Sets the value of a `t_jrgba` attribute, given its parent object and name.

Parameters

b The attribute's parent object

attrname The attribute's name

prgba The address of a valid `t_jrgba` whose values will be used to set the attribute.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.1.6.36 `t_max_err object_attr_setdouble_array (void *x, t_symbol *s, long count, double *vals)`

Sets the value of an attribute, given its parent object and name.

The function will call the attribute's `set` method, using the data provided.

Parameters

x The attribute's parent object

s The attribute's name

count The number of array elements in *vals*

vals Pointer to the first element of an array of double data

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.1.6.37 `t_max_err object_attr_setfloat (void *x, t_symbol *s, float c)`

Sets the value of an attribute, given its parent object and name.

The function will call the attribute's `set` method, using the data provided.

Parameters

x The attribute's parent object

s The attribute's name

c An floating point value; the new value for the attribute

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.1.6.38 `t_max_err object_attr_setfloat_array (void *x, t_symbol *s, long count, float *vals)`

Sets the value of an attribute, given its parent object and name.

The function will call the attribute's `set` method, using the data provided.

Parameters

- x* The attribute's parent object
- s* The attribute's name
- count* The number of array elements in *vals*
- vals* Pointer to the first element of an array of float data

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.39 `t_max_err object_attr_setjrgba (void *ob, t_symbol *s, t_jrgba *c)`

Sets the value of a color attribute, given its parent object and name.

The function will call the attribute's `set` method, using the data provided.

Parameters

- ob* The attribute's parent object
- s* The attribute's name
- c* The address of a [t_jrgba](#) struct that contains the new color.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.40 `t_max_err object_attr_setlong (void *x, t_symbol *s, long c)`

Sets the value of an attribute, given its parent object and name.

The function will call the attribute's `set` method, using the data provided.

Parameters

- x* The attribute's parent object
- s* The attribute's name
- c* An integer value; the new value for the attribute

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.41 `t_max_err object_attr_setlong_array (void *x, t_symbol *s, long count, long *vals)`

Sets the value of an attribute, given its parent object and name.

The function will call the attribute's `set` method, using the data provided.

Parameters

- x* The attribute's parent object
- s* The attribute's name
- count* The number of array elements in *vals*
- vals* Pointer to the first element of an array of long data

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.42 `t_max_err object_attr_setparse (t_object *x, t_symbol *s, C74_CONST char *parsestr)`

Set an attribute value with one or more atoms parsed from a C-string.

Parameters

- x* The object whose attribute will be set.
- s* The name of the attribute to set.
- parsestr* A C-string to parse into an array of atoms to set the attribute value.

Returns

A Max error code.

See also

[atom_setparse\(\)](#)

33.1.6.43 `t_max_err object_attr_setpt (t_object *o, t_symbol *name, t_pt *pt)`

Sets the value of a `t_pt` attribute, given its parent object and name.

Parameters

- o* The attribute's parent object
- name* The attribute's name
- pt* The address of a valid `t_pt` whose values will be used to set the attribute.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.44 t_max_err object_attr_setsize (t_object **o*, t_symbol **name*, t_size **size*)

Sets the value of a `t_size` attribute, given its parent object and name.

Parameters

- o* The attribute's parent object
- name* The attribute's name
- size* The address of a valid `t_size` whose values will be used to set the attribute.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.1.6.45 t_max_err object_attr_setsym (void **x*, t_symbol **s*, t_symbol **c*)

Sets the value of an attribute, given its parent object and name.

The function will call the attribute's `set` method, using the data provided.

Parameters

- x* The attribute's parent object
- s* The attribute's name
- c* A `t_symbol` *; the new value for the attribute

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.1.6.46 t_max_err object_attr_setsym_array (void **x*, t_symbol **s*, long *count*, t_symbol *vals*)**

Sets the value of an attribute, given its parent object and name.

The function will call the attribute's `set` method, using the data provided.

Parameters

- x* The attribute's parent object
- s* The attribute's name
- count* The number of array elements in *vals*
- vals* Pointer to the first element of an array of `t_symbol` *s

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.1.6.47 t_max_err object_attr_setvalueof (void *x, t_symbol *s, long argc, t_atom *argv)

Sets the value of an object's attribute.

Parameters

- x* Pointer to the object whose attribute is of interest
- s* The attribute's name
- argc* The count of arguments in *argv*
- argv* Array of *t_atoms*; the new desired data for the attribute

Returns

This function returns the error code **MAX_ERR_NONE** if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.48 long object_attr_usercanget (void *x, t_symbol *s)

Determines if the value of an object's attribute can be queried from the Max interface (i.e. if its **ATTR_GET_OPAQUE_USER** flag is set).

Parameters

- x* Pointer to the object whose attribute is of interest
- s* The attribute's name

Returns

This function returns 1 if the value of the attribute can be queried from the Max interface. Otherwise, it returns 0.

33.1.6.49 long object_attr_usercanset (void *x, t_symbol *s)

Determines if an object's attribute can be set from the Max interface (i.e. if its **ATTR_SET_OPAQUE_USER** flag is set).

Parameters

- x* Pointer to the object whose attribute is of interest
- s* The attribute's name

Returns

This function returns 1 if the attribute can be set from the Max interface. Otherwise, it returns 0.

33.1.6.50 t_max_err object_chuckattr (void *x, t_symbol *attrsym)

Detach an attribute from an object that was previously attached with [object_addattr\(\)](#).

This function will *not* free the attribute (use [object_free\(\)](#) to do this manually).

Parameters

- x* The object to which the attribute is attached
- attrsym* The attribute's name

Returns

This function returns the error code **MAX_ERR_NONE** if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.51 t_max_err object_deleteattr (void * *x*, t_symbol * *attrsym*)

Detach an attribute from an object that was previously attached with [object_addattr\(\)](#).

The function will also free all memory associated with the attribute. If you only wish to detach the attribute, without freeing it, see the [object_chuckattr\(\)](#) function.

Parameters

- x* The object to which the attribute is attached
- attrsym* The attribute's name

Returns

This function returns the error code **MAX_ERR_NONE** if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.1.6.52 void* object_new_parse (t_symbol * *name_space*, t_symbol * *classname*, C74_CONST char * *parsestr*)

Create a new object with one or more atoms parsed from a C-string.

The object's new method must have an [A_GIMME](#) signature.

Parameters

- name_space* The namespace in which to create the instance. Typically this is either **CLASS_BOX** or **CLASS_NOBOX**.
- classname* The name of the class to instantiate.
- parsestr* A C-string to parse into an array of atoms to set the attribute value.

Returns

A pointer to the new instance.

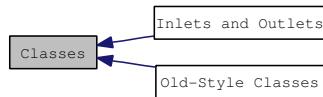
See also

- [atom_setparse\(\)](#)
- [object_new_typed\(\)](#)

33.2 Classes

When a user types the name of your object into an object box, Max looks for an external of this name in the searchpath and, upon finding it, loads the bundle or dll and calls the main() function.

Collaboration diagram for Classes:



Data Structures

- struct [t_class](#)

The data structure for a Max class.

Modules

- [Old-Style Classes](#)
- [Inlets and Outlets](#)

Routines for creating and communicating with inlets and outlets.

Defines

- #define [CLASS_BOX](#) gensym("box")

The namespace for all Max object classes which can be instantiated in a box, i.e.

- #define [CLASS_NOBOX](#) gensym("nobox")

A namespace for creating hidden or internal object classes which are not a direct part of the user creating patcher.

Enumerations

- enum [e_max_class_flags](#) {

 [CLASS_FLAG_BOX](#) = 0x00000001L,

 [CLASS_FLAG_POLYGLOT](#) = 0x00000002L,

 [CLASS_FLAG_NEWDICTIONARY](#) = 0x00000004L,

 [CLASS_FLAG_REGISTERED](#) = 0x00000008L,

 [CLASS_FLAG_UIOBJECT](#) = 0x00000010L,

 [CLASS_FLAG_ALIAS](#) = 0x00000020L,

 [CLASS_FLAG_SCHED_PURGE](#) = 0x00000040L,

 [CLASS_FLAG_DO_NOT_PARSE_ATTR_ARGS](#) = 0x00000080L,

 [CLASS_FLAG_NOATTRIBUTES](#) = 0x0010000L,

 [CLASS_FLAG_OWNATTRIBUTES](#) = 0x0020000L
 }

Class flags.

Functions

- **t_class * class_new** (C74_CONST char *name, C74_CONST method mnew, C74_CONST method mfree, long size, C74_CONST method mmenu, short type,...)
Initializes a class by informing Max of its name, instance creation and free functions, size and argument types.
- **t_max_err class_free (t_class *c)**
Frees a previously defined object class.
- **t_max_err class_register (t_symbol *name_space, t_class *c)**
Registers a previously defined object class.
- **t_max_err class_alias (t_class *c, t_symbol *aliasname)**
Registers an alias for a previously defined object class.
- **t_max_err class_addmethod (t_class *c, C74_CONST method m, C74_CONST char *name,...)**
Adds a method to a previously defined object class.
- **t_max_err class_addattr (t_class *c, t_object *attr)**
Adds an attribute to a previously defined object class.
- **t_symbol * class_nameget (t_class *c)**
Retrieves the name of a class, given the class's pointer.
- **t_class * class_findbyname (t_symbol *name_space, t_symbol *classname)**
Finds the class pointer for a class, given the class's namespace and name.
- **t_class * class_findbyname_casemode (t_symbol *name_space, t_symbol *classname)**
Finds the class pointer for a class, given the class's namespace and name.
- **t_max_err class_dumpout_wrap (t_class *c)**
Wraps user gettable attributes with a method that gets the values and sends out dumpout outlet.
- **void class_obexoffset_set (t_class *c, long offset)**
Registers the byte-offset of the obex member of the class's data structure with the previously defined object class.
- **long class_obexoffset_get (t_class *c)**
Retrieves the byte-offset of the obex member of the class's data structure.
- **long class_is_ui (t_class *c)**
Determine if a class is a user interface object.

33.2.1 Detailed Description

When a user types the name of your object into an object box, Max looks for an external of this name in the searchpath and, upon finding it, loads the bundle or dll and calls the main() function. Thus, Max classes are typically defined in the main() function of an external.

Historically, Max classes have been defined using an API that includes functions like `setup()` and `addmess()`. This interface is still supported, and the relevant documentation can be found in [Old-Style Classes](#).

A more recent and more flexible interface for creating objects was introduced with Jitter 1.0 and later included directly in Max 4.5. This newer API includes functions such as `class_new()` and `class_addmethod()`. Supporting attributes, user interface objects, and additional new features of Max requires the use of the newer interface for defining classes documented on this page.

You may not mix these two styles of creating classes within an object.

33.2.2 Define Documentation

33.2.2.1 #define CLASS_BOX gensym("box")

The namespace for all Max object classes which can be instantiated in a box, i.e. in a patcher.

33.2.3 Enumeration Type Documentation

33.2.3.1 enum e_max_class_flags

Class flags.

If not box or polyglot, class is only accessible in C via known interface

Enumerator:

CLASS_FLAG_BOX for use in a patcher

CLASS_FLAG_POLYGLOT for use by any text language (c/js/java/etc)

CLASS_FLAG_NEWDICTIONARY dictionary based constructor

CLASS_FLAG_REGISTERED for backward compatible messlist implementation (once reg'd can't grow)

CLASS_FLAG_UIOBJECT for objects that don't go inside a newobj box.

CLASS_FLAG_ALIAS for classes that are just copies of some other class (i.e. del is a copy of delay)

CLASS_FLAG_SCHED_PURGE for classes that have called `clock_new()` or `qelem_new()` (don't need to set this yourself)

CLASS_FLAG_DO_NOT_PARSE_ATTR_ARGS override dictionary based constructor attr arg parsing

CLASS_FLAG_NOATTRIBUTES for efficiency

CLASS_FLAG_OWNATTRIBUTES for classes which support a custom attr interface (e.g. jitter)

33.2.4 Function Documentation

33.2.4.1 `t_max_err class_addattr (t_class * c, t_object * attr)`

Adds an attribute to a previously defined object class.

Parameters

- c* The class pointer
- attr* The attribute to add. The attribute will be a pointer returned by `attribute_new()`, `attr_offset_new()` or `attr_offset_array_new()`.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.2.4.2 `t_max_err class_addmethod (t_class * c, C74_CONST method m, C74_CONST char * name, ...)`

Adds a method to a previously defined object class.

Parameters

- c* The class pointer
- m* Function to be called when the method is invoked
- name* C-string defining the message (message selector)
- ... One or more integers specifying the arguments to the message, in the standard Max type list format (see Chapter 3 of the Writing Externals in Max document for more information).

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

Remarks

The `class_addmethod()` function works essentially like the traditional `addmess()` function, adding the function pointed to by *m*, to respond to the message string *name* in the leftmost inlet of the object.

33.2.4.3 `t_max_err class_alias (t_class * c, t_symbol * aliasname)`

Registers an alias for a previously defined object class.

Parameters

- c* The class pointer
- aliasname* A symbol who's name will become an alias for the given class

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.2.4.4 `t_max_err class_dumpout_wrap (t_class * c)`

Wraps user gettable attributes with a method that gets the values and sends out dumpout outlet.

Parameters

c The class pointer

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.2.4.5 `t_class* class_findbyname (t_symbol * name_space, t_symbol * classname)`

Finds the class pointer for a class, given the class's namespace and name.

Parameters

name_space The desired class's name space. Typically, either the constant `CLASS_BOX`, for obex classes which can instantiate inside of a Max patcher (e.g. boxes, UI objects, etc.), or the constant `CLASS_NOBOX`, for classes which will only be used internally. Developers can define their own name spaces as well, but this functionality is currently undocumented.

classname The name of the class to be looked up

Returns

If successful, this function returns the class's data pointer. Otherwise, it returns NULL.

33.2.4.6 `t_class* class_findbyname_casefree (t_symbol * name_space, t_symbol * classname)`

Finds the class pointer for a class, given the class's namespace and name.

Parameters

name_space The desired class's name space. Typically, either the constant `CLASS_BOX`, for obex classes which can instantiate inside of a Max patcher (e.g. boxes, UI objects, etc.), or the constant `CLASS_NOBOX`, for classes which will only be used internally. Developers can define their own name spaces as well, but this functionality is currently undocumented.

classname The name of the class to be looked up (case free)

Returns

If successful, this function returns the class's data pointer. Otherwise, it returns NULL.

33.2.4.7 `t_max_err class_free (t_class * c)`

Frees a previously defined object class.

This function is not typically used by external developers.

Parameters

c The class pointer

Returns

This function returns the error code **MAX_ERR_NONE** if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.2.4.8 long class_is_ui (t_class * *c*)

Determine if a class is a user interface object.

Parameters

c The class pointer.

Returns

True is the class defines a user interface object, otherwise false.

33.2.4.9 t_symbol* class_nameget (t_class * *c*)

Retrieves the name of a class, given the class's pointer.

Parameters

c The class pointer

Returns

If successful, this function returns the name of the class as a [t_symbol](#) *.

**33.2.4.10 t_class* class_new (C74_CONST char * *name*, C74_CONST method *mnew*,
C74_CONST method *mfree*, long *size*, C74_CONST method *mmenu*, short *type*, ...)**

Initializes a class by informing Max of its name, instance creation and free functions, size and argument types.

Developers wishing to use obex class features (attributes, etc.) *must* use [class_new\(\)](#) instead of the traditional [setup\(\)](#) function.

Parameters

name The class's name, as a C-string

mnew The instance creation function

mfree The instance free function

size The size of the object's data structure in bytes. Usually you use the C sizeof operator here.

mmenu The function called when the user creates a new object of the class from the Patch window's palette (UI objects only). Pass 0L if you're not defining a UI object.

type A standard Max *type list* as explained in Chapter 3 of the Writing Externals in Max document (in the Max SDK). The final argument of the type list should be a 0. *Generally, obex objects have a single type argument, [A_GIMME](#), followed by a 0.*

Returns

This function returns the class pointer for the new object class. *This pointer is used by numerous other functions and should be stored in a global or static variable.*

33.2.4.11 long `class_obexoffset_get (t_class * c)`

Retrieves the byte-offset of the obex member of the class's data structure.

Parameters

c The class pointer

Returns

This function returns the byte-offset of the obex member of the class's data structure.

33.2.4.12 void `class_obexoffset_set (t_class * c, long offset)`

Registers the byte-offset of the obex member of the class's data structure with the previously defined object class.

Use of this function is required for obex-class objects. It must be called from `main()`.

Parameters

c The class pointer

offset The byte-offset to the obex member of the object's data structure. Conventionally, the macro `calcoffset` is used to calculate the offset.

33.2.4.13 t_max_err `class_register (t_symbol * name_space, t_class * c)`

Registers a previously defined object class.

This function is required, and should be called at the end of `main()`.

Parameters

name_space The desired class's name space. Typically, either the constant `CLASS_BOX`, for obex classes which can instantiate inside of a Max patcher (e.g. boxes, UI objects, etc.), or the constant `CLASS_NOBOX`, for classes which will only be used internally. Developers can define their own name spaces as well, but this functionality is currently undocumented.

c The class pointer

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.3 Old-Style Classes

Collaboration diagram for Old-Style Classes:



Functions

- void **setup** (*t_messlist **ident, method makefun, method freefun, short size, method menufun, short type,...)*

Use the [setup\(\)](#) function to initialize your class by informing Max of its size, the name of your functions that create and destroy instances, and the types of arguments passed to the instance creation function.

- void **addrmess** (*method f, char *s, short type,...)*

Use [addrmess\(\)](#) to bind a function to a message other than the standard ones covered by [addbang\(\)](#), [addint\(\)](#), etc.

- void **addbang** (*method f*)

Used to bind a function to the common triggering message bang.

- void **addint** (*method f*)

Use [addint\(\)](#) to bind a function to the int message received in the leftmost inlet.

- void **addfloat** (*method f*)

Use [addfloat\(\)](#) to bind a function to the float message received in the leftmost inlet.

- void **addinx** (*method f, short n*)

Use [addinx\(\)](#) to bind a function to a int message that will be received in an inlet other than the leftmost one.

- void **addftx** (*method f, short n*)

Use [addftx\(\)](#) to bind a function to a float message that will be received in an inlet other than the leftmost one.

- void * **newobject** (*void *maxclass*)

Use newobject to allocate the space for an instance of your class and initialize its object header.

- void **freeobject** (*t_object *op*)

Release the memory used by a Max object.

- void * **newinstance** (*t_symbol *s, short argc, t_atom *argv*)

Make a new instance of an existing Max class.

- void **alias** (*char *name*)

Use the alias function to allow users to refer to your object by a name other than that of your shared library.

- void **class_setname** (*char *obname, char *filename*)

Use [class_setname\(\)](#) to associate your object's name with its filename on disk.

- void * **typedmess** (*t_object *op, t_symbol *msg, short argc, t_atom *argp*)

Send a typed message directly to a Max object.

- method `getfn (t_object *op, t_symbol *msg)`

Use `getfn()` to send an untyped message to a Max object with error checking.

- method `egetfn (t_object *op, t_symbol *msg)`

Use `egetfn()` to send an untyped message to a Max object that always works.

- method `zgetfn (t_object *op, t_symbol *msg)`

Use `zgetfn()` to send an untyped message to a Max object without error checking.

33.3.1 Function Documentation

33.3.1.1 void addbang (method *f*)

Used to bind a function to the common triggering message bang.

Parameters

f Function to be the bang method.

33.3.1.2 void addfloat (method *f*)

Use `addfloat()` to bind a function to the float message received in the leftmost inlet.

Parameters

f Function to be the int method.

33.3.1.3 void addftx (method *f*, short *n*)

Use `addftx()` to bind a function to a float message that will be received in an inlet other than the leftmost one.

Parameters

f Function to be the float method.

n Number of the inlet connected to this method. 1 is the first inlet to the right of the left inlet.

Remarks

This correspondence between inlet locations and messages is not automatic, but it is strongly suggested that you follow existing practice. You must set the correspondence up when creating an object of your class with proper use of intin and floatin in your instance creation function [New Instance Routine](#).

33.3.1.4 void addint (method *f*)

Use [addint\(\)](#) to bind a function to the int message received in the leftmost inlet.

Parameters

f Function to be the int method.

33.3.1.5 void addinx (method *f*, short *n*)

Use [addinx\(\)](#) to bind a function to a int message that will be received in an inlet other than the leftmost one.

Parameters

f Function to be the int method.

n Number of the inlet connected to this method. 1 is the first inlet to the right of the left inlet.

Remarks

This correspondence between inlet locations and messages is not automatic, but it is strongly suggested that you follow existing practice. You must set the correspondence up when creating an object of your class with proper use of intin and floatin in your instance creation function [New Instance Routine](#).

33.3.1.6 void addmess (method *f*, char * *s*, short *type*, ...)

Use [addmess\(\)](#) to bind a function to a message other than the standard ones covered by [addbang\(\)](#), [addint\(\)](#), etc.

Parameters

f Function you want to be the method.

s C string defining the message.

type The first of one or more integers from [e_max_atomtypes](#) specifying the arguments to the message.

... Any additional types from [e_max_atomtypes](#) for additional arguments.

See also

[Anatomy of a Max Object](#)

33.3.1.7 void alias (char * *name*)

Use the alias function to allow users to refer to your object by a name other than that of your shared library.

Parameters

name An alternative name for the user to use to make an object of your class.

33.3.1.8 void class_setname (char * *obname*, char * *filename*)

Use [class_setname\(\)](#) to associate your object's name with its filename on disk.

Parameters

obname A character string with the name of your object class as it appears in Max.

filename A character string with the name of your external's file as it appears on disk.

33.3.1.9 method egetfn (t_object * *op*, t_symbol * *msg*)

Use [egetfn\(\)](#) to send an untyped message to a Max object that always works.

Parameters

op Receiver of the message.

msg Message selector.

Returns

egetfn returns a pointer to the method bound to the message selector msg in the receiver's message list. If the method can't be found, a pointer to a do-nothing function is returned.

33.3.1.10 void freeobject (t_object * *op*)

Release the memory used by a Max object.

[freeobject\(\)](#) calls an object's free function, if any, then disposes the memory used by the object itself. [freeobject\(\)](#) should be used on any instance of a standard Max object data structure, with the exception of Qelems and Atombufs. Clocks, Binbufs, Proxies, etc. should be freed with [freeobject\(\)](#).

Parameters

op The object instance pointer to free.

Remarks

This function can be replaced by the use of [object_free\(\)](#). Unlike [freeobject\(\)](#), [object_free\(\)](#) checks to make sure the pointer is not NULL before trying to free it.

See also

[newobject\(\)](#)

[object_free\(\)](#)

33.3.1.11 method getfn (t_object * *op*, t_symbol * *msg*)

Use [getfn\(\)](#) to send an untyped message to a Max object with error checking.

Parameters

op Receiver of the message.

msg Message selector.

Returns

getfn returns a pointer to the method bound to the message selector msg in the receiver's message list. It returns 0 and prints an error message in Max Window if the method can't be found.

33.3.1.12 void* newinstance (**t_symbol** * *s*, **short** *argc*, **t_atom** * *argv*)

Make a new instance of an existing Max class.

Parameters

s className Symbol specifying the name of the class of the instance to be created.

argc Count of arguments in argv.

argv Array of t_atoms; arguments to the class's instance creation function.

Returns

A pointer to the created object, or 0 if the class didn't exist or there was another type of error in creating the instance.

Remarks

This function creates a new instance of the specified class. Using newinstance is equivalent to typing something in a New Object box when using Max. The difference is that no object box is created in any Patcher window, and you can send messages to the object directly without connecting any patch cords. The messages can either be type- checked (using typedmess) or non-type-checked (using the members of the getfn family).

This function is useful for taking advantage of other already-defined objects that you would like to use 'privately' in your object, such as tables. See the source code for the coll object for an example of using a privately defined class.

33.3.1.13 void* newobject (**void** * *maxclass*)

Use newobject to allocate the space for an instance of your class and initialize its object header.

Parameters

maxclass The global class variable initialized in your main routine by the setup function.

Returns

A pointer to the new instance.

Remarks

You call [newobject\(\)](#) when creating an instance of your class in your creation function. newobject allocates the proper amount of memory for an object of your class and installs a pointer to your class in the object, so that it can respond with your class's methods if it receives a message.

33.3.1.14 void setup (t_messlist ***ident*, method *makefun*, method *freefun*, short *size*, method *menufun*, short *type*, ...)

Use the [setup\(\)](#) function to initialize your class by informing Max of its size, the name of your functions that create and destroy instances, and the types of arguments passed to the instance creation function.

Parameters

ident A global variable in your code that points to the initialized class.

makefun Your instance creation function.

freefun Your instance free function (see Chapter 7).

size The size of your objects data structure in bytes. Usually you use the C sizeof operator here.

menufun No longer used. You should pass NULL for this parameter.

type The first of a list of arguments passed to makefun when an object is created.

... Any additional arguments passed to makefun when an object is created. Together with the type parameter, this creates a standard Max type list as enumerated in [e_max_atomtypes](#). The final argument of the type list should be a 0.

See also

[Anatomy of a Max Object](#)

33.3.1.15 void* typedmess (t_object **op*, t_symbol **msg*, short *argc*, t_atom **argp*)

Send a typed message directly to a Max object.

Parameters

op Max object that will receive the message.

msg The message selector.

argc Count of message arguments in argv.

argp Array of t_atoms; the message arguments.

Returns

If the receiver object can respond to the message, [typedmess\(\)](#) returns the result. Otherwise, an error message will be seen in the Max window and 0 will be returned.

Remarks

typedmess sends a message to a Max object (receiver) a message with arguments. Note that the message must be a [t_symbol](#), not a character string, so you must call gensym on a string before passing it to typedmess. Also, note that untyped messages defined for classes with the argument list [A_CANT](#) cannot be sent using typedmess. You must use [getfn\(\)](#) etc. instead.

Example:

```
//If you want to send a bang message to the object bang_me...
void *bangResult;
bangResult = typedmess(bang_me, gensym("bang"), 0, 0L);
```

33.3.1.16 method **zgetfn** (*t_object * op, t_symbol * msg*)

Use [zgetfn\(\)](#) to send an untyped message to a Max object without error checking.

Parameters

op Receiver of the message.

msg Message selector.

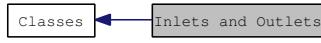
Returns

`zgetfn` returns a pointer to the method bound to the message selector `msg` in the receiver's message list. It returns 0 but doesn't print an error message in Max Window if the method can't be found.

33.4 Inlets and Outlets

Routines for creating and communicating with inlets and outlets.

Collaboration diagram for Inlets and Outlets:



Functions

- `void * inlet_new (void *x, C74_CONST char *s)`

Use [inlet_new\(\)](#) to create an inlet that can receive a specific message or any message.

- `void * intin (void *x, short n)`

Use [intin\(\)](#) to create an inlet typed to receive only integers.

- `void * floatin (void *x, short n)`

Use [floatin\(\)](#) to create an inlet typed to receive only floats.

- `void * outlet_new (void *x, C74_CONST char *s)`

Use [outlet_new\(\)](#) to create an outlet that can send a specific non-standard message, or any message.

- `void * bangout (void *x)`

Use [bangout\(\)](#) to create an outlet that will always send the bang message.

- `void * intout (void *x)`

Use [intout\(\)](#) to create an outlet that will always send the int message.

- `void * floatout (void *x)`

Use [floatout\(\)](#) to create an outlet that will always send the float message.

- `void * listout (void *x)`

Use [listout\(\)](#) to create an outlet that will always send the list message.

- `void * outlet_bang (void *o)`

Use [outlet_bang\(\)](#) to send a bang message out an outlet.

- `void * outlet_int (void *o, long n)`

Use [outlet_int\(\)](#) to send a float message out an outlet.

- `void * outlet_float (void *o, double f)`

Use [outlet_float\(\)](#) to send an int message out an outlet.

- `void * outlet_list (void *o, t_symbol *s, short ac, t_atom *av)`

Use [outlet_list\(\)](#) to send a list message out an outlet.

- `void * outlet_anything (void *o, t_symbol *s, short ac, t_atom *av)`

Use [outlet_anything\(\)](#) to send any message out an outlet.

- `void * proxy_new (void *x, long id, long *stuffloc)`

Use proxy_new to create a new Proxy object.

- `long proxy_getinlet (t_object *master)`

Use proxy_getinlet to get the inlet number in which a message was received.

33.4.1 Detailed Description

Routines for creating and communicating with inlets and outlets.

33.4.2 Function Documentation

33.4.2.1 void* bangout (void * x)

Use `bangout()` to create an outlet that will always send the bang message.

Parameters

x Your object.

Returns

A pointer to the new outlet.

Remarks

You can send a bang message out a general purpose outlet, but creating an outlet using `bangout()` allows Max to type-check the connection a user might make and refuse to connect the outlet to any object that cannot receive a bang message. `bangout()` returns the created outlet.

33.4.2.2 void* floatin (void * x, short n)

Use `floatin()` to create an inlet typed to receive only floats.

Parameters

x Your object.

n Location of the inlet from 1 to 9. 1 is immediately to the right of the leftmost inlet.

Returns

A pointer to the new inlet.

33.4.2.3 void* floatout (void * x)

Use `floatout()` to create an outlet that will always send the float message.

Parameters

x Your object.

Returns

A pointer to the new outlet.

33.4.2.4 void* inlet_new (void * x, C74_CONST char * s)

Use [inlet_new\(\)](#) to create an inlet that can receive a specific message or any message.

Parameters

x Your object.

s Character string of the message, or NULL to receive any message.

Returns

A pointer to the new inlet.

Remarks

[inlet_new\(\)](#) creates a general purpose inlet. You can use it in circumstances where you would like special messages to be received in inlets other than the leftmost one. To create an inlet that receives a particular message, pass the message's character string. For example, to create an inlet that receives only bang messages, do the following

```
inlet_new (myObject, "bang");
```

To create an inlet that can receive any message, pass NULL for msg

```
inlet_new (myObject, NULL);
```

Proxies are an alternative method for general-purpose inlets that have a number of advantages. If you create multiple inlets as shown above, there would be no way to figure out which inlet received a message. See the discussion in [Creating and Using Proxies](#).

33.4.2.5 void* intin (void * x, short n)

Use [intin\(\)](#) to create an inlet typed to receive only integers.

Parameters

x Your object.

n Location of the inlet from 1 to 9. 1 is immediately to the right of the leftmost inlet.

Returns

A pointer to the new inlet.

Remarks

[intin](#) creates integer inlets. It takes a pointer to your newly created object and an integer *n*, from 1 to 9. The number specifies the message type you'll get, so you can distinguish one inlet from another. For example, an integer sent in inlet 1 will be of message type in1 and a floating point number sent in inlet 4 will be of type ft4. You use [addinx\(\)](#) and [addftx\(\)](#) to add methods to respond to these messages.

The order you create additional inlets is important. If you want the rightmost inlet to be the highest number in- or ft- message (which is usually the case), you should create the highest number message inlet first.

33.4.2.6 void* intout (void * *x*)

Use [intout\(\)](#) to create an outlet that will always send the int message.

Parameters

x Your object.

Returns

A pointer to the new outlet.

Remarks

You can send a bang message out a general purpose outlet, but creating an outlet using [bangout\(\)](#) allows Max to type-check the connection a user might make and refuse to connect the outlet to any object that cannot receive a bang message. [bangout\(\)](#) returns the created outlet.

33.4.2.7 void* listout (void * *x*)

Use [listout\(\)](#) to create an outlet that will always send the list message.

Parameters

x Your object.

Returns

A pointer to the new outlet.

33.4.2.8 void* outlet_anything (void * *o*, t_symbol * *s*, short *ac*, t_atom * *av*)

Use [outlet_anything\(\)](#) to send any message out an outlet.

Parameters

o Outlet that will send the message.
s The message selector [t_symbol*](#).
ac Number of elements in the list in argv.
av Atoms constituting the list.

Returns

Returns 0 if a stack overflow occurred, otherwise returns 1.

Remarks

This function lets you send an arbitrary message out an outlet. Here are a couple of examples of its use.

First, here's a hard way to send the bang message (see [outlet_bang\(\)](#) for an easier way):

```
outlet_anything(myOutlet, gensym("bang"), 0, NIL);
```

Remarks

And here's an even harder way to send a single integer (instead of using `outlet_int()`).

```
t_atom myNumber;

atom_setlong(&myNumber, 432);
outlet_anything(myOutlet, gensym("int"), 1, &myNumber);
```

Notice that `outlet_anything()` expects the message argument as a `t_symbol*`, so you must use `gensym()` on a character string.

If you'll be sending the same message a lot, you might call `gensym()` on the message string at initialization time and store the result in a global variable to save the (significant) overhead of calling `gensym()` every time you want to send a message.

Also, do not send lists using `outlet_anything()` with list as the selector argument. Use the `outlet_list()` function instead.

33.4.2.9 void* outlet_bang (void * o)

Use `outlet_bang()` to send a bang message out an outlet.

Parameters

- o* Outlet that will send the message.

Returns

Returns 0 if a stack overflow occurred, otherwise returns 1.

33.4.2.10 void* outlet_float (void * o, double f)

Use `outlet_float()` to send an int message out an outlet.

Parameters

- o* Outlet that will send the message.
- f* Float value to send.

Returns

Returns 0 if a stack overflow occurred, otherwise returns 1.

33.4.2.11 void* outlet_int (void * o, long n)

Use `outlet_int()` to send a float message out an outlet.

Parameters

- o* Outlet that will send the message.
- n* Integer value to send.

Returns

Returns 0 if a stack overflow occurred, otherwise returns 1.

33.4.2.12 void* outlet_list (void * o, t_symbol * s, short ac, t_atom * av)

Use [outlet_list\(\)](#) to send a list message out an outlet.

Parameters

- o* Outlet that will send the message.
- s* Should be NULL, but can be the _sym_list.
- ac* Number of elements in the list in argv.
- av* Atoms constituting the list.

Returns

Returns 0 if a stack overflow occurred, otherwise returns 1.

Remarks

[outlet_list\(\)](#) sends the list specified by argv and argc out the specified outlet. The outlet must have been created with listout or outlet_new in your object creation function (see above). You create the list as an array of Atoms, but the first item in the list must be an integer or float.

Here's an example of sending a list of three numbers.

```
t_atom myList[3];
long theNumbers[3];
short i;

theNumbers[0] = 23;
theNumbers[1] = 12;
theNumbers[2] = 5;
for (i=0; i < 3; i++) {
    atom_setlong(myList+i, theNumbers[i]);
}
outlet_list(myOutlet, 0L, 3, &myList);
```

Remarks

It's not a good idea to pass large lists to outlet_list that are comprised of local (automatic) variables. If the list is small, as in the above example, there's no problem. If your object will regularly send lists, it might make sense to keep an array of t_atoms inside your object's data structure.

33.4.2.13 void* outlet_new (void * x, C74_CONST char * s)

Use [outlet_new\(\)](#) to create an outlet that can send a specific non-standard message, or any message.

Parameters

- x* Your object.
- s* A C-string specifying the message that will be sent out this outlet, or NULL to indicate the outlet will be used to send various messages. The advantage of this kind of outlet's flexibility is balanced by the fact that Max must perform a message-lookup in real-time for every message sent through it, rather than when a patch is being constructed, as is true for other types of outlets. Patchers execute faster when outlets are typed, since the message lookup can be done before the program executes.

Returns

A pointer to the new outlet.

33.4.2.14 long proxy_getinlet (*t_object * master*)

Use proxy_getinlet to get the inlet number in which a message was received.

Note that the `owner` argument should point to your external object's instance, not a proxy object.

Parameters

master Your object.

Returns

The index number of the inlet that received the message.

33.4.2.15 void* proxy_new (*void * x, long id, long * stuffloc*)

Use proxy_new to create a new Proxy object.

Parameters

x Your object.

id A non-zero number to be written into your object when a message is received in this particular Proxy. Normally, id will be the inlet number analogous to in1, in2 etc.

stuffloc A pointer to a location where the id value will be written.

Returns

A pointer to the new proxy inlet.

Remarks

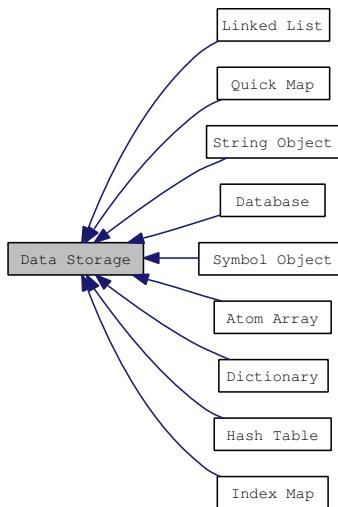
This routine creates a new Proxy object (that includes an inlet). It allows you to identify messages based on an id value stored in the location specified by stuffLoc. You should store the pointer returned by [proxy_new\(\)](#) because you'll need to free all Proxies in your object's free function using [object_free\(\)](#).

After your method has finished, Proxy sets the stuffLoc location back to 0, since it never sees messages coming in an object's leftmost inlet. You'll know you received a message in the leftmost inlet if the contents of stuffLoc is 0. As of Max 4.3, stuffLoc is not always guaranteed to be a correct indicator of the inlet in which a message was received. Use [proxy_getinlet\(\)](#) to determine the inlet number.

33.5 Data Storage

Max provides a number of ways of storing and manipulating data at a high level.

Collaboration diagram for Data Storage:



Modules

- [Atom Array](#)

Max's atomarray object is a container for an array of atoms with an interface for manipulating that array.

- [Database](#)

Max's database support currently consists of a SQLite (<http://sqlite.org>) extension which is loaded dynamically by Max at launch time.

- [Dictionary](#)

In Max 5, we have a new "dictionary" object which can be used for object prototypes, object serialization, object constructors, and other tasks.

- [Hash Table](#)

Max's hashtable object implements a hash table (http://en.wikipedia.org/wiki/Hash_table).

- [Index Map](#)

An indexmap is basically a managed array of pointers, but it allows you to derive relatively quickly the index from a pointer in the array.

- [Linked List](#)

Max's linklist object implements a doubly-linked-list (http://en.wikipedia.org/wiki/Linked_list) together with a high-level interface for manipulating and accessing values in the list.

- [Quick Map](#)

A quickmap implements a pair of `t_hashtable` hash tables so that it is fast to look up a unique value for a unique key or vice-versa.

- **String Object**

Max's string object is a simple wrapper for c-strings, useful when working with Max's [t_dictionary](#), [t_linklist](#), or [t_hashtab](#).

- **Symbol Object**

The symobject class is a simple object that wraps a [t_symbol](#) together with a couple of additional fields.*

Typedefs

- **typedef long(* t_cmpfn)(void *, void *)**

Comparison function pointer type.

Enumerations

- **enum e_max_datastore_flags {**

```
OBJ_FLAG_OBJ = 0x00000000,
OBJ_FLAG_REF = 0x00000001,
OBJ_FLAG_DATA = 0x00000002,
OBJ_FLAG_MEMORY = 0x00000004,
OBJ_FLAG_SILENT = 0x00000100,
OBJ_FLAG_INHERITABLE = 0x00000200 }
```

Flags used in linklist and hashtab objects.

33.5.1 Detailed Description

Max provides a number of ways of storing and manipulating data at a high level. It is recommended to use Max's data storage mechanisms where possible, as Max's systems are designed for thread-safety and integration with the rest of Max API.

33.5.2 Typedef Documentation

33.5.2.1 **typedef long(* t_cmpfn)(void *, void *)**

Comparison function pointer type.

Methods that require a comparison function pointer to be passed in use this type. It should return `true` or `false` depending on the outcome of the comparison of the two linklist items passed in as arguments.

See also

[linklist_match\(\)](#)
[hashtab_findfirst\(\)](#)
[indexmap_sort\(\)](#)

33.5.3 Enumeration Type Documentation

33.5.3.1 enum e_max_datastore_flags

Flags used in linklist and hashtab objects.

Enumerator:

OBJ_FLAG_OBJ free using `object_free()`

OBJ_FLAG_REF don't free

OBJ_FLAG_DATA don't free data or call method

OBJ_FLAG_MEMORY don't call method, and when freeing use `sysmem_freeptr()` instead of `freeobject`

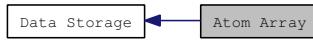
OBJ_FLAG_SILENT don't notify when modified

OBJ_FLAG_INHERITABLE obexprototype entry will be inherited by subpatchers and abstractions

33.6 Atom Array

Max's atomarray object is a container for an array of atoms with an interface for manipulating that array.

Collaboration diagram for Atom Array:



Data Structures

- struct `t_atomarray`

The atomarray object.

Defines

- #define `ATOMARRAY_FLAG_FREECHILDREN` (1)

The atomarray flags.

Functions

- `t_atomarray * atomarray_new (long ac, t_atom *av)`

Create a new atomarray object.

- `void atomarray_flags (t_atomarray *x, long flags)`

Set the atomarray flags.

- `long atomarray_getflags (t_atomarray *x)`

Get the atomarray flags.

- `t_max_err atomarray_setatoms (t_atomarray *x, long ac, t_atom *av)`

Replace the existing array contents with a new set of atoms Note that atoms provided to this function will be copied.

- `t_max_err atomarray_getatoms (t_atomarray *x, long *ac, t_atom **av)`

Retrieve a pointer to the first atom in the internal array of atoms.

- `t_max_err atomarray_copyatoms (t_atomarray *x, long *ac, t_atom **av)`

Retrieve a copy of the atoms in the array.

- `long atomarray_getsize (t_atomarray *x)`

Return the number of atoms in the array.

- `t_max_err atomarray_getindex (t_atomarray *x, long index, t_atom *av)`

Copy an a specific atom from the array.

- `void * atomarray_duplicate (t_atomarray *x)`

Create a new atomarray object which is a copy of another atomarray object.

- **void atomarray_appendatom (*t_atomarray* **x*, *t_atom* **a*)**
Copy a new atom onto the end of the array.
- **void atomarray_appendatoms (*t_atomarray* **x*, long ac, *t_atom* **av*)**
Copy multiple new atoms onto the end of the array.
- **void atomarray_chuckindex (*t_atomarray* **x*, long index)**
Remove an atom from any location within the array.
- **void atomarray_clear (*t_atomarray* **x*)**
Clear the array.
- **void atomarray_funall (*t_atomarray* **x*, method fun, void *arg)**
Call the specified function for every item in the atom array.

33.6.1 Detailed Description

Max's atomarray object is a container for an array of atoms with an interface for manipulating that array. It can be useful for passing lists as a single atom, such as for the return value of an [A_GIMMEBACK](#) method. It also used frequently in when working with Max's [t_dictionary](#) object.

See also

[Dictionary](#)

33.6.2 Define Documentation

33.6.2.1 #define ATOMARRAY_FLAG_FREECHILDREN (1)

The atomarray flags.

Currently the only flag is ATOMARRAY_FLAG_FREECHILDREN. If set via [atomarray_flags\(\)](#) the atomarray will free any contained A_OBJ atoms when the atomarray is freed.

33.6.3 Function Documentation

33.6.3.1 void atomarray_appendatom (*t_atomarray* **x*, *t_atom* **a*)

Copy a new atom onto the end of the array.

Parameters

x The atomarray instance.

a A pointer to the new atom to append to the end of the array.

See also

[atomarray_appendatoms\(\)](#)
[atomarray_setatoms\(\)](#)

33.6.3.2 void atomarray_appendatoms (*t_atomarray *x*, *long ac*, *t_atom *av*)

Copy multiple new atoms onto the end of the array.

Parameters

- x* The atomarray instance.
- ac* The number of new atoms to be appended to the array.
- av* A pointer to the first of the new atoms to append to the end of the array.

See also

[atomarray_appendatom\(\)](#)
[atomarray_setatoms\(\)](#)

33.6.3.3 void atomarray_chuckindex (*t_atomarray *x*, *long index*)

Remove an atom from any location within the array.

The array will be resized and collapsed to fill in the gap.

Parameters

- x* The atomarray instance.
- index* The zero-based index of the atom to remove from the array.

33.6.3.4 void atomarray_clear (*t_atomarray *x*)

Clear the array.

Frees all of the atoms and sets the size to zero. This function does not perform a 'deep' free, meaning that any [A_OBJ](#) atoms will not have their object's freed. Only the references to those objects contained in the atomarray will be freed.

Parameters

- x* The atomarray instance.

Returns

The number of atoms in the array.

33.6.3.5 t_max_err atomarray_copyatoms (*t_atomarray *x*, *long *ac*, *t_atom **av*)

Retrieve a copy of the atoms in the array.

This method does not copy the atoms, but simply provides access to them. To retrieve a copy of the atoms use [atomarray_copyatoms\(\)](#).

Parameters

- x* The atomarray instance.

ac The address of a long where the number of atoms will be set.

av The address of a `t_atom` pointer where the atoms will be allocated and copied.

Returns

A Max error code.

Remarks

You are responsible for freeing memory allocated for the copy of the atoms returned.

```
long ac = 0;
t_atom *av = NULL;

atomarray_copyatoms(anAtomarray, &ac, &av);
if(ac && av) {
    // do something with ac and av here...
    sysmem_freeptr(av);
}
```

See also

[atomarray_getatoms\(\)](#)

33.6.3.6 void* atomarray_duplicate (`t_atomarray *x`)

Create a new atomarray object which is a copy of another atomarray object.

Parameters

x The atomarray instance which is to be copied.

Returns

A new atomarray which is copied from x.

See also

[atomarray_new\(\)](#)

33.6.3.7 void atomarray_flags (`t_atomarray *x, long flags`)

Set the atomarray flags.

Parameters

x The atomarray instance.

flags The new value for the flags.

33.6.3.8 void atomarray_funall (`t_atomarray *x, method fun, void *arg`)

Call the specified function for every item in the atom array.

Parameters

- x* The atomarray instance.
- fun* The function to call, specified as function pointer cast to a Max [method](#).
- arg* An argument that you would like to pass to the function being called.

Returns

A max error code.

Remarks

The [atomarray_funall\(\)](#) method will call your function for every item in the list. It will pass both a pointer to the item in the list, and any argument that you provide. The following example shows a function that could be called by [hashtab_funall\(\)](#).

```
void myFun(t_atom *a, void *myArg)
{
    // do something with a and myArg here
    // a is the atom in the atom array
}
```

See also

[linklist_funall\(\)](#)
[hashtab_funall\(\)](#)

33.6.3.9 t_max_err atomarray_getatoms (t_atomarray *x, long *ac, t_atom **av)

Retrieve a pointer to the first atom in the internal array of atoms.

This method does not copy the atoms, but simply provides access to them. To retrieve a copy of the atoms use [atomarray_copyatoms\(\)](#).

Parameters

- x* The atomarray instance.
- ac* The address of a long where the number of atoms will be set.
- av* The address of a [t_atom](#) pointer where the address of the first atom of the array will be set.

Returns

A Max error code.

See also

[atomarray_copyatoms\(\)](#)

33.6.3.10 long atomarray_getflags (t_atomarray *x)

Get the atomarray flags.

Parameters

- x* The atomarray instance.

Returns

The current value of the atomarray flags.

33.6.3.11 t_max_err atomarray_getindex (t_atomarray *x, long index, t_atom *av)

Copy an a specific atom from the array.

Parameters

x The atomarray instance.

index The zero-based index into the array from which to retrieve an atom pointer.

av The address of an atom to contain the copy.

Returns

A Max error code.

Remarks

Example:

```
{  
    t_atom a;  
  
    // fetch a copy of the second atom in a previously existing array  
    atomarray_getindex(anAtomarray, 1, &a);  
    // do something with the atom here...  
}
```

33.6.3.12 long atomarray_getsize (t_atomarray *x)

Return the number of atoms in the array.

Parameters

x The atomarray instance.

Returns

The number of atoms in the array.

33.6.3.13 t_atomarray* atomarray_new (long ac, t_atom *av)

Create a new atomarray object.

Note that atoms provided to this function will be *copied*. The copies stored internally to the atomarray instance. You can free the atomarray by calling [object_free\(\)](#).

Parameters

ac The number of atoms to be initially contained in the atomarray.

av A pointer to the first of an array of atoms to initially copy into the atomarray.

Returns

Pointer to the new atomarray object.

Remarks

Note that due to the unusual prototype of this method that you cannot instantiate this object using the [object_new_typed\(\)](#) function. If you wish to use the dynamically bound creator to instantiate the object, you should instead use [object_new\(\)](#) as demonstrated below. The primary reason that you might choose to instantiate an atomarray using [object_new\(\)](#) instead of [atomarray_new\(\)](#) is for using the atomarray object in code that is also intended to run in Max 4.

```
object_new(CLASS_NOBOX, gensym("atomarray"), argc, argv);
```

See also

[atomarray_duplicate\(\)](#)

33.6.3.14 **t_max_err atomarray_setatoms (t_atomarray *x, long ac, t_atom *av)**

Replace the existing array contents with a new set of atoms Note that atoms provided to this function will be *copied*.

The copies stored internally to the atomarray instance.

Parameters

x The atomarray instance.

ac The number of atoms to be initially contained in the atomarray.

av A pointer to the first of an array of atoms to initially copy into the atomarray.

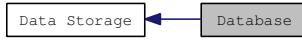
Returns

A Max error code.

33.7 Database

Max's database support currently consists of a SQLite (<http://sqlite.org>) extension which is loaded dynamically by Max at launch time.

Collaboration diagram for Database:



Typedefs

- **typedef t_object t_database**
A database object.
- **typedef t_object t_db_result**
A database result object.
- **typedef t_object t_db_view**
A database view object.

Functions

- **BEGIN_USING_C_LINKAGE t_max_err db_open (t_symbol *dbname, const char *fullpath, t_database **db)**
Create an instance of a database.
- **t_max_err db_close (t_database **db)**
Close an open database.
- **t_max_err db_query (t_database *db, t_db_result **dbresult, const char *sql,...)**
Execute a SQL query on the database.
- **t_max_err db_query_silent (t_database *db, t_db_result **dbresult, const char *sql,...)**
Execute a SQL query on the database, temporarily overriding the database's error logging attribute.
- **t_max_err db_query_getlastinsertid (t_database *db, long *id)**
Determine the id (key) number for the most recent INSERT query executed on the database.
- **t_max_err db_query_table_new (t_database *db, const char *tablename)**
Create a new table in a database.
- **t_max_err db_query_table_addcolumn (t_database *db, const char *tablename, const char *columnname, const char *columntype, const char *flags)**
Add a new column to an existing table in a database.
- **t_max_err db_transaction_start (t_database *db)**
Begin a database transaction.

- **t_max_err db_transaction_end (t_database *db)**
Finalize a database transaction.
- **t_max_err db_transaction_flush (t_database *db)**
Force any open transactions to close.
- **t_max_err db_view_create (t_database *db, const char *sql, t_db_view **dbview)**
A database view is a way of looking at a particular set of records in the database.
- **t_max_err db_view_remove (t_database *db, t_db_view **dbview)**
Remove a database view created using `db_view_create()`.
- **t_max_err db_view_getresult (t_db_view *dbview, t_db_result **result)**
Fetch the pointer for a `t_db_view`'s query result.
- **t_max_err db_view_setquery (t_db_view *dbview, char *newquery)**
Set the query used by the view.
- **char ** db_result_nextrecord (t_db_result *result)**
Return the next record from a set of results that you are walking.
- **void db_result_reset (t_db_result *result)**
Reset the interface for walking a result's record list to the first record.
- **void db_result_clear (t_db_result *result)**
Zero-out a database result.
- **long db_result_numrecords (t_db_result *result)**
Return a count of all records in the query result.
- **long db_result_numfields (t_db_result *result)**
Return a count of all fields (columns) in the query result.
- **char * db_result_fieldname (t_db_result *result, long fieldindex)**
Return the name of a field specified by its index number.
- **char * db_result_string (t_db_result *result, long recordindex, long fieldindex)**
Return a single value from a result according to its index and field coordinates.
- **long db_result_long (t_db_result *result, long recordindex, long fieldindex)**
Return a single value from a result according to its index and field coordinates.
- **float db_result_float (t_db_result *result, long recordindex, long fieldindex)**
Return a single value from a result according to its index and field coordinates.
- **unsigned long db_result_datetimeinseconds (t_db_result *result, long recordindex, long fieldindex)**
Return a single value from a result according to its index and field coordinates.

- void **db_util_stringtodate** (const char *string, unsigned long *date)

A utility to convert from a sql datetime string into seconds.

- void **db_util_datetostring** (const unsigned long date, char *string)

A utility to convert from seconds into a sql-ready datetime string.

33.7.1 Detailed Description

Max's database support currently consists of a SQLite (<http://sqlite.org>) extension which is loaded dynamically by Max at launch time. Because it is loaded dynamically, all interfacing with the sqlite object relies on Max's message passing interface, using **object_method()** and related functions.

For most common database needs, a C-interface is defined in the ext_database.h header file and implemented in the ext_database.c source file. The functions defined in this interface wrap the message passing calls and provide a convenient means by which you can work with databases. ext_database.c is located in the 'common' folder inside of the 'max-includes' folder. If you use any of the functions defined ext_database.h, you will need to add ext_database.c to your project.

33.7.2 Typedef Documentation

33.7.2.1 **typedef t_object t_database**

A database object.

Use **db_open()** and **db_close()** to create and free database objects.

33.7.2.2 **typedef t_object t_db_result**

A database result object.

This is what the database object returns when a query is executed.

33.7.2.3 **typedef t_object t_db_view**

A database view object.

A database view wraps a query and a result for a given database, and is always updated and in-sync with the database.

33.7.3 Function Documentation

33.7.3.1 **t_max_err db_close (t_database ** db)**

Close an open database.

Parameters

db The address of the **t_database** pointer for your database instance. The pointer will be freed and set NULL upon return.

Returns

An error code.

33.7.3.2 BEGIN_USING_C_LINKAGE *t_max_err db_open (t_symbol * dbname, const char * fullpath, t_database ** db)*

Create an instance of a database.

Parameters

dbname The name of the database.

fullpath If a database with this dbname is not already open, this will specify a full path to the location where the database is stored on disk. If NULL is passed for this argument, the database will reside in memory only. The path should be formatted as a Max style path.

db The address of a [t_database](#) pointer that will be set to point to the new database instance. If the pointer is not NULL, then it will be treated as a pre-existing database instance and thus will be freed.

Returns

An error code.

33.7.3.3 *t_max_err db_query (t_database * db, t_db_result ** dbresult, const char * sql, ...)*

Execute a SQL query on the database.

Parameters

db The [t_database](#) pointer for your database instance.

dbresult The address of a [t_db_result](#) pointer. If the pointer is passed-in set to NULL then a new dbresult will be created. If the pointer is not NULL then it is assumed to be a valid dbresult, which will be filled in with the query results. When you are done with the dbresult you should free it with [object_free\(\)](#).

sql A C-string containing a valid SQL query, possibly with sprintf() formatting codes.

... If an sprintf() formatting codes are used in the sql string, these values will be interpolated into the sql string.

Returns

An error code.

33.7.3.4 *t_max_err db_query_getlastinsertid (t_database * db, long * id)*

Determine the id (key) number for the most recent INSERT query executed on the database.

Parameters

db The [t_database](#) pointer for your database instance.

id The address of a variable to hold the result on return.

Returns

An error code.

33.7.3.5 t_max_err db_query_silent (t_database *db, t_db_result **dbresult, const char *sql, ...)

Execute a SQL query on the database, temporarily overriding the database's error logging attribute.

Parameters

db The [t_database](#) pointer for your database instance.

dbresult The address of a [t_db_result](#) pointer. If the pointer is passed-in set to NULL then a new dbresult will be created. If the pointer is not NULL then it is assumed to be a valid dbresult, which will be filled in with the query results. When you are done with the dbresult you should free it with [object_free\(\)](#).

sql A C-string containing a valid SQL query, possibly with sprintf() formatting codes.

... If an sprintf() formatting codes are used in the sql string, these values will be interpolated into the sql string.

Returns

An error code.

33.7.3.6 t_max_err db_query_table_adddcolumn (t_database *db, const char *tablename, const char *columnname, const char *columntype, const char *flags)

Add a new column to an existing table in a database.

Parameters

db The [t_database](#) pointer for your database instance.

tablename The name of the table to which the column should be added.

columnname The name to use for the new column.

columntype The SQL type for the data that will be stored in the column. For example: "INTEGER" or "VARCHAR"

flags If you wish to specify any additional information for the column, then pass that here. Otherwise pass NULL.

Returns

An error code.

33.7.3.7 t_max_err db_query_table_new (t_database *db, const char *tablename)

Create a new table in a database.

Parameters

db The [t_database](#) pointer for your database instance.

tablename The name to use for the new table. The new table will be created with one column, which holds the primary key for the table, and is named according the form {tablename}_id.

Returns

An error code.

33.7.3.8 void db_result_clear (*t_db_result* * *result*)

Zero-out a database result.

Parameters

result The [t_db_result](#) pointer for your query results.

33.7.3.9 unsigned long db_result_datetimeminseconds (*t_db_result* * *result*, *long recordindex*, *long fieldindex*)

Return a single value from a result according to its index and field coordinates.

The value will be coerced from an expected datetime field into seconds.

Parameters

result The [t_db_result](#) pointer for your query results.

recordindex The zero-based index number of the record (row) in the result.

fieldindex The zero-based index number of the field (column) in the result.

Returns

The datetime represented in seconds.

33.7.3.10 char* db_result_fieldname (*t_db_result* * *result*, *long fieldindex*)

Return the name of a field specified by its index number.

Parameters

result The [t_db_result](#) pointer for your query results.

fieldindex The zero-based index number of the field (column) in the result.

Returns

A C-String with the name of the field.

33.7.3.11 float db_result_float (*t_db_result* * *result*, *long recordindex*, *long fieldindex*)

Return a single value from a result according to its index and field coordinates.

Parameters

result The [t_db_result](#) pointer for your query results.

recordindex The zero-based index number of the record (row) in the result.

fieldindex The zero-based index number of the field (column) in the result.

Returns

The content of the specified cell from the result scanned out to a float.

33.7.3.12 long db_result_long (t_db_result * *result*, long *recordindex*, long *fieldindex*)

Return a single value from a result according to its index and field coordinates.

Parameters

result The [t_db_result](#) pointer for your query results.

recordindex The zero-based index number of the record (row) in the result.

fieldindex The zero-based index number of the field (column) in the result.

Returns

The content of the specified cell from the result scanned out to a long int.

33.7.3.13 char db_result_nextrecord (t_db_result * *result*)**

Return the next record from a set of results that you are walking.

When you are returned a result from a query of the database, the result is prepared for walking the results from the beginning. You can also reset the result manually to the beginning of the record list by calling [db_result_reset\(\)](#).

Parameters

result The [t_db_result](#) pointer for your query results.

Returns

An array of C-Strings with the values for every requested column (field) of a database record. To find out how many columns are represented in the array, use [db_result_numfields\(\)](#).

33.7.3.14 long db_result_numfields (t_db_result * *result*)

Return a count of all fields (columns) in the query result.

Parameters

result The [t_db_result](#) pointer for your query results.

Returns

The count of fields in the query result.

33.7.3.15 long db_result_numrecords (t_db_result * *result*)

Return a count of all records in the query result.

Parameters

result The [t_db_result](#) pointer for your query results.

Returns

The count of records in the query result.

33.7.3.16 void db_result_reset (*t_db_result* * *result*)

Reset the interface for walking a result's record list to the first record.

Parameters

result The [t_db_result](#) pointer for your query results.

33.7.3.17 char* db_result_string (*t_db_result* * *result*, long *recordindex*, long *fieldindex*)

Return a single value from a result according to its index and field coordinates.

Parameters

result The [t_db_result](#) pointer for your query results.

recordindex The zero-based index number of the record (row) in the result.

fieldindex The zero-based index number of the field (column) in the result.

Returns

A C-String with the content of the specified cell in the result.

33.7.3.18 t_max_err db_transaction_end (*t_database* * *db*)

Finalize a database transaction.

Parameters

db The [t_database](#) pointer for your database instance.

Returns

An error code.

33.7.3.19 t_max_err db_transaction_flush (*t_database* * *db*)

Force any open transactions to close.

Parameters

db The [t_database](#) pointer for your database instance.

Returns

An error code.

33.7.3.20 t_max_err db_transaction_start (t_database * db)

Begin a database transaction.

When you are working with a file-based database, then the database will not be flushed to disk until db_transacation_end() is called. This means that you can _much_ more efficiently execute a sequence of queries in one transaction rather than independently.

That database object reference counts transactions, so it is possible nest calls to db_transacation_start() and db_transacation_end(). It is important to balance all calls with db_transacation_end() or the database contents will never be flushed to disk.

Parameters

db The [t_database](#) pointer for your database instance.

Returns

An error code.

33.7.3.21 void db_util_datetostring (const unsigned long date, char * string)

A utility to convert from seconds into a sql-ready datetime string.

Parameters

date The datetime represented in seconds.

string The address of a valid C-string whose contents will be set to a SQL-ready string format upon return.

33.7.3.22 void db_util_stringtodata (const char * string, unsigned long * date)

A utility to convert from a sql datetime string into seconds.

Parameters

string A C-string containing a date and time in SQL format.

date The datetime represented in seconds upon return.

33.7.3.23 t_max_err db_view_create (t_database * db, const char * sql, t_db_view ** dbview)

A database view is a way of looking at a particular set of records in the database.

This particular set of records is defined with a standard SQL query, and the view maintains a copy of the results of the query internally. Any time the database is modified the internal result set is updated, and any objects listening to the view are notified via [object_notify\(\)](#).

Parameters

db The [t_database](#) pointer for your database instance.

sql A SQL query that defines the set of results provided by the view.

dbview The address of a NULL [t_db_view](#) pointer which will be set with the new view upon return.

Returns

An error code.

33.7.3.24 t_max_err db_view_getresult (t_db_view * *dbview*, t_db_result ** *result*)

Fetch the pointer for a [t_db_view](#)'s query result.

Parameters

dbview The [t_db_view](#) pointer for your database view instance.

result The address of a pointer to a [t_db_result](#) object. This pointer will be overwritten with the view's result pointer upon return.

Returns

An error code.

33.7.3.25 t_max_err db_view_remove (t_database * *db*, t_db_view ** *dbview*)

Remove a database view created using [db_view_create\(\)](#).

Parameters

db The [t_database](#) pointer for your database instance for which this view was created.

dbview The address of the [t_db_view](#) pointer for the view. This pointer will be freed and set NULL upon return.

Returns

An error code.

33.7.3.26 t_max_err db_view_setquery (t_db_view * *dbview*, char * *newquery*)

Set the query used by the view.

Parameters

dbview The [t_db_view](#) pointer for your database view instance.

newquery The SQL string to define a new query for the view, replacing the old query.

Returns

An error code.

33.8 Dictionary

In Max 5, we have a new "dictionary" object which can be used for object prototypes, object serialization, object constructors, and other tasks.

Collaboration diagram for Dictionary:



Data Structures

- struct `t_dictionary_entry`

A dictionary entry.

- struct `t_dictionary`

The dictionary object.

Functions

- `t_dictionary * dictionary_new ()`

Create a new linklist object.

- `t_max_err dictionary_appendlong (t_dictionary *d, t_symbol *key, long value)`

Add a long integer value to the dictionary.

- `t_max_err dictionary_appendfloat (t_dictionary *d, t_symbol *key, double value)`

Add a double-precision float value to the dictionary.

- `t_max_err dictionary_appendsym (t_dictionary *d, t_symbol *key, t_symbol *value)`

Add a `t_symbol` value to the dictionary.*

- `t_max_err dictionary_appendatom (t_dictionary *d, t_symbol *key, t_atom *value)`

Add a `t_atom` value to the dictionary.*

- `t_max_err dictionary_appendstring (t_dictionary *d, t_symbol *key, const char *value)`

Add a C-string to the dictionary.

- `t_max_err dictionary_appendatoms (t_dictionary *d, t_symbol *key, long argc, t_atom *argv)`

Add an array of atoms to the dictionary.

- `t_max_err dictionary_appendatomarray (t_dictionary *d, t_symbol *key, t_object *value)`

Add an `Atom Array` object to the dictionary.

- `t_max_err dictionary_appenddictionary (t_dictionary *d, t_symbol *key, t_object *value)`

Add a dictionary object to the dictionary.

- `t_max_err dictionary_appendobject (t_dictionary *d, t_symbol *key, t_object *value)`

Add an object to the dictionary.

- **t_max_err dictionary_getlong (t_dictionary *d, t_symbol *key, long *value)**
Retrieve a long integer from the dictionary.
- **t_max_err dictionary_getfloat (t_dictionary *d, t_symbol *key, double *value)**
Retrieve a double-precision float from the dictionary.
- **t_max_err dictionary_getsym (t_dictionary *d, t_symbol *key, t_symbol **value)**
Retrieve a t_symbol from the dictionary.*
- **t_max_err dictionary_getatom (t_dictionary *d, t_symbol *key, t_atom *value)**
Copy a t_atom from the dictionary.
- **t_max_err dictionary_getstring (t_dictionary *d, t_symbol *key, const char **value)**
Retrieve a C-string pointer from the dictionary.
- **t_max_err dictionary_getatoms (t_dictionary *d, t_symbol *key, long *argc, t_atom **argv)**
Retrieve the address of a t_atom array of in the dictionary.
- **t_max_err dictionary_copyatoms (t_dictionary *d, t_symbol *key, long *argc, t_atom **argv)**
Retrieve copies of a t_atom array in the dictionary.
- **t_max_err dictionary_getatomarray (t_dictionary *d, t_symbol *key, t_object **value)**
Retrieve a t_atomarray pointer from the dictionary.
- **t_max_err dictionary_getdictionary (t_dictionary *d, t_symbol *key, t_object **value)**
Retrieve a t_dictionary pointer from the dictionary.
- **t_max_err dictionary_getobject (t_dictionary *d, t_symbol *key, t_object **value)**
Retrieve a t_object pointer from the dictionary.
- **long dictionary_entryisstring (t_dictionary *d, t_symbol *key)**
Test a key to set if the data stored with that key contains a t_string object.
- **long dictionary_entryisatomarray (t_dictionary *d, t_symbol *key)**
Test a key to set if the data stored with that key contains a t_atomarray object.
- **long dictionary_entryisdictionary (t_dictionary *d, t_symbol *key)**
Test a key to set if the data stored with that key contains a t_dictionary object.
- **long dictionary_hasentry (t_dictionary *d, t_symbol *key)**
Test a key to set if it exists in the dictionary.
- **long dictionary_getentrycount (t_dictionary *d)**
Return the number of keys in a dictionary.
- **t_max_err dictionary_getkeys (t_dictionary *d, long *numkeys, t_symbol ***keys)**
Retrieve all of the key names stored in a dictionary.

- void `dictionary_freekeys (t_dictionary *d, long numkeys, t_symbol **keys)`
Free memory allocated by the `dictionary_getkeys()` method.
- `t_max_err dictionary_deleteentry (t_dictionary *d, t_symbol *key)`
Remove a value from the dictionary.
- `t_max_err dictionary_chuckentry (t_dictionary *d, t_symbol *key)`
Remove a value from the dictionary without freeing it.
- `t_max_err dictionary_clear (t_dictionary *d)`
Delete all values from a dictionary.
- void `dictionary_funall (t_dictionary *d, method fun, void *arg)`
Call the specified function for every entry in the dictionary.
- `t_symbol * dictionary_entry_getkey (t_dictionary_entry *x)`
Given a `t_dictionary_entry`, return the key associated with that entry.*
- void `dictionary_entry_getvalue (t_dictionary_entry *x, t_atom *value)`
Given a `t_dictionary_entry`, return the value associated with that entry.*
- `t_max_err dictionary_copyunique (t_dictionary *d, t_dictionary *copyfrom)`
Given 2 dictionaries, copy the keys unique to one of the dictionaries to the other dictionary.
- `t_max_err dictionary_getdeflong (t_dictionary *d, t_symbol *key, long *value, long def)`
Retrieve a long integer from the dictionary.
- `t_max_err dictionary_getdeffloat (t_dictionary *d, t_symbol *key, double *value, double def)`
Retrieve a double-precision float from the dictionary.
- `t_max_err dictionary_getdefsym (t_dictionary *d, t_symbol *key, t_symbol **value, t_symbol *def)`
Retrieve a `t_symbol` from the dictionary.*
- `t_max_err dictionary_getdefatom (t_dictionary *d, t_symbol *key, t_atom *value, t_atom *def)`
Retrieve a `t_atom` from the dictionary.*
- `t_max_err dictionary_getdefstring (t_dictionary *d, t_symbol *key, const char **value, char *def)`
Retrieve a C-string from the dictionary.
- `t_max_err dictionary_getdefatoms (t_dictionary *d, t_symbol *key, long *argc, t_atom **argv, t_atom *def)`
Retrieve the address of a `t_atom` array of in the dictionary.
- `t_max_err dictionary_copydefatoms (t_dictionary *d, t_symbol *key, long *argc, t_atom **argv, t_atom *def)`
Retrieve copies of a `t_atom` array in the dictionary.
- `t_max_err dictionary_dump (t_dictionary *d, long recurse, long console)`
Print the contents of a dictionary to the Max window.

- `t_max_err dictionary_copyentries (t_dictionary *src, t_dictionary *dst, t_symbol **keys)`
Copy specified entries from one dictionary to another.
- `t_dictionary * dictionary_sprintf (C74_CONST char *fmt,...)`
Create a new dictionary populated with values using a combination of attribute and sprintf syntax.
- `t_max_err dictionary_read (char *filename, short path, t_dictionary **d)`
Read the specified JSON file and return a `t_dictionary` object.
- `t_max_err dictionary_write (t_dictionary *d, char *filename, short path)`
Serialize the specified `t_dictionary` object to a JSON file.
- `void postdictionary (t_object *d)`
Print the contents of a dictionary to the Max window.

33.8.1 Detailed Description

In Max 5, we have a new "dictionary" object which can be used for object prototypes, object serialization, object constructors, and other tasks. A dictionary is ultimately a collection of atom values assigned to symbolic keys. In addition to primitive `A_LONG`, `A_FLOAT`, and `A_SYM` atom types, the `A_OBJ` atom type is used for `t_atomarray` (for a set of atoms assigned to a key), `t_dictionary` (for hierarchical use), `t_string` (for large blocks of text which we don't wish to bloat the symbol table), and potentially other object data types. Internally, the dictionary object uses a combination data structure of a hash table (for fast key lookup) and a linked-list (to maintain ordering of information within the dictionary).

Dictionaries are clonable entities, but note that all the member objects of a given dictionary may not be clonable. At the time of this writing, for example, the `t_string` object is not clonable, though it will be made clonable in the near future. In order for prototype entities to be guaranteed their passage into the constructor, they must be clonable (currently a symbol conversion is in place for the `t_string` class).

33.8.2 Using Dictionaries

Dictionaries are used in many places in Max 5. They can be confusing in many respects. It is easy to produce memory leaks or bugs where objects are freed twice. It is easy to confuse what type of dictionary is used for what. This page will begin with some high level information to help understand when to free and when not to free. Then, we will offer recipes for using dictionaries to accomplish common tasks.

33.8.2.1 Understanding Dictionaries

A dictionary stores atom values under named key entries. These atoms can contain `A_OBJ` values. When the dictionary is freed, any `A_OBJ` values that are in the dictionary will also be freed. Thus, it is easy to mistakenly free objects twice, thus this is something to be careful about. For example, look at this code:

```
t_dictionary *d = dictionary_new();
t_dictionary *sd = dictionary_new();
dictionary_appenddictionary(d, gensym("subdictionary"), sd);
do_something(d);
object_free(d); // this will free *both* d and sd since sd is contained by d
// freeing "sd" here would be bad
```

You primarily need to keep this in mind when calling [dictionary_appendobject\(\)](#), [dictionary_appenddictionary\(\)](#), or [dictionary_appendatomarray\(\)](#). So, what do you do if you need to free a dictionary but you also want to hang on to an object that is inside of the dictionary? In this case, chuck the entry in question first. For example, let's assume that for some reason you cannot free the "sd" dictionary in the code above. Perhaps it doesn't belong to you. But, to do some operation you need to append it to a new dictionary. Then, do this:

```
void function_foo(t_dictionary *sd) {
    t_dictionary *d = dictionary_new();
    dictionary_appenddictionary(d, gensym("subdictionary"), sd);
    do_something(d);
    dictionary_chuckentry(d, gensym("subdictionary"));
    object_free(d);
}
```

33.8.2.2 When to Free a Dictionary

So, how do you know when you need to free a dictionary? Well, generally if you make a dictionary, you need to free it when you are done (unless you transfer ownership of the dictionary to someone else). On the other hand, if you are passed a dictionary (i.e. as a parameter of your function or method) then it is not yours to free and you should just use it. However, it is not always obvious that you made a dictionary vs just borrowed it.

Here are some common (and not so common) ways to make a dictionary. These functions return a new dictionary and thus the dictionary you get should be freed when you are done, unless you pass the dictionary on to someone else who will free it at an appropriate time. Here they are:

- [dictionary_new\(\)](#)
- [dictionary_clone\(\)](#)
- [dictionary_read\(\)](#)
- [dictionary_sprintf\(\)](#)
- [dictionary_vsprintf\(\)](#)
- [jsonreader_parse\(\)](#)
- [jpatcher_monikerforobject\(\)](#)
- [class_cloneprototype\(\)](#)
- [prototype_getdictionary\(\)](#)
- [clipboard_todictionary\(\)](#)
- [jpatchercontroller_copytodictionary\(\)](#)
- probably others of course

Here are some functions that return borrowed dictionaries. These are dictionaries that you can use but you cannot free since you do not own them. Here they are:

- [dictionary_prototypefromclass\(\)](#)
- [object_refpage_get_class_info_fromclassname\(\)](#)
- [object_refpage_get_class_info\(\)](#)

- [object_dictionaryarg\(\)](#)

Finally, most functions that accept dictionaries as parameters will not assume ownership of the dictionary. Usually the way ownership is assumed is if you add a dictionary as a subdictionary to a dictionary that you do not own. One exception is the utility `newobject_fromdictionary_delete()` who's name makes it clear that the dictionary will be deleted after calling the function.

33.8.2.3 Some Common Uses of Dictionaries

You can make a patcher by passing a dictionary to `object_new_typed()` when making a "jpatcher". Using `atom_setparse()` and `attr_args_dictionary()` makes this relatively easy.

Use `newobject_sprintf()` to programmatically make an object in a patch. Actually, you don't explicitly use a dictionary here! If you do want more control, so you can touch the dictionary to customize it, then see the next bullet.

Use `dictionary_sprintf()` to make a dictionary to specify a box (i.e. specify class with @maxclass attr). Then, make another dictionary and append your box dictionary to it under the key "box" via `dictionary_appenddictionary()`. Finally, make your object with `newobject_fromdictionary()`.

See also

[Linked List](#)
[Hash Table](#)

Version

5.0

33.8.3 Function Documentation

33.8.3.1 `t_max_err dictionary_appendatom (t_dictionary * d, t_symbol * key, t_atom * value)`

Add a `t_atom*` value to the dictionary.

Parameters

d The dictionary instance.

key The name of the key used to index the new value. All keys must be unique. If the key name already exists, then the existing value associated with the key will be freed prior to the new value's assignment.

value The new value to append to the dictionary.

Returns

A Max error code.

33.8.3.2 `t_max_err dictionary_appendatomarray (t_dictionary * d, t_symbol * key, t_object * value)`

Add an [Atom Array](#) object to the dictionary.

Note that from this point on that you should not free the `t_atomarray*`, because the atomarray is now owned by the dictionary, and freeing the dictionary will free the atomarray as discussed in [When to Free a Dictionary](#).

Parameters

d The dictionary instance.

key The name of the key used to index the new value. All keys must be unique. If the key name already exists, then the existing value associated with the key will be freed prior to the new value's assignment.

value The new value to append to the dictionary.

Returns

A Max error code.

33.8.3.3 t_max_err dictionary_appendatoms (t_dictionary * *d*, t_symbol * *key*, long *argc*, t_atom * *argv*)

Add an array of atoms to the dictionary.

Internally these atoms will be copied into a [t_atomarray](#) object, which will be appended to the dictionary with the given key.

Parameters

d The dictionary instance.

key The name of the key used to index the new value. All keys must be unique. If the key name already exists, then the existing value associated with the key will be freed prior to the new value's assignment.

argc The number of atoms to append to the dictionary.

argv The address of the first atom in the array to append to the dictionary.

Returns

A Max error code.

33.8.3.4 t_max_err dictionary_appenddictionary (t_dictionary * *d*, t_symbol * *key*, t_object * *value*)

Add a dictionary object to the dictionary.

Note that from this point on that you should not free the [t_dictionary*](#) that is being added, because the newly-added dictionary is now owned by the dictionary to which it has been added, as discussed in [When to Free a Dictionary](#).

Parameters

d The dictionary instance.

key The name of the key used to index the new value. All keys must be unique. If the key name already exists, then the existing value associated with the key will be freed prior to the new value's assignment.

value The new value to append to the dictionary.

Returns

A Max error code.

33.8.3.5 t_max_err dictionary_appendfloat (t_dictionary * *d*, t_symbol * *key*, double *value*)

Add a double-precision float value to the dictionary.

Parameters

d The dictionary instance.

key The name of the key used to index the new value. All keys must be unique. If the key name already exists, then the existing value associated with the key will be freed prior to the new value's assignment.

value The new value to append to the dictionary.

Returns

A Max error code.

33.8.3.6 t_max_err dictionary_appendlong (t_dictionary * *d*, t_symbol * *key*, long *value*)

Add a long integer value to the dictionary.

Parameters

d The dictionary instance.

key The name of the key used to index the new value. All keys must be unique. If the key name already exists, then the existing value associated with the key will be freed prior to the new value's assignment.

value The new value to append to the dictionary.

Returns

A Max error code.

33.8.3.7 t_max_err dictionary_appendobject (t_dictionary * *d*, t_symbol * *key*, t_object * *value*)

Add an object to the dictionary.

Note that from this point on that you should not free the `t_object*` that is being added, because the newly-added object is now owned by the dictionary to which it has been added, as discussed in [When to Free a Dictionary](#).

Parameters

d The dictionary instance.

key The name of the key used to index the new value. All keys must be unique. If the key name already exists, then the existing value associated with the key will be freed prior to the new value's assignment.

value The new value to append to the dictionary.

Returns

A Max error code.

33.8.3.8 `t_max_err dictionary_appendstring (t_dictionary *d, t_symbol *key, const char *value)`

Add a C-string to the dictionary.

Internally this uses the `t_symbol` object. It is useful to use the `t_string` in dictionaries rather than the `t_symbol` to avoid bloating Max's symbol table unnecessarily.

Parameters

d The dictionary instance.

key The name of the key used to index the new value. All keys must be unique. If the key name already exists, then the existing value associated with the key will be freed prior to the new value's assignment.

value The new value to append to the dictionary.

Returns

A Max error code.

33.8.3.9 `t_max_err dictionary_appendsym (t_dictionary *d, t_symbol *key, t_symbol *value)`

Add a `t_symbol*` value to the dictionary.

Parameters

d The dictionary instance.

key The name of the key used to index the new value. All keys must be unique. If the key name already exists, then the existing value associated with the key will be freed prior to the new value's assignment.

value The new value to append to the dictionary.

Returns

A Max error code.

33.8.3.10 `t_max_err dictionary_chuckentry (t_dictionary *d, t_symbol *key)`

Remove a value from the dictionary without freeing it.

Parameters

d The dictionary instance.

key The key associated with the value to delete.

Returns

A max error code.

See also

[dictionary_deleteentry\(\)](#)

33.8.3.11 **t_max_err dictionary_clear (t_dictionary * *d*)**

Delete all values from a dictionary.

This method will free the objects in the dictionary. If freeing the objects is inappropriate or undesirable then you should iterate through the dictionary and use [dictionary_chuckentry\(\)](#) instead.

Parameters

d The dictionary instance.

Returns

A max error code.

See also

[dictionary_getkeys\(\)](#)
[dictionary_chuckentry\(\)](#)
[dictionary_deleteentry\(\)](#)

33.8.3.12 **t_max_err dictionary_copyatoms (t_dictionary * *d*, t_symbol * *key*, long * *argc*, t_atom ** *argv*)**

Retrieve copies of a [t_atom](#) array in the dictionary.

The retrieved pointer of [t_atoms](#) in the dictionary has memory allocated and copied to it from within the function. You are responsible for freeing it with [sysmem_freeptr\(\)](#).

Parameters

d The dictionary instance.

key The key associated with the value to lookup.

argc The address of a variable to hold the number of atoms in the array.

argv The address of a variable to hold a pointer to the first atom in the array. You should initialize this pointer to NULL prior to passing it to [dictionary_copyatoms\(\)](#).

Returns

A Max error code.

See also

[dictionary_getatoms\(\)](#)

33.8.3.13 **t_max_err dictionary_copydefatoms (t_dictionary * *d*, t_symbol * *key*, long * *argc*, t_atom ** *argv*, t_atom * *def*)**

Retrieve copies of a [t_atom](#) array in the dictionary.

The retrieved pointer of [t_atoms](#) in the dictionary has memory allocated and copied to it from within the function. You are responsible for freeing it with [sysmem_freeptr\(\)](#). If the named key doesn't exist, then copy a default array of atoms, specified as a [t_atomarray*](#).

Parameters

- d* The dictionary instance.
- key* The key associated with the value to lookup.
- argc* The address of a variable to hold the number of atoms in the array.
- argv* The address of a variable to hold a pointer to the first atom in the array. You should initialize this pointer to NULL prior to passing it to [dictionary_copyatoms\(\)](#).
- def* The default values specified as an instance of the [t_atomarray](#) object.

Returns

A Max error code.

See also

[dictionary_getdefatoms\(\)](#)
[dictionary_copyatoms\(\)](#)

33.8.3.14 t_max_err dictionary_copyentries (t_dictionary * src, t_dictionary * dst, t_symbol ** keys)

Copy specified entries from one dictionary to another.

Parameters

- src* The source dictionary from which to copy entries.
- dst* The destination dictionary to which the entries will be copied.
- keys* The address of the first of an array of [t_symbol](#)* that specifies which keys to copy.

Returns

A Max error code.

See also

[dictionary_copyunique\(\)](#)

33.8.3.15 t_max_err dictionary_copyunique (t_dictionary * d, t_dictionary * copyfrom)

Given 2 dictionaries, copy the keys unique to one of the dictionaries to the other dictionary.

Parameters

- d* A dictionary instance. This will be the destination for any values that are copied.
- copyfrom* A dictionary instance from which we will copy any values with unique keys.

Returns

A Max error code.

See also

[dictionary_copyentries\(\)](#)

33.8.3.16 `t_max_err dictionary_deleteentry (t_dictionary *d, t_symbol *key)`

Remove a value from the dictionary.

This method will free the object in the dictionary. If freeing the object is inappropriate or undesirable, use [dictionary_chuckentry\(\)](#) instead.

Parameters

d The dictionary instance.

key The key associated with the value to delete.

Returns

A max error code.

See also

[dictionary_chuckentry\(\)](#)

[dictionary_clear\(\)](#)

33.8.3.17 `t_max_err dictionary_dump (t_dictionary *d, long recurse, long console)`

Print the contents of a dictionary to the Max window.

Parameters

d The dictionary instance.

recurse If non-zero, the dictionary will be recursively unravelled to the Max window. Otherwise it will only print the top level.

console If non-zero, the dictionary will be posted to the console rather than the Max window. On the Mac you can view this using Console.app. On Windows you can use the free DbgView program which can be downloaded from Microsoft.

Returns

A Max error code.

33.8.3.18 `t_symbol* dictionary_entry_getkey (t_dictionary_entry *x)`

Given a [t_dictionary_entry*](#), return the key associated with that entry.

Parameters

x The dictionary entry.

Returns

The key associated with the entry.

See also

[dictionary_entry_getvalue\(\)](#)

[dictionary_funall\(\)](#)

33.8.3.19 void dictionary_entry_getvalue (t_dictionary_entry **x*, t_atom **value*)

Given a [t_dictionary_entry*](#), return the value associated with that entry.

Parameters

x The dictionary entry.

value The address of a [t_atom](#) to which the value will be copied.

See also

[dictionary_entry_getkey\(\)](#)

[dictionary_funall\(\)](#)

33.8.3.20 long dictionary_entryisatomarray (t_dictionary **d*, t_symbol **key*)

Test a key to set if the data stored with that key contains a [t_atomarray](#) object.

Parameters

d The dictionary instance.

key The key associated with the value to test.

Returns

Returns true if the key contains a [t_atomarray](#), otherwise returns false.

33.8.3.21 long dictionary_entryisdictionary (t_dictionary **d*, t_symbol **key*)

Test a key to set if the data stored with that key contains a [t_dictionary](#) object.

Parameters

d The dictionary instance.

key The key associated with the value to test.

Returns

Returns true if the key contains a [t_dictionary](#), otherwise returns false.

33.8.3.22 long dictionary_entryisstring (t_dictionary **d*, t_symbol **key*)

Test a key to set if the data stored with that key contains a [t_string](#) object.

Parameters

d The dictionary instance.

key The key associated with the value to test.

Returns

Returns true if the key contains a [t_string](#), otherwise returns false.

33.8.3.23 void dictionary_freekeys (t_dictionary * *d*, long *numkeys*, t_symbol ** *keys*)

Free memory allocated by the [dictionary_getkeys\(\)](#) method.

Parameters

d The dictionary instance.

numkeys The address of a long where the number of keys retrieved will be set.

keys The address of the first of an array [t_symbol](#) pointers where the retrieved keys will be set.

Returns

A max error code.

See also

[dictionary_getkeys\(\)](#)

33.8.3.24 void dictionary_funall (t_dictionary * *d*, method *fun*, void * *arg*)

Call the specified function for every entry in the dictionary.

Parameters

d The dictionary instance.

fun The function to call, specified as function pointer cast to a Max [method](#).

arg An argument that you would like to pass to the function being called.

Remarks

The [dictionary_funall\(\)](#) method will call your function for every entry in the dictionary. It will pass both a pointer to the [t_dictionary_entry](#), and any argument that you provide. The following example shows a function that could be called by [dictionary_funall\(\)](#).

```
void my_function(t_dictionary_entry *entry, void* my_arg)
{
    t_symbol *key;
    t_atom     value;

    key = dictionary_entry_getkey(entry);
    dictionary_entry_getvalue(entry, &value);

    // do something with key, value, and my_arg...
}
```

See also

[dictionary_entry_getkey\(\)](#)
[dictionary_entry_getvalue\(\)](#)

33.8.3.25 t_max_err dictionary_getatom (t_dictionary * *d*, t_symbol * *key*, t_atom * *value*)

Copy a [t_atom](#) from the dictionary.

Parameters

d The dictionary instance.

key The key associated with the value to lookup.

value The address of variable to hold the value associated with the key.

Returns

A Max error code.

33.8.3.26 **t_max_err dictionary_getatomarray (t_dictionary *d, t_symbol *key, t_object **value)**

Retrieve a [t_atomarray](#) pointer from the dictionary.

Parameters

d The dictionary instance.

key The key associated with the value to lookup.

value The address of variable to hold the value associated with the key.

Returns

A Max error code.

33.8.3.27 **t_max_err dictionary_getatoms (t_dictionary *d, t_symbol *key, long *argc, t_atom **argv)**

Retrieve the address of a [t_atom](#) array of in the dictionary.

The retrieved pointer references the [t_atoms](#) in the dictionary. To fetch a copy of the [t_atoms](#) from the dictionary, use [dictionary_copyatoms\(\)](#).

Parameters

d The dictionary instance.

key The key associated with the value to lookup.

argc The address of a variable to hold the number of atoms in the array.

argv The address of a variable to hold a pointer to the first atom in the array.

Returns

A Max error code.

See also

[dictionary_copyatoms\(\)](#)

33.8.3.28 **t_max_err dictionary_getdefatom (t_dictionary *d, t_symbol *key, t_atom *value, t_atom *def)**

Retrieve a [t_atom*](#) from the dictionary.

If the named key doesn't exist, then return a specified default value.

Parameters

- d* The dictionary instance.
- key* The key associated with the value to lookup.
- value* The address of variable to hold the value associated with the key.
- def* The default value to return in the absence of the key existing in the dictionary.

Returns

A Max error code.

See also

[dictionary_getatom\(\)](#)

33.8.3.29 `t_max_err dictionary_getdefatoms (t_dictionary *d, t_symbol *key, long *argc, t_atom **argv, t_atom *def)`

Retrieve the address of a `t_atom` array of in the dictionary.

The retrieved pointer references the `t_atoms` in the dictionary. To fetch a copy of the `t_atoms` from the dictionary, use [dictionary_copyatoms\(\)](#). If the named key doesn't exist, then return a default array of atoms, specified as a `t_atomarray*`.

Parameters

- d* The dictionary instance.
- key* The key associated with the value to lookup.
- argc* The address of a variable to hold the number of atoms in the array.
- argv* The address of a variable to hold a pointer to the first atom in the array.
- def* The default values specified as an instance of the `t_atomarray` object.

Returns

A Max error code.

See also

[dictionary_getatoms\(\)](#)
[dictionary_copydefatoms\(\)](#)

33.8.3.30 `t_max_err dictionary_getdeffloat (t_dictionary *d, t_symbol *key, double *value, double def)`

Retrieve a double-precision float from the dictionary.

If the named key doesn't exist, then return a specified default value.

Parameters

- d* The dictionary instance.
- key* The key associated with the value to lookup.

value The address of variable to hold the value associated with the key.

def The default value to return in the absence of the key existing in the dictionary.

Returns

A Max error code.

See also

[dictionary_getfloat\(\)](#)

33.8.3.31 `t_max_err dictionary_getdeflong (t_dictionary *d, t_symbol *key, long *value, long def)`

Retrieve a long integer from the dictionary.

If the named key doesn't exist, then return a specified default value.

Parameters

d The dictionary instance.

key The key associated with the value to lookup.

value The address of variable to hold the value associated with the key.

def The default value to return in the absence of the key existing in the dictionary.

Returns

A Max error code.

See also

[dictionary_getlong\(\)](#)

33.8.3.32 `t_max_err dictionary_getdefstring (t_dictionary *d, t_symbol *key, const char **value, char *def)`

Retrieve a C-string from the dictionary.

If the named key doesn't exist, then return a specified default value.

Parameters

d The dictionary instance.

key The key associated with the value to lookup.

value The address of variable to hold the value associated with the key.

def The default value to return in the absence of the key existing in the dictionary.

Returns

A Max error code.

See also

[dictionary_getstring\(\)](#)

33.8.3.33 `t_max_err dictionary_getdefsym (t_dictionary *d, t_symbol *key, t_symbol **value, t_symbol *def)`

Retrieve a `t_symbol*` from the dictionary.

If the named key doesn't exist, then return a specified default value.

Parameters

d The dictionary instance.

key The key associated with the value to lookup.

value The address of variable to hold the value associated with the key.

def The default value to return in the absence of the key existing in the dictionary.

Returns

A Max error code.

See also

[dictionary_getsym\(\)](#)

33.8.3.34 `t_max_err dictionary_getdictionary (t_dictionary *d, t_symbol *key, t_object **value)`

Retrieve a `t_dictionary` pointer from the dictionary.

Parameters

d The dictionary instance.

key The key associated with the value to lookup.

value The address of variable to hold the value associated with the key.

Returns

A Max error code.

33.8.3.35 `long dictionary_getentrycount (t_dictionary *d)`

Return the number of keys in a dictionary.

Parameters

d The dictionary instance.

Returns

The number of keys in the dictionary.

33.8.3.36 `t_max_err dictionary_getfloat (t_dictionary *d, t_symbol *key, double *value)`

Retrieve a double-precision float from the dictionary.

Parameters

d The dictionary instance.

key The key associated with the value to lookup.

value The address of variable to hold the value associated with the key.

Returns

A Max error code.

33.8.3.37 `t_max_err dictionary_getkeys (t_dictionary *d, long *numkeys, t_symbol *keys)`**

Retrieve all of the key names stored in a dictionary.

The numkeys and keys parameters should be initialized to zero. The [dictionary_getkeys\(\)](#) method will allocate memory for the keys it returns. You are then responsible for freeing this memory using [dictionary_freekeys\(\)](#). You must use [dictionary_freekeys\(\)](#), not some other method for freeing the memory.

Parameters

d The dictionary instance.

numkeys The address of a long where the number of keys retrieved will be set.

keys The address of the first of an array [t_symbol](#) pointers where the retrieved keys will be set.

Returns

A max error code.

Remarks

The following example demonstrates fetching all of the keys from a dictionary named 'd' in order to iterate through each item stored in the dictionary.

```
t_symbol **keys = NULL;
long      numkeys = 0;
long      i;
t_object *anItem;

dictionary_getkeys(d, &numkeys, &keys);
for(i=0; i<numkeys; i++) {
    // do something with the keys...
}
if(keys)
    dictionary_freekeys(d, numkeys, keys);
```

See also

[dictionary_freekeys\(\)](#)

33.8.3.38 `t_max_err dictionary_getlong (t_dictionary *d, t_symbol *key, long *value)`

Retrieve a long integer from the dictionary.

Parameters

d The dictionary instance.

key The key associated with the value to lookup.

value The address of variable to hold the value associated with the key.

Returns

A Max error code.

33.8.3.39 `t_max_err dictionary_getobject (t_dictionary *d, t_symbol *key, t_object **value)`

Retrieve a [t_object](#) pointer from the dictionary.

Parameters

d The dictionary instance.

key The key associated with the value to lookup.

value The address of variable to hold the value associated with the key.

Returns

A Max error code.

33.8.3.40 `t_max_err dictionary_getstring (t_dictionary *d, t_symbol *key, const char **value)`

Retrieve a C-string pointer from the dictionary.

The retrieved pointer references the string in the dictionary, it is not a copy.

Parameters

d The dictionary instance.

key The key associated with the value to lookup.

value The address of variable to hold the value associated with the key.

Returns

A Max error code.

33.8.3.41 `t_max_err dictionary_getsym (t_dictionary *d, t_symbol *key, t_symbol **value)`

Retrieve a [t_symbol](#)* from the dictionary.

Parameters

d The dictionary instance.

key The key associated with the value to lookup.

value The address of variable to hold the value associated with the key.

Returns

A Max error code.

33.8.3.42 long dictionary_hasentry (*t_dictionary* * *d*, *t_symbol* * *key*)

Test a key to set if it exists in the dictionary.

Parameters

d The dictionary instance.

key The key associated with the value to test.

Returns

Returns true if the key exists, otherwise returns false.

33.8.3.43 *t_dictionary** dictionary_new ()

Create a new linklist object.

You can free the linklist by calling [object_free\(\)](#). However, you should keep in mind the guidelines provided in [When to Free a Dictionary](#).

Returns

Pointer to the new dictionary object.

See also

[object_free\(\)](#)

33.8.3.44 *t_max_err* dictionary_read (*char* * *filename*, *short* *path*, *t_dictionary* ** *d*)

Read the specified JSON file and return a [t_dictionary](#) object.

You are responsible for freeing the dictionary with [object_free\(\)](#), subject to the caveats explained in [When to Free a Dictionary](#).

Parameters

filename The name of the file.

path The path of the file.

d The address of a [t_dictionary](#) pointer that will be set to the newly created dictionary.

Returns

A Max error code

33.8.3.45 `t_dictionary* dictionary_sprintf (C74_CONST char *fmt, ...)`

Create a new dictionary populated with values using a combination of attribute and sprintf syntax.

Parameters

- fmt* An sprintf-style format string specifying key-value pairs with attribute nomenclature.
- ... One or more arguments which are to be substituted into the format string.

Returns

A new dictionary instance.

Remarks

Max attribute syntax is used to define key-value pairs. For example,

```
"@key1 value @key2 another_value"
```

One common use of this to create dictionary that represents an element of a patcher, or even an entire patcher itself. The example below creates a dictionary that can be passed to a function like [newobject_fromdictionary\(\)](#) to create a new object.

```
t_dictionary *d;
char text[4];

strncpy_zero(text, "foo", 4);

d = dictionary_sprintf("@maxclass comment @varname _name \
@text \"%s\" @patching_rect %.2f %.2f %.2f \
@fontsize %f @textcolor %f %f %f 1.0 \
@fontname %s @bgcolor 0.001 0.001 0.001 0.",
text, 20.0, 20.0, 200.0, 24.0,
18, 0.9, 0.9, 0.9, "Arial");

// do something with the dictionary here.

object_free(d);
```

See also

- [newobject_sprintf\(\)](#)
- [newobject_fromdictionary\(\)](#)
- [atom_setparse\(\)](#)

33.8.3.46 `t_max_err dictionary_write (t_dictionary *d, char *filename, short path)`

Serialize the specified `t_dictionary` object to a JSON file.

Parameters

- d* The dictionary to serialize into JSON format and write to disk.
- filename* The name of the file to write.
- path* The path to which the file should be written.

Returns

A Max error code.

33.8.3.47 void postdictionary (*t_object* **d*)

Print the contents of a dictionary to the Max window.

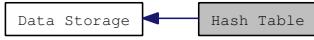
Parameters

d A pointer to a dictionary object.

33.9 Hash Table

Max's hashtab object implements a hash table (http://en.wikipedia.org/wiki/Hash_table).

Collaboration diagram for Hash Table:



Data Structures

- struct `t_hashtab_entry`

A hashtab entry.

- struct `t_hashtab`

The hashtab object.

Defines

- #define `HASH_DEFSLOTS` 57

Default number of slots in the hash table.

Functions

- `t_hashtab * hashtab_new (long slotcount)`

Create a new hashtab object.

- `t_max_err hashtab_store (t_hashtab *x, t_symbol *key, t_object *val)`

Store an item in a hashtab with an associated key.

- `t_max_err hashtab_store_safe (t_hashtab *x, t_symbol *key, t_object *val)`

Store an item in a hashtab with an associated key.

- `t_max_err hashtab_storeflags (t_hashtab *x, t_symbol *key, t_object *val, long flags)`

Store an item in a hashtab with an associated key and also flags that define the behavior of the item.

- `t_max_err hashtab_lookup (t_hashtab *x, t_symbol *key, t_object **val)`

Return an item stored in a hashtab with the specified key.

- `t_max_err hashtab_lookupflags (t_hashtab *x, t_symbol *key, t_object **val, long *flags)`

Return an item stored in a hashtab with the specified key, also returning the items flags.

- `t_max_err hashtab_delete (t_hashtab *x, t_symbol *key)`

Remove an item from a hashtab associated with the specified key and free it.

- `t_max_err hashtab_clear (t_hashtab *x)`

Delete all items stored in a hashtab.

- **t_max_err hashtab_chuckkey (t_hashtab *x, t_symbol *key)**
Remove an item from a hashtab associated with a given key.
- **t_max_err hashtab_chuck (t_hashtab *x)**
Free a hashtab, but don't free the items it contains.
- **t_max_err hashtab_findfirst (t_hashtab *x, void **o, long cmpfn(void *, void *), void *cmpdata)**
Search the hash table for the first item meeting defined criteria.
- **t_max_err hashtab_methodall (t_hashtab *x, t_symbol *s,...)**
Call the named message on every object in the hashtab.
- **t_max_err hashtab_funall (t_hashtab *x, method fun, void *arg)**
Call the specified function for every item in the hashtab.
- **long hashtab_getsize (t_hashtab *x)**
Return the number of items stored in a hashtab.
- **void hashtab_print (t_hashtab *x)**
Post a hashtab's statistics to the Max window.
- **void hashtab_READONLY (t_hashtab *x, long readonly)**
Set the hashtab's readonly bit.
- **void hashtab_FLAGS (t_hashtab *x, long flags)**
Set the hashtab's datastore flags.
- **long hashtab_getflags (t_hashtab *x)**
Get the hashtab's datastore flags.
- **t_max_err hashtab_KEYFLAGS (t_hashtab *x, t_symbol *key, long flags)**
Change the flags for an item stored in the hashtab with a given key.
- **long hashtab_getkeyflags (t_hashtab *x, t_symbol *key)**
Retrieve the flags for an item stored in the hashtab with a given key.
- **t_max_err hashtab_GETKEYS (t_hashtab *x, long *kc, t_symbol ***kv)**
Retrieve all of the keys stored in a hashtab.

33.9.1 Detailed Description

Max's hashtab object implements a hash table (http://en.wikipedia.org/wiki/Hash_table). Any type of value may be stored in the table, indexed using a **t_symbol** as the unique key.

See also

[Linked List](#)

33.9.2 Define Documentation

33.9.2.1 #define HASH_DEFSLOTS 57

Default number of slots in the hash table.

Creating a hashtab using [hashtab_new\(\)](#) with an argument of 0 will use the default number of slots. Primes typically work well for the number of slots.

33.9.3 Function Documentation

33.9.3.1 t_max_err hashtab_chuck (t_hashtab **x*)

Free a hashtab, but don't free the items it contains.

The hashtab can contain a variety of different types of data. By default, the hashtab assumes that all items are max objects with a valid [t_object](#) header.

You can alter the hashtab's notion of what it contains by using the [hashtab_flags\(\)](#) method.

When you free the hashtab by calling [object_free\(\)](#) it then tries to free all of the items it contains. If the hashtab is storing a custom type of data, or should otherwise not free the data it contains, then call [hashtab_chuck\(\)](#) to free the object instead of [object_free\(\)](#).

Parameters

x The hashtab object to be freed.

Returns

A max error code.

See also

[object_free](#)

33.9.3.2 t_max_err hashtab_chuckkey (t_hashtab **x*, t_symbol **key*)

Remove an item from a hashtab associated with a given key.

You are responsible for freeing any memory associated with the item that is removed from the hashtab.

Parameters

x The hashtab instance.

key The key of the item to delete.

Returns

A Max error code.

See also

[hashtab_delete](#)

33.9.3.3 `t_max_err hashtab_clear (t_hashtab *x)`

Delete all items stored in a hashtab.

This is the equivalent of calling `hashtab_delete()` on every item in a hashtab.

Returns

A max error code.

See also

`hashtab_flags()`

`hashtab_delete()`

33.9.3.4 `t_max_err hashtab_delete (t_hashtab *x, t_symbol *key)`

Remove an item from a hashtab associated with the specified key and free it.

The hashtab can contain a variety of different types of data. By default, the hashtab assumes that all items are max objects with a valid `t_object` header. Thus by default, it frees items by calling `object_free()` on them.

You can alter the hashtab's notion of what it contains by using the `hashtab_flags()` method.

If you wish to remove an item from the hashtab and free it yourself, then you should use `hashtab_chuckkey()`.

Parameters

x The hashtab instance.

key The key of the item to delete.

Returns

A Max error code.

See also

`hashtab_chuckkey()`

`hashtab_clear()`

`hashtab_flags()`

33.9.3.5 `t_max_err hashtab_findfirst (t_hashtab *x, void **o, long cmpfnvoid *, void *, void *cmpdata)`

Search the hash table for the first item meeting defined criteria.

The items in the hashtab are iteratively processed, calling a specified comparison function on each until the comparison function returns true.

Parameters

x The hashtab instance.

o The address to pointer that will be set with the matching item.

cmpfn The function used to determine a match in the list.

cmpdata An argument to be passed to the [t_cmpfn](#). This will be passed as the second of the two args to the [t_cmpfn](#). The first arg will be the hashtab item at each iteration in the list.

Returns

A max error code.

See also

[linklist_findfirst\(\)](#)
[t_cmpfn](#)

33.9.3.6 void hashtab_flags (t_hashtab *x, long flags)

Set the hashtab's datastore flags.

The available flags are enumerated in [e_max_datastore_flags](#). These flags control the behavior of the hashtab, particularly when removing items from the list using functions such as [hashtab_clear\(\)](#), [hashtab_delete\(\)](#), or when freeing the hashtab itself.

Parameters

x The hashtab instance.

flags A valid value from the [e_max_datastore_flags](#). The default is [OBJ_FLAG_OBJ](#).

33.9.3.7 t_max_err hashtab_funall (t_hashtab *x, method fun, void *arg)

Call the specified function for every item in the hashtab.

Parameters

x The hashtab instance.

fun The function to call, specified as function pointer cast to a Max [method](#).

arg An argument that you would like to pass to the function being called.

Returns

A max error code.

Remarks

The [hashtab_funall\(\)](#) method will call your function for every item in the list. It will pass both a pointer to the item in the list, and any argument that you provide. The following example shows a function that could be called by [hashtab_funall\(\)](#).

```
void myFun(t_hashtab_entry *e, void *myArg)
{
    if (e->key && e->value) {
        // do something with e->key, e->value, and myArg here as appropriate
    }
}
```

33.9.3.8 long hashtab_getflags (t_hashtab **x*)

Get the hashtab's datastore flags.

Parameters

x The hashtab instance.

Returns

The current state of the hashtab flags as enumerated in [e_max_datastore_flags](#).

33.9.3.9 long hashtab_getkeyflags (t_hashtab **x*, t_symbol **key*)

Retrieve the flags for an item stored in the hashtab with a given key.

Parameters

x The hashtab instance.

key The key in the hashtab whose flags will be returned.

Returns

The flags for the given key.

See also

[hashtab_store_flags\(\)](#)

33.9.3.10 t_max_err hashtab_getkeys (t_hashtab **x*, long **kc*, t_symbol **kv*)**

Retrieve all of the keys stored in a hashtab.

If the *kc* and *kv* parameters are properly initialized to zero, then [hashtab_getkeys\(\)](#) will allocate memory for the keys it returns. You are then responsible for freeing this memory using [sysmem_freeptr\(\)](#).

Parameters

x The hashtab instance.

kc The address of a long where the number of keys retrieved will be set.

kv The address of the first of an array [t_symbol](#) pointers where the retrieved keys will be set.

Returns

A max error code.

Remarks

The following example demonstrates fetching all of the keys from a hashtab in order to iterate through each item stored in the hashtab.

```
t_symbol **keys = NULL;
long      numKeys = 0;
long      i;
t_object *anItem;
```

```

hashtab_getkeys(aHashtab, &numKeys, &keys);
for(i=0; i<numKeys; i++) {
    hashtab_lookup(aHashtab, keys[i], &anItem);
    // Do something with anItem here...
}
if(keys)
    sysmem_freeptr(keys);

```

33.9.3.11 long hashtab_getsize (*t_hashtab* **x*)

Return the number of items stored in a hashtab.

Parameters

x The hashtab instance.

Returns

The number of items in the hash table.

33.9.3.12 t_max_err hashtab_keyflags (*t_hashtab* **x*, *t_symbol* **key*, long*flags*)

Change the flags for an item stored in the hashtab with a given key.

Parameters

x The hashtab instance.

key The key in the hashtab whose flags will be changed.

flags One of the values listed in [e_max_datastore_flags](#).

Returns

A Max error code.

See also

[hashtab_store_flags\(\)](#)

33.9.3.13 t_max_err hashtab_lookup (*t_hashtab* **x*, *t_symbol* **key*, *t_object* ***val*)

Return an item stored in a hashtab with the specified key.

Parameters

x The hashtab instance.

key The key in the hashtab to fetch.

val The address of a pointer to which the fetched value will be assigned.

Returns

A Max error code.

See also

[hashtab_store\(\)](#)

33.9.3.14 `t_max_err hashtab_lookupflags (t_hashtab *x, t_symbol *key, t_object **val, long *flags)`

Return an item stored in a hashtab with the specified key, also returning the items flags.

Parameters

- x* The hashtab instance.
- key* The key in the hashtab to fetch.
- val* The address of a pointer to which the fetched value will be assigned.
- flags* The address of a value to which the fetched flags will be assigned.

Returns

A Max error code.

See also

[hashtab_lookup\(\)](#)
[hashtab_store_flags\(\)](#)

33.9.3.15 `t_max_err hashtab_methodall (t_hashtab *x, t_symbol *s, ...)`

Call the named message on every object in the hashtab.

The [hashtab_methodall\(\)](#) function requires that all items in the hashtab are object instances with a valid `t_object` header.

Parameters

- x* The hashtab instance.
- s* The name of the message to send to the objects.
- ... Any arguments to be sent with the message.

Returns

A max error code.

Remarks

Internally, this function uses [object_method\(\)](#), meaning that no errors will be posted if the message name does not exist for the object. It also means that messages sent methods with [A_GIMME](#) definitions will need to be given a symbol argument prior to the argc and argv array information.

33.9.3.16 `t_hashtab* hashtab_new (long slotcount)`

Create a new hashtab object.

You can free the hashtab by calling [object_free\(\)](#) on the hashtab's pointer, or by using [hashtab_chuck\(\)](#).

Parameters

- slotcount* The number of slots in the hash table. Prime numbers typically work well. Pass 0 to get the default size.

Returns

Pointer to the new hashtab object.

See also

[HASH_DEFSLOTS](#)
[object_free\(\)](#)
[hashtab_chuck\(\)](#)

33.9.3.17 void hashtab_print (**t_hashtab** **x*)

Post a hashtab's statistics to the Max window.

Parameters

x The hashtab instance.

33.9.3.18 void hashtab_READONLY (**t_hashtab** **x*, long *readonly*)

Set the hashtab's readonly bit.

By default the readonly bit is 0, indicating that it is threadsafe for both reading and writing. Setting the readonly bit to 1 will disable the hashtab's threadsafety mechanism, increasing performance but at the expense of threadsafe operation. Unless you can guarantee the threading context for a hashtab's use, you should leave this set to 0.

Parameters

x The hashtab instance.

readonly A 1 or 0 for setting the readonly bit.

33.9.3.19 t_max_err hashtab_store (**t_hashtab** **x*, **t_symbol** **key*, **t_object** **val*)

Store an item in a hashtab with an associated key.

Parameters

x The hashtab instance.

key The key in the hashtab with which to associate the value.

val The value to store.

Returns

A Max error code.

See also

[hashtab_lookup\(\)](#)

33.9.3.20 t_max_err hashtab_store_safe (t_hashtab **x*, t_symbol **key*, t_object **val*)

Store an item in a hashtab with an associated key.

The difference between [hashtab_store\(\)](#) and [hashtab_store\(\)](#) is what happens in the event of a collision in the hash table. The normal [hashtab_store\(\)](#) function will free the existing value at the collision location with [sysmem_freeptr\(\)](#) and then replaces it. This version doesn't try to free the existing value at the collision location, but instead just over-writes it.

Parameters

x The hashtab instance.

key The key in the hashtab with which to associate the value.

val The value to store.

Returns

A Max error code.

See also

[hashtab_store\(\)](#)

33.9.3.21 t_max_err hashtab_storeflags (t_hashtab **x*, t_symbol **key*, t_object **val*, long*flags*)

Store an item in a hashtab with an associated key and also flags that define the behavior of the item.

The [hashtab_store\(\)](#) method is the same as calling this method with the default (0) flags.

Parameters

x The hashtab instance.

key The key in the hashtab with which to associate the value.

val The value to store.

flags One of the values listed in [e_max_datastore_flags](#).

Returns

A Max error code.

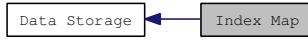
See also

[hashtab_store\(\)](#)

33.10 Index Map

An indexmap is basically a managed array of pointers, but it allows you to derive relatively quickly the index from a pointer in the array.

Collaboration diagram for Index Map:



Data Structures

- struct [t_indexmap_entry](#)

An indexmap element.

- struct [t_indexmap](#)

An indexmap object.

Functions

- [t_indexmap * indexmap_new \(void\)](#)

Create a new indexmap object.

- [void indexmap_append \(t_indexmap *x, void *data\)](#)

Add an item to an indexmap.

- [t_max_err indexmap_move \(t_indexmap *x, void *data, long newindex\)](#)

Move an item to a different position in an indexmap.

- [t_max_err indexmap_delete \(t_indexmap *x, void *data\)](#)

Delete a specified item from an indexmap.

- [t_max_err indexmap_delete_index \(t_indexmap *x, long index\)](#)

Delete an item from the indexmap by index.

- [t_max_err indexmap_delete_multi \(t_indexmap *x, long count, void **pdata\)](#)

Delete multiple specified items from an indexmap.

- [t_max_err indexmap_delete_index_multi \(t_indexmap *x, long count, long *indices\)](#)

Delete multiple items from an indexmap by index.

- [void * indexmap_datafromindex \(t_indexmap *x, long index\)](#)

Get an item from an indexmap by index.

- [t_max_err indexmap_indexfromdata \(t_indexmap *x, void *data, long *index\)](#)

Find the index of an item given a pointer to the item.

- [long indexmap_getsize \(t_indexmap *x\)](#)

Return the number of items in an indexmap.

- `void indexmap_clear (t_indexmap *x)`
Delete all items in an indexmap.
- `void indexmap_sort (t_indexmap *x, t_cmpfn fn)`
Sort the items in an indexmap.

33.10.1 Detailed Description

An indexmap is basically a managed array of pointers, but it allows you to derive relatively quickly the index from a pointer in the array. The index is assumed to be 0-N (where N is the current size of the array). You can sort the data and retain access to an index from the data relatively quickly. There is a hashtable which holds pieces of memory that hold indices that can be referenced by the data pointer. There is also an array of data pointers -- this is in "index" order. When operations take place on the array (insert, delete, sort), the pointers in the hashtable are updated with new indices.

33.10.2 Function Documentation

33.10.2.1 void indexmap_append (t_indexmap *x, void *data)

Add an item to an indexmap.

Parameters

- x* The indexmap instance.
data The item to add.

33.10.2.2 void indexmap_clear (t_indexmap *x)

Delete all items in an indexmap.

Parameters

- x* The indexmap instance.

33.10.2.3 void* indexmap_datafromindex (t_indexmap *x, long index)

Get an item from an indexmap by index.

Parameters

- x* The indexmap instance.
index The index from which to fetch a stored item.

Returns

The item stored at the specified index.

33.10.2.4 t_max_err indexmap_delete (t_indexmap **x*, void **data*)

Delete a specified item from an indexmap.

Parameters

- x* The indexmap instance.
- data* The item pointer to remove from the indexmap.

Returns

A Max error code.

33.10.2.5 t_max_err indexmap_delete_index (t_indexmap **x*, long *index*)

Delete an item from the indexmap by index.

Parameters

- x* The indexmap instance.
- index* The index of the item to remove from the indexmap.

Returns

A Max error code.

33.10.2.6 t_max_err indexmap_delete_index_multi (t_indexmap **x*, long *count*, long **indices*)

Delete multiple items from an indexmap by index.

Parameters

- x* The indexmap instance.
- count* The number of items to remove from the indexmap.
- indices* The address of the first of an array of index numbers to remove the indexmap.

Returns

A Max error code.

33.10.2.7 t_max_err indexmap_delete_multi (t_indexmap **x*, long *count*, void *pdata*)**

Delete multiple specified items from an indexmap.

Parameters

- x* The indexmap instance.
- count* The number of items to remove from the indexmap.
- pdata* The address of the first of an array of item pointers to remove from the indexmap.

Returns

A Max error code.

33.10.2.8 long indexmap_getsize (t_indexmap **x*)

Return the number of items in an indexmap.

Parameters

x The indexmap instance.

Returns

The number of items in the indexmap.

33.10.2.9 t_max_err indexmap_indexfromdata (t_indexmap **x*, void **data*, long **index*)

Find the index of an item given a pointer to the item.

Parameters

x The indexmap instance.

data The item whose index you wish to look up.

index The address of a variable to hold the retrieved index.

Returns

A Max error code.

33.10.2.10 t_max_err indexmap_move (t_indexmap **x*, void **data*, long *newindex*)

Move an item to a different position in an indexmap.

Parameters

x The indexmap instance.

data The item in the indexmap to move.

newindex The new index to which to move the item.

Returns

A Max error code.

33.10.2.11 t_indexmap* indexmap_new (void)

Create a new indexmap object.

Returns

Pointer to the new indexmap object.

33.10.2.12 void indexmap_sort (*t_indexmap* **x*, *t_cmpfn* *fn*)

Sort the items in an indexmap.

Items are sorted using a [*t_cmpfn*](#) function that is passed in as an argument.

Parameters

x The indexmap instance.

fn The function used to sort the list.

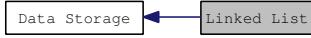
See also

[linklist_sort\(\)](#)

33.11 Linked List

Max's linklist object implements a doubly-linked-list (http://en.wikipedia.org/wiki/Linked_list) together with a high-level interface for manipulating and accessing values in the list.

Collaboration diagram for Linked List:



Data Structures

- struct `t_llitem`
A linklist element.
- struct `t_linklist`
The linklist object.

Functions

- `t_linklist * linklist_new (void)`
Create a new linklist object.
- `void linklist_chuck (t_linklist *x)`
Free a linklist, but don't free the items it contains.
- `long linklist_getsize (t_linklist *x)`
Return the number of items in a linklist object.
- `void * linklist_getindex (t_linklist *x, long index)`
Return the item stored in a linklist at a specified index.
- `long linklist_objptr2index (t_linklist *x, void *p)`
Return an item's index, given the item itself.
- `long linklist_append (t_linklist *x, void *o)`
Add an item to the end of the list.
- `long linklist_insertindex (t_linklist *x, void *o, long index)`
Insert an item into the list at the specified index.
- `long linklist_insert_sorted (t_linklist *x, void *o, long cmpfn(void *, void *))`
Insert an item into the list, keeping the list sorted according to a specified comparison function.
- `t_llitem * linklist_insertafterobjptr (t_linklist *x, void *o, void *objptr)`
Insert an item into the list after another specified item.
- `t_llitem * linklist_insertbeforeobjptr (t_linklist *x, void *o, void *objptr)`

Insert an item into the list before another specified item.

- `t_llitem * linklist_moveafterobjptr (t_linklist *x, void *o, void *objptr)`

Move an existing item in the list to a position after another specified item in the list.

- `t_llitem * linklist_movebeforeobjptr (t_linklist *x, void *o, void *objptr)`

Move an existing item in the list to a position before another specified item in the list.

- `long linklist_deleteindex (t_linklist *x, long index)`

Remove the item from the list at the specified index and free it.

- `long linklist_chuckindex (t_linklist *x, long index)`

Remove the item from the list at the specified index.

- `void linklist_chuckobject (t_linklist *x, void *o)`

Remove the specified item from the list.

- `void linklist_clear (t_linklist *x)`

Remove and free all items in the list.

- `long linklist_makearray (t_linklist *x, void **a, long max)`

Retrieve linklist items as an array of pointers.

- `void linklist_reverse (t_linklist *x)`

Reverse the order of items in the linked-list.

- `void linklist_rotate (t_linklist *x, long i)`

Rotate items in the linked list in circular fashion.

- `void linklist_shuffle (t_linklist *x)`

Randomize the order of items in the linked-list.

- `void linklist_swap (t_linklist *x, long a, long b)`

Swap the position of two items in the linked-list, specified by index.

- `long linklist_findfirst (t_linklist *x, void **o, long cmpfn(void *, void *), void *cmpdata)`

Search the linked list for the first item meeting defined criteria.

- `void linklist_findall (t_linklist *x, t_linklist **out, long cmpfn(void *, void *), void *cmpdata)`

Search the linked list for all items meeting defined criteria.

- `void linklist_methodall (t_linklist *x, t_symbol *s,...)`

Call the named message on every object in the linklist.

- `void * linklist_methodindex (t_linklist *x, long i, t_symbol *s,...)`

Call the named message on an object specified by index.

- `void linklist_sort (t_linklist *x, long cmpfn(void *, void *))`

Sort the linked list.

- void `linklist_funall` (`t_linklist *x`, `method` fun, `void *arg`)
Call the specified function for every item in the linklist.
- long `linklist_funall_break` (`t_linklist *x`, `method` fun, `void *arg`)
Call the specified function for every item in the linklist.
- void * `linklist_funindex` (`t_linklist *x`, long i, `method` fun, `void *arg`)
Call the specified function for an item specified by index.
- void * `linklist_substitute` (`t_linklist *x`, void *p, `void *newp`)
Given an item in the list, replace it with a different value.
- void * `linklist_next` (`t_linklist *x`, void *p, `void **next`)
Given an item in the list, find the next item.
- void * `linklist_prev` (`t_linklist *x`, void *p, `void **prev`)
Given an item in the list, find the previous item.
- void * `linklist_last` (`t_linklist *x`, `void **item`)
Return the last item (the tail) in the linked-list.
- void `linklist_READONLY` (`t_linklist *x`, long readonly)
Set the linklist's readonly bit.
- void `linklist_FLAGS` (`t_linklist *x`, long flags)
Set the linklist's datastore flags.
- long `linklist_getflags` (`t_linklist *x`)
Get the linklist's datastore flags.
- long `linklist_MATCH` (void *a, void *b)
A linklist comparison method that determines if two item pointers are equal.

33.11.1 Detailed Description

Max's linklist object implements a doubly-linked-list (http://en.wikipedia.org/wiki/Linked_list) together with a high-level interface for manipulating and accessing values in the list.

33.11.2 Function Documentation

33.11.2.1 long `linklist_append` (`t_linklist *x`, `void *o`)

Add an item to the end of the list.

Parameters

- `x` The linklist instance.
- `o` The item pointer to append to the linked-list.

Returns

The updated size of the linklist after appending the new item, or -1 if the append failed.

33.11.2.2 void linklist_chuck (*t_linklist* **x*)

Free a linklist, but don't free the items it contains.

The linklist can contain a variety of different types of data. By default, the linklist assumes that all items are max objects with a valid *t_object* header.

You can alter the linklist's notion of what it contains by using the [linklist_flags\(\)](#) method.

When you free the linklist by calling [object_free\(\)](#) it then tries to free all of the items it contains. If the linklist is storing a custom type of data, or should otherwise not free the data it contains, then call [linklist_chuck\(\)](#) to free the object instead of [object_free\(\)](#).

Parameters

x The linklist object to be freed.

See also

[object_free](#)

33.11.2.3 long linklist_chuckindex (*t_linklist* **x*, long *index*)

Remove the item from the list at the specified index.

You are responsible for freeing any memory associated with the item that is removed from the linklist.

Parameters

x The linklist instance.

index The index of the item to remove.

Returns

Returns [MAX_ERR_NONE](#) on successful removal, otherwise returns [MAX_ERR_GENERIC](#)

See also

[linklist_deleteindex](#)

[linklist_chuckobject](#)

33.11.2.4 void linklist_chuckobject (*t_linklist* **x*, void **o*)

Remove the specified item from the list.

You are responsible for freeing any memory associated with the item that is removed from the linklist.

Parameters

x The linklist instance.

o The pointer to the item to remove.

See also

[linklist_deleteindex](#)
[linklist_chuckindex](#)

33.11.2.5 void linklist_clear (*t_linklist* **x*)

Remove and free all items in the list.

Freeing items in the list is subject to the same rules as [linklist_deleteindex\(\)](#). You can alter the linklist's notion of what it contains, and thus how items are freed, by using the [linklist_flags\(\)](#) method.

Parameters

x The linklist instance.

33.11.2.6 long linklist_deleteindex (*t_linklist* **x*, long *index*)

Remove the item from the list at the specified index and free it.

The linklist can contain a variety of different types of data. By default, the linklist assumes that all items are max objects with a valid [t_object](#) header. Thus by default, it frees items by calling [object_free\(\)](#) on them.

You can alter the linklist's notion of what it contains by using the [linklist_flags\(\)](#) method.

If you wish to remove an item from the linklist and free it yourself, then you should use [linklist_chuckptr\(\)](#).

Parameters

x The linklist instance.

index The index of the item to delete.

Returns

Returns the index number of the item deleted, or -1 if the operation failed.

See also

[linklist_chuckindex](#)
[linklist_chuckobject](#)

33.11.2.7 void linklist_findall (*t_linklist* **x*, *t_linklist* *out*, long *cmpfnvoid* *, void *, void * *cmpdata*)**

Search the linked list for all items meeting defined criteria.

The items in the list are traversed, calling a specified comparison function on each, and returning the matches in another linklist.

Parameters

x The linklist instance.

out The address to a [t_linklist](#) pointer. You should initialize the pointer to NULL before calling [linklist_findall\(\)](#). A new linklist will be created internally by [linklist_findall\(\)](#) and returned here.

cmpfn The function used to determine a match in the list.

cmpdata An argument to be passed to the [t_cmpfn](#). This will be passed as the second of the two args to the [t_cmpfn](#). The first arg will be the linklist item at each iteration in the list.

Remarks

The following example assumes you have a linklist called myLinkList, and [t_cmpfn](#) called myCmpFunction, and some sort of data to match in someCriteria.

```
t_linklist *results = NULL;

linklist_findall(myLinkList, &results, myCmpFunction, (void *)someCriteria);
// do something here with the 'results' linklist
// then free the results linklist
linklist_chuck(results);
```

See also

[linklist_match](#)
[t_cmpfn](#)
[linklist_findfirst](#)

33.11.2.8 long linklist_findfirst (t_linklist *x, void **o, long *cmpfn* void *, void *, void * *cmpdata*)

Search the linked list for the first item meeting defined criteria.

The items in the list are traversed, calling a specified comparison function on each until the comparison function returns true.

Parameters

- x*** The linklist instance.
- o*** The address to pointer that will be set with the matching item.
- cmpfn*** The function used to determine a match in the list.
- cmpdata*** An argument to be passed to the [t_cmpfn](#). This will be passed as the second of the two args to the [t_cmpfn](#). The first arg will be the linklist item at each iteration in the list.

Returns

The index of the matching item, or -1 if no match is found.

Remarks

The following shows how to manually do what [linklist_chuckobject\(\)](#) does.

```
void *obj;
long index;

index = linklist_findfirst(x, &obj, #linklist_match, o);
if(index != -1)
    linklist_chuckindex(x, index);
```

See also

[linklist_match](#)
[t_cmpfn](#)
[linklist_findall](#)

33.11.2.9 void linklist_flags (t_linklist *x, long flags)

Set the linklist's datastore flags.

The available flags are enumerated in [e_max_datastore_flags](#). These flags control the behavior of the linklist, particularly when removing items from the list using functions such as [linklist_clear\(\)](#), [linklist_deleteindex\(\)](#), or when freeing the linklist itself.

Parameters

x The linklist instance.

flags A valid value from the [e_max_datastore_flags](#). The default is [OBJ_FLAG_OBJ](#).

33.11.2.10 void linklist_funall (t_linklist *x, method fun, void *arg)

Call the specified function for every item in the linklist.

Parameters

x The linklist instance.

fun The function to call, specified as function pointer cast to a Max [method](#).

arg An argument that you would like to pass to the function being called.

Remarks

The [linklist_funall\(\)](#) method will call your function for every item in the list. It will pass both a pointer to the item in the list, and any argument that you provide. The following example shows a function that could be called by [linklist_funall\(\)](#).

```
void myFun(t_object *myObj, void *myArg)
{
    // do something with myObj and myArg here
    // myObj is the item in the linklist
}
```

33.11.2.11 long linklist_funall_break (t_linklist *x, method fun, void *arg)

Call the specified function for every item in the linklist.

The iteration through the list will halt if the function returns a non-zero value.

Parameters

x The linklist instance.

fun The function to call, specified as function pointer cast to a Max [method](#).

arg An argument that you would like to pass to the function being called.

Remarks

The [linklist_funall\(\)](#) method will call your function for every item in the list. It will pass both a pointer to the item in the list, and any argument that you provide. The following example shows a function that could be called by [linklist_funall\(\)](#).

```

long myFun(t_symbol *myListSymbol, void *myArg)
{
    // this function is called by a linklist that contains symbols for its item
    if(myListSymbol == gensym("")) {
        error("empty symbol -- aborting linklist traversal")
        return 1;
    }
    else{
        // do something with the symbol
        return 0;
    }
}

```

33.11.2.12 void* linklist_funindex (t_linklist **x*, long *i*, method *fun*, void **arg*)

Call the specified function for an item specified by index.

Parameters

- x* The linklist instance.
- i* The index of the item to which to send the message.
- fun* The function to call, specified as function pointer cast to a Max [method](#).
- arg* An argument that you would like to pass to the function being called.

Remarks

The [linklist_funindex\(\)](#) method will call your function for an item in the list. It will pass both a pointer to the item in the list, and any argument that you provide. The following example shows a function that could be called by [linklist_funindex\(\)](#).

```

void myFun(t_object *myObj, void *myArg)
{
    // do something with myObj and myArg here
    // myObj is the item in the linklist
}

```

33.11.2.13 long linklist_getflags (t_linklist **x*)

Get the linklist's datastore flags.

Parameters

- x* The linklist instance.

Returns

The current state of the linklist flags as enumerated in [e_max_datastore_flags](#).

33.11.2.14 void* linklist_getindex (t_linklist **x*, long *index*)

Return the item stored in a linklist at a specified index.

Parameters

- x* The linklist instance.

index The index in the linklist to fetch. Indices are zero-based.

Returns

The item from the linklist stored at index. If there is no item at the index, NULL is returned

33.11.2.15 long linklist_getsize (**t_linklist** **x*)

Return the number of items in a linklist object.

Parameters

x The linklist instance.

Returns

The number of items in the linklist object.

33.11.2.16 long linklist_insert_sorted (**t_linklist** **x*, **void** **o*, **long cmpfnvoid** *, **void** *)

Insert an item into the list, keeping the list sorted according to a specified comparison function.

Parameters

x The linklist instance.

o The item pointer to insert.

cmpfn A comparison function by which the list should be sorted.

Returns

The index of the new item in the linklist, or -1 if the insert failed.

33.11.2.17 **t_llitem*** linklist_insertafterobjptr (**t_linklist** **x*, **void** **o*, **void** **objptr*)

Insert an item into the list after another specified item.

Parameters

x The linklist instance.

o The item pointer to insert.

objptr The item pointer after which to insert in the list.

Returns

An opaque linklist element.

33.11.2.18 `t_llitem* linklist_insertbeforeobjptr (t_linklist *x, void *o, void *objptr)`

Insert an item into the list before another specified item.

Parameters

- x* The linklist instance.
- o* The item pointer to insert.
- objptr* The item pointer before which to insert in the list.

Returns

An opaque linklist element.

33.11.2.19 `long linklist_insertindex (t_linklist *x, void *o, long index)`

Insert an item into the list at the specified index.

Parameters

- x* The linklist instance.
- o* The item pointer to insert.
- index* The index at which to insert. Index 0 is the head of the list.

Returns

The index of the item in the linklist, or -1 if the insert failed.

33.11.2.20 `void* linklist_last (t_linklist *x, void **item)`

Return the last item (the tail) in the linked-list.

Parameters

- x* The linklist instance.
- item* The address of pointer in which to store the last item in the linked-list.

Returns

always returns NULL

33.11.2.21 `long linklist_makearray (t_linklist *x, void **a, long max)`

Retrieve linklist items as an array of pointers.

Parameters

- x* The linklist instance.
- a* The address of the first pointer in the array to fill.
- max* The number of pointers in the array.

Returns

The number of items from the list actually returned in the array.

33.11.2.22 long linklist_match (void * *a*, void * *b*)

A linklist comparison method that determines if two item pointers are equal.

Parameters

- a* The first item to compare.
- b* The second item to compare.

Returns

Returns 1 if the items are equal, otherwise 0.

See also

[t_cmpfn](#)

33.11.2.23 void linklist_methodall (t_linklist * *x*, t_symbol * *s*, ...)

Call the named message on every object in the linklist.

The [linklist_methodall\(\)](#) function requires that all items in the linklist are object instances with a valid [t_object](#) header.

Parameters

- x* The linklist instance.
- s* The name of the message to send to the objects.
- ... Any arguments to be sent with the message.

Remarks

Internally, this function uses [object_method\(\)](#), meaning that no errors will be posted if the message name does not exist for the object. It also means that messages sent methods with [A_GIMME](#) definitions will need to be given a symbol argument prior to the argc and argv array information.

33.11.2.24 void* linklist_methodindex (t_linklist * *x*, long *i*, t_symbol * *s*, ...)

Call the named message on an object specified by index.

The item must be an object instance with a valid [t_object](#) header.

Parameters

- x* The linklist instance.
- i* The index of the item to which to send the message.
- s* The name of the message to send to the objects.
- ... Any arguments to be sent with the message.

Remarks

Internally, this function uses [object_method\(\)](#), meaning that no errors will be posted if the message name does not exist for the object. It also means that messages sent methods with [A_GIMME](#) definitions will need to be given a symbol argument prior to the argc and argv array information.

33.11.2.25 `t_llitem* linklist_moveafterobjptr (t_linklist *x, void *o, void *objptr)`

Move an existing item in the list to a position after another specified item in the list.

Parameters

- x* The linklist instance.
- o* The item pointer to insert.
- objptr* The item pointer after which to move o in the list.

Returns

An opaque linklist element.

33.11.2.26 `t_llitem* linklist_movebeforeobjptr (t_linklist *x, void *o, void *objptr)`

Move an existing item in the list to a position before another specified item in the list.

Parameters

- x* The linklist instance.
- o* The item pointer to insert.
- objptr* The item pointer before which to move o in the list.

Returns

An opaque linklist element.

33.11.2.27 `t_linklist* linklist_new (void)`

Create a new linklist object.

You can free the linklist by calling [object_free\(\)](#) on the linklist's pointer, or by using [linklist_chuck\(\)](#).

Returns

Pointer to the new linklist object.

See also

- [object_free\(\)](#)
- [linklist_chuck\(\)](#)

33.11.2.28 `void* linklist_next (t_linklist *x, void *p, void **next)`

Given an item in the list, find the next item.

This provides an means for walking the list.

Parameters

- x* The linklist instance.
- p* An item in the list.
- next* The address of a pointer to set with the next item in the list.

33.11.2.29 long linklist_objptr2index (t_linklist **x*, void **p*)

Return an item's index, given the item itself.

Parameters

- x* The linklist instance.
- p* The item pointer to search for in the linklist.

Returns

The index of the item given in the linklist. If the item is not in the linklist **MAX_ERR_GENERIC** is returned.

33.11.2.30 void* linklist_prev (t_linklist **x*, void **p*, void *prev*)**

Given an item in the list, find the previous item.

This provides an means for walking the list.

Parameters

- x* The linklist instance.
- p* An item in the list.
- prev* The address of a pointer to set with the previous item in the list.

33.11.2.31 void linklist_READONLY (t_linklist **x*, long *readonly*)

Set the linklist's readonly bit.

By default the readonly bit is 0, indicating that it is threadsafe for both reading and writing. Setting the readonly bit to 1 will disable the linklist's threadsafety mechanism, increasing performance but at the expense of threadsafe operation. Unless you can guarantee the threading context for a linklist's use, you should leave this set to 0.

Parameters

- x* The linklist instance.
- readonly* A 1 or 0 for setting the readonly bit.

33.11.2.32 void linklist_reverse (t_linklist **x*)

Reverse the order of items in the linked-list.

Parameters

- x* The linklist instance.

33.11.2.33 void linklist_rotate (t_linklist *x, long i)

Rotate items in the linked list in circular fashion.

Parameters

- x* The linklist instance.
- i* The number of positions in the list to shift items.

33.11.2.34 void linklist_shuffle (t_linklist *x)

Randomize the order of items in the linked-list.

Parameters

- x* The linklist instance.

33.11.2.35 void linklist_sort (t_linklist *x, long cmpfnvoid *, void *)

Sort the linked list.

The items in the list are ordered using a `t_cmpfn` function that is passed in as an argument.

Parameters

- x* The linklist instance.
- cmpfn* The function used to sort the list.

Remarks

The following is example is a real-world example of sorting a linklist of symbols alphabetically by first letter only. First the `cmpfn` is defined, then it is used in a different function by `linklist_sort()`.

```
long myAlphabeticalCmpfn(void *a, void *b)
{
    t_symbol *s1 = (t_symbol *)a;
    t_symbol *s2 = (t_symbol *)b;

    if(s1->s_name[0] < s2->s_name[0])
        return true;
    else
        return false;
}

void mySortMethod(t_myobj *x)
{
    // the linklist was already created and filled with items previously
    linklist_sort(x->myLinkList, myAlphabeticalCmpfn);
}
```

33.11.2.36 void* linklist_substitute (t_linklist *x, void *p, void *newp)

Given an item in the list, replace it with a different value.

Parameters

- x* The linklist instance.

p An item in the list.

newp The new value.

Returns

Always returns NULL.

33.11.2.37 void linklist_swap (t_linklist **x*, long *a*, long *b*)

Swap the position of two items in the linked-list, specified by index.

Parameters

x The linklist instance.

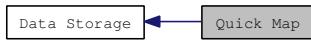
a The index of the first item to swap.

b The index of the second item to swap.

33.12 Quick Map

A quickmap implements a pair of [t_hashtab](#) hash tables so that it is fast to look up a unique value for a unique key or vice-versa.

Collaboration diagram for Quick Map:



Data Structures

- struct [t_quickmap](#)

The quickmap object.

Functions

- BEGIN_USING_C_LINKAGE void * [quickmap_new](#) (void)
Create a new quickmap object.
- void [quickmap_add](#) ([t_quickmap](#) *x, void *p1, void *p2)
Add a pair of keys mapped to each other to the quickmap.
- void [quickmap_drop](#) ([t_quickmap](#) *x, void *p1, void *p2)
Drop a pair of keys mapped to each other in the quickmap.
- long [quickmap_lookup_key1](#) ([t_quickmap](#) *x, void *p1, void **p2)
Given a (first) key, lookup the value (the second key).
- long [quickmap_lookup_key2](#) ([t_quickmap](#) *x, void *p1, void **p2)
Given a (second) key, lookup the value (the first key).
- void [quickmap_READONLY](#) ([t_quickmap](#) *x, long way)
Set the readonly flag of the quickmap's hash tables.

33.12.1 Detailed Description

A quickmap implements a pair of [t_hashtab](#) hash tables so that it is fast to look up a unique value for a unique key or vice-versa. This implies that both the keys and the values must be unique so that look-ups can be performed in both directions.

33.12.2 Function Documentation

33.12.2.1 void [quickmap_add](#) ([t_quickmap](#) * x, void * p1, void * p2)

Add a pair of keys mapped to each other to the quickmap.

Note that these are considered to be a `t_symbol` internally. This means that if you are mapping a `t_symbol` to a `t_object`, for example, the `t_object` will not automatically be freed when you free the quickmap (unlike what happens when you typically free a `t_hashtab`).

Parameters

- x* The quickmap instance.
- p1* The (first) key.
- p2* The value (or the second key).

Returns

A Max error code.

33.12.2.2 void quickmap_drop (`t_quickmap *x, void *p1, void *p2`)

Drop a pair of keys mapped to each other in the quickmap.

Parameters

- x* The quickmap instance.
- p1* The first key.
- p2* The second key.

Returns

A Max error code.

33.12.2.3 long quickmap_lookup_key1 (`t_quickmap *x, void *p1, void **p2`)

Given a (first) key, lookup the value (the second key).

Parameters

- x* The quickmap instance.
- p1* The (first) key.
- p2* The address of a pointer which will hold the resulting key upon return.

Returns

A Max error code.

33.12.2.4 long quickmap_lookup_key2 (`t_quickmap *x, void *p1, void **p2`)

Given a (second) key, lookup the value (the first key).

Parameters

- x* The quickmap instance.
- p1* The (second) key.
- p2* The address of a pointer which will hold the resulting key upon return.

Returns

A Max error code.

33.12.2.5 BEGIN_USING_C_LINKAGE void* quickmap_new (void)

Create a new quickmap object.

Returns

Pointer to the new quickmap object.

33.12.2.6 void quickmap_READONLY (t_quickmap *x, long way)

Set the readonly flag of the quickmap's hash tables.

See [hashtab_READONLY\(\)](#) for more information about this.

Parameters

x The quickmap instance.

way Set to true to make the quickmap readonly (disable thread protection) or false (the default) to enable thread protection.

33.13 String Object

Max's string object is a simple wrapper for c-strings, useful when working with Max's [t_dictionary](#), [t_linklist](#), or [t_hashtab](#).

Collaboration diagram for String Object:



Data Structures

- struct [t_string](#)

The string object.

Functions

- [t_string * string_new \(const char *psz\)](#)

Create a new string object.

- [const char * string_getptr \(t_string *x\)](#)

Create a new string object.

33.13.1 Detailed Description

Max's string object is a simple wrapper for c-strings, useful when working with Max's [t_dictionary](#), [t_linklist](#), or [t_hashtab](#).

See also

[Dictionary](#)

33.13.2 Function Documentation

33.13.2.1 [const char* string_getptr \(t_string * x\)](#)

Create a new string object.

Parameters

x The string object instance.

Returns

A pointer to the internally maintained C-string.

33.13.2.2 `t_string* string_new (const char *psz)`

Create a new string object.

Parameters

psz Pointer to a C-string that will be copied to memory internal to this string object instance.

Returns

The new string object instance pointer.

33.14 Symbol Object

The symobject class is a simple object that wraps a `t_symbol*` together with a couple of additional fields.

Collaboration diagram for Symbol Object:



Data Structures

- struct `t_symobject`

The symobject data structure.

Functions

- void * `symobject_new (t_symbol *sym)`

The symobject data structure.

- long `symobject_linklist_match (void *a, void *b)`

Utility for searching a linklist containing symobjects.

33.14.1 Detailed Description

The symobject class is a simple object that wraps a `t_symbol*` together with a couple of additional fields. It is useful for storing symbols, possibly with additional flags or pointers, into a [Hash Table](#) or [Linked List](#).

Version

5.0

33.14.2 Function Documentation

33.14.2.1 long `symobject_linklist_match (void * a, void * b)`

Utility for searching a linklist containing symobjects.

Parameters

a (opaque)

b (opaque)

Returns

Returns true if a match is found, otherwise returns false.

Remarks

The following example shows one common use of the this method.

```
t_symobject *item = NULL;
long      index;
t_symbol *textsym;

textsym = gensym("something to look for");

// search for a symobject with the symbol 'something to look for'
index = linklist_findfirst(s_ll_history, (void **) &item,
                           symobject_linklist_match, textsym);
if(index == -1){
    // symobject not found.
}
else{
    do something with the symobject, or with the index of the symobject in the
    linklist
}
```

33.14.2.2 void* symobject_new (t_symbol * *sym*)

The symobject data structure.

Parameters

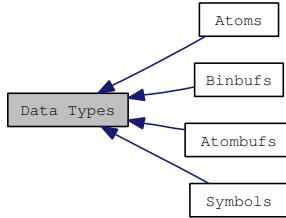
sym A symbol with which to initialize the new symobject.

Returns

Pointer to the new symobject instance.

33.15 Data Types

Collaboration diagram for Data Types:



Data Structures

- struct [t_rect](#)
Coordinates for specifying a rectangular region.
- struct [t_pt](#)
Coordinates for specifying a point.
- struct [t_size](#)
Coordinates for specifying the size of a region.

Modules

- [Atoms](#)
- [Atombufs](#)
An Atombuf is an alternative to [Binbufs](#) for temporary storage of atoms.
- [Binbufs](#)
*You won't need to know about the internal structure of a Binbuf, so you can use the void * type to refer to one.*
- [Symbols](#)
Max maintains a symbol table of all strings to speed lookup for message passing.

Typedefs

- [typedef void *\(* method \) \(void *,...\)](#)
Function pointer type for generic methods.
- [typedef long\(* longmethod \) \(void *,...\)](#)
Function pointer type for methods returning a long.
- [typedef void *\(* voidstarvoid \)\(\)](#)
Function pointer type for a function with no arguments, returning a generic pointer.

- **typedef char * t_ptr**
Generic pointer type.
- **typedef char ** t_handle**
Generic pointer-to-a-pointer type.
- **typedef void * t_vptr**
Void pointer type.
- **typedef void *(* zero_meth)(void *x)**
Function pointer type for methods with no arguments.
- **typedef void *(* one_meth)(void *x, void *z)**
Function pointer type for methods with a single argument.
- **typedef void *(* two_meth)(void *x, void *z, void *a)**
Function pointer type for methods with two arguments.
- **typedef long *(* gimmeback_meth)(void *x, t_symbol *s, long ac, t_atom *av, t_atom *rv)**
*Function pointer type for methods that pass back a result value through the last parameter as a **t_atom**, and return an error.*
- **typedef unsigned long ulong**
An unsigned long integer.
- **typedef unsigned int uint**
An unsigned integer.
- **typedef unsigned short ushort**
An unsigned short integer.
- **typedef unsigned char uchar**
An unsigned char.
- **typedef long t_max_err**
A Max error code.

33.15.1 Typedef Documentation

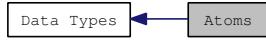
33.15.1.1 **typedef long t_max_err**

A Max error code.

Common error codes are defined in [e_max_errorcodes](#).

33.16 Atoms

Collaboration diagram for Atoms:



Data Structures

- union [word](#)
Union for packing any of the datum defined in [e_max_atomtypes](#).
- struct [t_atom](#)
An atom is a typed datum.

Defines

- #define [ATOM_MAX_STRLEN](#) (32768)
Defines the largest possible string size for an atom.

Enumerations

- enum [e_max_atomtypes](#) {
 [A_NOTHING](#) = 0,
 [A_LONG](#),
 [A_FLOAT](#),
 [A_SYM](#),
 [A_OBJ](#),
 [A_DEFLONG](#),
 [A_DEFFLOAT](#),
 [A_DEFSYM](#),
 [A_GIMME](#),
 [A_CANT](#),
 [A_SEMI](#),
 [A_COMMA](#),
 [A_DOLLAR](#),
 [A_DOLLSYM](#),
 [A_GIMMEBACK](#),
 [A_DEFER](#) = 0x41,
 [A_USURP](#) = 0x42,
 [A_DEFER_LOW](#) = 0x43,
 [A_USURP_LOW](#) = 0x44 }

the list of officially recognized types, including pseudotypes for commas and semicolons.

- enum `e_max_atom_gettext_flags` {

`OBEX_UTIL_ATOM_GETTEXT_DEFAULT` = 0x00000000,

`OBEX_UTIL_ATOM_GETTEXT_TRUNCATE_ZEROS` = 0x00000001,

`OBEX_UTIL_ATOM_GETTEXT_SYM_NO_QUOTE` = 0x00000002,

`OBEX_UTIL_ATOM_GETTEXT_SYM_FORCE_QUOTE` = 0x00000004,

`OBEX_UTIL_ATOM_GETTEXT_COMMA_DELIM` = 0x00000008,

`OBEX_UTIL_ATOM_GETTEXT_FORCE_ZEROS` = 0x00000010,

`OBEX_UTIL_ATOM_GETTEXT_NUM_HI_RES` = 0x00000020
 }

Flags that determine how functions convert atoms into text (C-strings).

Functions

- `t_max_err atom_setlong (t_atom *a, long b)`

Inserts an integer into a `t_atom` and change the `t_atom`'s type to `A_LONG`.
- `t_max_err atom_setfloat (t_atom *a, double b)`

Inserts a floating point number into a `t_atom` and change the `t_atom`'s type to `A_FLOAT`.
- `t_max_err atom_setsym (t_atom *a, t_symbol *b)`

*Inserts a `t_symbol` * into a `t_atom` and change the `t_atom`'s type to `A_SYM`.*
- `t_max_err atom_setobj (t_atom *a, void *b)`

Inserts a generic pointer value into a `t_atom` and change the `t_atom`'s type to `A_OBJ`.
- `long atom_getlong (t_atom *a)`

Retrieves a long integer value from a `t_atom`.
- `float atom_getfloat (t_atom *a)`

Retrieves a floating point value from a `t_atom`.
- `t_symbol * atom_getsym (t_atom *a)`

*Retrieves a `t_symbol` * value from a `t_atom`.*
- `void * atom_getobj (t_atom *a)`

Retrieves a generic pointer value from a `t_atom`.
- `long atom_getcharfix (t_atom *a)`

Retrieves an unsigned integer value between 0 and 255 from a `t_atom`.
- `long atom_gettype (t_atom *a)`

Retrieves type from a `t_atom`.
- `long atom_arg_getlong (long *c, long idx, long ac, t_atom *av)`

Retrieves the integer value of a particular `t_atom` from an atom list, if the atom exists.

- `long atom_arg_getfloat (float *c, long idx, long ac, t_atom *av)`
Retrieves the floating point value of a particular `t_atom` from an atom list, if the atom exists.
- `long atom_arg_getdouble (double *c, long idx, long ac, t_atom *av)`
Retrieves the floating point value, as a double, of a particular `t_atom` from an atom list, if the atom exists.
- `long atom_arg_getsym (t_symbol **c, long idx, long ac, t_atom *av)`
*Retrieves the `t_symbol *` value of a particular `t_atom` from an atom list, if the atom exists.*
- `t_max_err atom_alloc (long *ac, t_atom **av, char *alloc)`
Allocate a single atom.
- `t_max_err atom_alloc_array (long minsize, long *ac, t_atom **av, char *alloc)`
Allocate an array of atoms.
- `t_max_err atom_setchar_array (long ac, t_atom *av, long count, unsigned char *vals)`
Assign an array of char values to an array of atoms.
- `t_max_err atom_setlong_array (long ac, t_atom *av, long count, long *vals)`
Assign an array of long integer values to an array of atoms.
- `t_max_err atom_setfloat_array (long ac, t_atom *av, long count, float *vals)`
Assign an array of 32bit float values to an array of atoms.
- `t_max_err atom_setdouble_array (long ac, t_atom *av, long count, double *vals)`
Assign an array of 64bit float values to an array of atoms.
- `t_max_err atom_setsym_array (long ac, t_atom *av, long count, t_symbol **vals)`
Assign an array of `t_symbol` values to an array of atoms.*
- `t_max_err atom_setatom_array (long ac, t_atom *av, long count, t_atom *vals)`
Assign an array of `t_atom` values to an array of atoms.
- `t_max_err atom_setobj_array (long ac, t_atom *av, long count, t_object **vals)`
Assign an array of `t_object` values to an array of atoms.*
- `t_max_err atom_setparse (long *ac, t_atom **av, C74_CONST char *parsestr)`
Parse a C-string into an array of atoms.
- `t_max_err atom_setformat (long *ac, t_atom **av, C74_CONST char *fmt,...)`
Create an array of atoms populated with values using sprintf-like syntax.
- `t_max_err atom_getformat (long ac, t_atom *av, C74_CONST char *fmt,...)`
Retrieve values from an array of atoms using sscanf-like syntax.
- `t_max_err atom_gettext (long ac, t_atom *av, long *textsize, char **text, long flags)`
Convert an array of atoms into a C-string.
- `t_max_err atom_getchar_array (long ac, t_atom *av, long count, unsigned char *vals)`
Fetch an array of char values from an array of atoms.

- `t_max_err atom_getlong_array` (long ac, `t_atom` *av, long count, long *vals)
Fetch an array of long integer values from an array of atoms.
- `t_max_err atom_getfloat_array` (long ac, `t_atom` *av, long count, float *vals)
Fetch an array of 32bit float values from an array of atoms.
- `t_max_err atom_getdouble_array` (long ac, `t_atom` *av, long count, double *vals)
Fetch an array of 64bit float values from an array of atoms.
- `t_max_err atom_getsym_array` (long ac, `t_atom` *av, long count, `t_symbol` **vals)
Fetch an array of `t_symbol` values from an array of atoms.*
- `t_max_err atom_getatom_array` (long ac, `t_atom` *av, long count, `t_atom` *vals)
Fetch an array of `t_atom` values from an array of atoms.
- `t_max_err atom_getobj_array` (long ac, `t_atom` *av, long count, `t_object` **vals)
Fetch an array of `t_object` values from an array of atoms.*
- long `atomisstring` (`t_atom` *a)
Determines whether or not an atom represents a `t_string` object.
- long `atomisatomarray` (`t_atom` *a)
Determines whether or not an atom represents a `t_atomarray` object.
- long `atomisdictionary` (`t_atom` *a)
Determines whether or not an atom represents a `t_dictionary` object.
- BEGIN_USING_C_LINKAGE void `atom_copy` (short argc1, `t_atom` *argv1, `t_atom` *argv2)
Copy an array of atoms.
- void `postargs` (short argc, `t_atom` *argv)
Print the contents of an array of atoms to the Max window.
- `t_max_err atom_arg_getobjclass` (`t_object` **x, long idx, long argc, `t_atom` *argv, `t_symbol` *cls)
Return a pointer to an object contained in an atom if it is of the specified class.
- void * `atom_getobjclass` (`t_atom` *av, `t_symbol` *cls)
Return a pointer to an object contained in an atom if it is of the specified class.

33.16.1 Enumeration Type Documentation

33.16.1.1 enum e_max_atom_gettext_flags

Flags that determine how functions convert atoms into text (C-strings).

Enumerator:

`OBEX_UTIL_ATOM_GETTEXT_DEFAULT` default translation rules for getting text from atoms

OBEX_UTIL_ATOM_GETTEXT_TRUNCATE_ZEROS eliminate redundant zeros for floating point numbers (default used)

OBEX_UTIL_ATOM_GETTEXT_SYM_NO_QUOTE don't introduce quotes around symbols with spaces

OBEX_UTIL_ATOM_GETTEXT_SYM_FORCE_QUOTE always introduce quotes around symbols (useful for JSON)

OBEX_UTIL_ATOM_GETTEXT_COMMA_DELIM separate atoms with commas (useful for JSON)

OBEX_UTIL_ATOM_GETTEXT_FORCE_ZEROS always print the zeros

OBEX_UTIL_ATOM_GETTEXT_NUM_HI_RES print more decimal places

33.16.1.2 enum e_max_atomtypes

the list of officially recognized types, including pseudotypes for commas and semicolons.

Used in two places: 1. the reader, when it reads a string, returns long, float, sym, comma, semi, or dollar; and 2. each object method comes with an array of them saying what types it needs, from among long, float, sym, obj, gimme, and cant.

Remarks

While these values are defined in an enum, you should use a long to represent the value. Using the enum type creates ambiguity in struct size and is subject to various inconsistent compiler settings.

Enumerator:

A_NOTHING no type, thus no atom

A_LONG long integer

A_FLOAT 32-bit float

A_SYM [t_symbol](#) pointer

A_OBJ [t_object](#) pointer (for argtype lists; passes the value of sym)

A_DEFLONG long but defaults to zero

A_DEFFLOAT float, but defaults to zero

A_DEFSYM symbol, defaults to ""

A_GIMME request that args be passed as an array, the routine will check the types itself.

A_CANT cannot typecheck args

A_SEMI semicolon

A_COMMA comma

A_DOLLAR dollar

A_DOLLSYM dollar

A_GIMMEBACK request that args be passed as an array, the routine will check the types itself. can return atom value in final atom ptr arg. function returns long error code 0 = no err. see [gimmeback_meth](#) typedef

A_DEFER A special signature for declaring methods. This is like A_GIMME, but the call is deferred.

A_USURP A special signature for declaring methods. This is like A_GIMME, but the call is deferred and multiple calls within one servicing of the queue are filtered down to one call.

A_DEFER_LOW A special signature for declaring methods. This is like A_GIMME, but the call is deferref to the back of the queue.

A_USURP_LOW A special signature for declaring methods. This is like A_GIMME, but the call is deferred to the back of the queue and multiple calls within one servicing of the queue are filtered down to one call.

33.16.2 Function Documentation

33.16.2.1 `t_max_err atom_alloc (long * ac, t_atom ** av, char * alloc)`

Allocate a single atom.

If ac and av are both zero then memory is allocated. Otherwise it is presumed that memory is already allocated and nothing will happen.

Parameters

ac The address of a variable that will contain the number of atoms allocated (1).

av The address of a pointer that will be set with the new allocated memory for the atom.

alloc Address of a variable that will be set true if memory is allocated, otherwise false.

Returns

A Max error code.

33.16.2.2 `t_max_err atom_alloc_array (long minsize, long * ac, t_atom ** av, char * alloc)`

Allocate an array of atoms.

If ac and av are both zero then memory is allocated. Otherwise it is presumed that memory is already allocated and nothing will happen.

Parameters

minsize The minimum number of atoms that this array will need to contain. This determines the amount of memory allocated.

ac The address of a variable that will contain the number of atoms allocated.

av The address of a pointer that will be set with the new allocated memory for the atoms.

alloc Address of a variable that will be set true if memory is allocated, otherwise false.

Returns

A Max error code.

33.16.2.3 `long atom_arg_getdouble (double * c, long idx, long ac, t_atom * av)`

Retrieves the floating point value, as a double, of a particular `t_atom` from an atom list, if the atom exists.

Parameters

c Pointer to a double variable to receive the atom's data if the function is successful. Otherwise the value is left unchanged.

idx Offset into the atom list of the atom of interest, starting from 0. For instance, if you want data from the 3rd atom in the atom list, *idx* should be set to 2.

ac Count of av.

av Pointer to the first [t_atom](#) of an atom list.

Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.16.2.4 long atom_arg_getfloat (float * *c*, long *idx*, long *ac*, t_atom * *av*)

Retrieves the floating point value of a particular [t_atom](#) from an atom list, if the atom exists.

Parameters

c Pointer to a float variable to receive the atom's data if the function is successful. Otherwise, the value is left unchanged.

idx Offset into the atom list of the atom of interest, starting from 0. For instance, if you want data from the 3rd atom in the atom list, *idx* should be set to 2.

ac Count of av.

av Pointer to the first [t_atom](#) of an atom list.

Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.16.2.5 long atom_arg_getlong (long * *c*, long *idx*, long *ac*, t_atom * *av*)

Retrieves the integer value of a particular [t_atom](#) from an atom list, if the atom exists.

Parameters

c Pointer to a long variable to receive the atom's data if the function is successful.

idx Offset into the atom list of the atom of interest, starting from 0. For instance, if you want data from the 3rd atom in the atom list, *idx* should be set to 2.

ac Count of av.

av Pointer to the first [t_atom](#) of an atom list.

Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

Remarks

The [atom_arg_getlong\(\)](#) function only changes the value of *c* if the function is successful. For instance, the following code snippet illustrates a simple, but typical use:

```

void myobject_mymessage(t_myobject *x, t_symbol *s, long ac, t_atom *av)
{
    long var = -1;

    // here, we are expecting a value of 0 or greater
    atom_arg_getlong(&var, 0, ac, av);
    if (val == -1) // i.e. unchanged
        post("it is likely that the user did not provide a valid argument");
    else {
        ...
    }
}

```

33.16.2.6 `t_max_err atom_arg_getobjclass (t_object **x, long idx, long argc, t_atom *argv, t_symbol *cls)`

Return a pointer to an object contained in an atom if it is of the specified class.

Parameters

- x* The address of a pointer to the object contained in av if it is of the specified class upon return.
Otherwise NULL upon return.
- idx* The index of the atom in the array from which to get the object pointer.
- argc* The count of atoms in argv.
- argv* The address to the first of an array of atoms.
- cls* A symbol containing the class name of which the object should be an instance.

Returns

A Max error code.

33.16.2.7 `long atom_arg_getsym (t_symbol **c, long idx, long ac, t_atom *av)`

Retrieves the `t_symbol *` value of a particular `t_atom` from an atom list, if the atom exists.

Parameters

- c* Pointer to a `t_symbol *` variable to receive the atom's data if the function is successful. Otherwise, the value is left unchanged.
- idx* Offset into the atom list of the atom of interest, starting from 0. For instance, if you want data from the 3rd atom in the atom list, `idx` should be set to 2.
- ac* Count of av.
- av* Pointer to the first `t_atom` of an atom list.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

Remarks

The `atom_arg_getsym()` function only changes the value of `c` if the function is successful. For instance, the following code snippet illustrates a simple, but typical use:

```

void myobject_open(t_myobject *x, t_symbol *s, long ac, t_atom *av)
{
    t_symbol *filename = _sym_nothing;

    // here, we are expecting a file name.
    // if we don't get it, open a dialog box
    atom_arg_getsym(&filename, 0, ac, av);
    if (filename == _sym_nothing) { // i.e. unchanged
        // open the file dialog box,
        // get a value for filename
    }
    // do something with the filename
}

```

33.16.2.8 BEGIN_USING_C_LINKAGE void atom_copy (short argc1, t_atom *argv1, t_atom *argv2)

Copy an array of atoms.

Parameters

argc1 The count of atoms in argv1.

argv1 The address to the first of an array of atoms that is the source for the copy.

argv2 The address to the first of an array of atoms that is the destination for the copy. Note that this array must already be allocated using [sysmem_newptr\(\)](#) or [atom_alloc\(\)](#).

33.16.2.9 t_max_err atom_getatom_array (long ac, t_atom *av, long count, t_atom *vals)

Fetch an array of [t_atom](#) values from an array of atoms.

Parameters

ac The number of atoms allocated in the av parameter.

av The address to the first of an array of allocated atoms.

count The number of values to fetch from the array specified by vals.

vals The address of the array to which is copied the values from av.

Returns

A Max error code.

33.16.2.10 t_max_err atom_getchar_array (long ac, t_atom *av, long count, unsigned char *vals)

Fetch an array of char values from an array of atoms.

Parameters

ac The number of atoms allocated in the av parameter.

av The address to the first of an array of allocated atoms.

count The number of values to fetch from the array specified by vals.

vals The address of the array to which is copied the values from av.

Returns

A Max error code.

33.16.2.11 long atom_getcharfix (*t_atom* * *a*)

Retrieves an unsigned integer value between 0 and 255 from a [t_atom](#).

Parameters

a Pointer to a [t_atom](#) whose value is of interest

Returns

This function returns the value of the specified [t_atom](#) as an integer between 0 and 255, if possible. Otherwise, it returns 0.

Remarks

If the [t_atom](#) is typed [A_LONG](#), but the data falls outside of the range 0-255, the data is truncated to that range before output.

If the [t_atom](#) is typed [A_FLOAT](#), the floating point value is multiplied by 255. and truncated to the range 0-255 before output. For example, the floating point value 0.5 would be output from atom_getcharfix as 127 (0.5 * 255. = 127.5).

No attempt is also made to coerce [t_symbol](#) data.

33.16.2.12 t_max_err atom_getdouble_array (long *ac*, *t_atom* * *av*, long *count*, double * *vals*)

Fetch an array of 64bit float values from an array of atoms.

Parameters

ac The number of atoms allocated in the *av* parameter.

av The address to the first of an array of allocated atoms.

count The number of values to fetch from the array specified by *vals*.

vals The address of the array to which is copied the values from *av*.

Returns

A Max error code.

33.16.2.13 float atom_getfloat (*t_atom* * *a*)

Retrieves a floating point value from a [t_atom](#).

Parameters

a Pointer to a [t_atom](#) whose value is of interest

Returns

This function returns the value of the specified [t_atom](#) as a floating point number, if possible. Otherwise, it returns 0.

Remarks

If the [t_atom](#) is not of the type specified by the function, the function will attempt to coerce a valid value from the [t_atom](#). For instance, if the [t_atom](#) `at` is set to type [A_LONG](#) with a value of 5, the [atom_getfloat\(\)](#) function will return the value of `at` as a float, or 5.0. An attempt is also made to coerce [t_symbol](#) data.

33.16.2.14 [t_max_err atom_getfloat_array \(long ac, t_atom * av, long count, float * vals\)](#)

Fetch an array of 32bit float values from an array of atoms.

Parameters

- `ac` The number of atoms allocated in the `av` parameter.
- `av` The address to the first of an array of allocated atoms.
- `count` The number of values to fetch from the array specified by `vals`.
- `vals` The address of the array to which is copied the values from `av`.

Returns

A Max error code.

33.16.2.15 [t_max_err atom_getformat \(long ac, t_atom * av, C74_CONST char * fmt, ...\)](#)

Retrieve values from an array of atoms using sscanf-like syntax.

[atom_getformat\(\)](#) supports clfdsoaCLFDOSA tokens (primitive type scalars and arrays respectively for the char, long, float, double, [t_symbol*](#), [t_object*](#), [t_atom*](#)). It does not support vbp@ the tokens found in [atom_setformat\(\)](#).

Parameters

- `ac` The number of atoms to parse in `av`.
- `av` The address of the first [t_atom](#) pointer in an array to parse.
- `fmt` An sscanf-style format string specifying types for the atoms.
- ... One or more arguments which are address of variables to be set according to the `fmt` string.

Returns

A Max error code.

See also

[atom_setformat\(\)](#)

33.16.2.16 long atom_getlong (*t_atom* * *a*)

Retrieves a long integer value from a *t_atom*.

Parameters

a Pointer to a *t_atom* whose value is of interest

Returns

This function returns the value of the specified *t_atom* as an integer, if possible. Otherwise, it returns 0.

Remarks

If the *t_atom* is not of the type specified by the function, the function will attempt to coerce a valid value from the *t_atom*. For instance, if the *t_atom* at is set to type *A_FLOAT* with a value of 3.7, the *atom_getlong()* function will return the truncated integer value of at, or 3. An attempt is also made to coerce *t_symbol* data.

33.16.2.17 t_max_err atom_getlong_array (long *ac*, *t_atom* * *av*, long *count*, long * *vals*)

Fetch an array of long integer values from an array of atoms.

Parameters

ac The number of atoms allocated in the *av* parameter.

av The address to the first of an array of allocated atoms.

count The number of values to fetch from the array specified by *vals*.

vals The address of the array to which is copied the values from *av*.

Returns

A Max error code.

33.16.2.18 void* atom_getobj (*t_atom* * *a*)

Retrieves a generic pointer value from a *t_atom*.

Parameters

a Pointer to a *t_atom* whose value is of interest

Returns

This function returns the value of the specified *A_OBJ*-typed *t_atom*, if possible. Otherwise, it returns NULL.

33.16.2.19 `t_max_err atom_getobj_array (long ac, t_atom *av, long count, t_object **vals)`

Fetch an array of `t_object*` values from an array of atoms.

Parameters

- ac* The number of atoms allocated in the `av` parameter.
- av* The address to the first of an array of allocated atoms.
- count* The number of values to fetch from the array specified by `vals`.
- vals* The address of the array to which is copied the values from `av`.

Returns

A Max error code.

33.16.2.20 `void* atom_getobjclass (t_atom *av, t_symbol *cls)`

Return a pointer to an object contained in an atom if it is of the specified class.

Parameters

- av* A pointer to the atom from which to get the `t_object`.
- cls* A symbol containing the class name of which the object should be an instance.

Returns

A pointer to the object contained in `av` if it is of the specified class, otherwise NULL.

33.16.2.21 `t_symbol* atom_getsym (t_atom *a)`

Retrieves a `t_symbol *` value from a `t_atom`.

Parameters

- a* Pointer to a `t_atom` whose value is of interest

Returns

This function returns the value of the specified `A_SYM`-typed `t_atom`, if possible. Otherwise, it returns an empty, but valid, `t_symbol *`, equivalent to `gensym ("")`, or `_sym_nothing`.

Remarks

No attempt is made to coerce non-matching data types.

33.16.2.22 `t_max_err atom_getsym_array (long ac, t_atom *av, long count, t_symbol **vals)`

Fetch an array of `t_symbol*` values from an array of atoms.

Parameters

- ac* The number of atoms allocated in the `av` parameter.

av The address to the first of an array of allocated atoms.
count The number of values to fetch from the array specified by *vals*.
vals The address of the array to which is copied the values from *av*.

Returns

A Max error code.

33.16.2.23 `t_max_err atom_gettext (long ac, t_atom * av, long * textsize, char ** text, long flags)`

Convert an array of atoms into a C-string.

Parameters

ac The number of atoms to fetch in *av*.
av The address of the first `t_atom` pointer in an array to retrieve.
textsize The size of the string to which the atoms will be formatted and copied.
text The address of the string to which the text will be written.
flags Determines the rules by which atoms will be translated into text. Values are bit mask as defined by `e_max_atom_gettext_flags`.

Returns

A Max error code.

See also

[atom_setparse\(\)](#)

33.16.2.24 `long atom_gettype (t_atom * a)`

Retrieves type from a `t_atom`.

Parameters

a Pointer to a `t_atom` whose type is of interest

Returns

This function returns the type of the specified `t_atom` as defined in `e_max_atomtypes`

33.16.2.25 `t_max_err atom_setatom_array (long ac, t_atom * av, long count, t_atom * vals)`

Assign an array of `t_atom` values to an array of atoms.

Parameters

ac The number of atoms to try to fetch from the second array of atoms. You should have at least this number of atoms allocated in *av*.
av The address to the first of an array of allocated atoms.

count The number of values in the array specified by *vals*.

vals The array from which to copy the values into the array of atoms at *av*.

Returns

A Max error code.

33.16.2.26 **t_max_err atom_setchar_array (long *ac*, t_atom * *av*, long *count*, unsigned char * *vals*)**

Assign an array of char values to an array of atoms.

Parameters

ac The number of atoms to try to fetch from the array of chars. You should have at least this number of atoms allocated in *av*.

av The address to the first of an array of allocated atoms.

count The number of values in the array specified by *vals*.

vals The array from which to copy the values into the array of atoms at *av*.

Returns

A Max error code.

33.16.2.27 **t_max_err atom_setdouble_array (long *ac*, t_atom * *av*, long *count*, double * *vals*)**

Assign an array of 64bit float values to an array of atoms.

Parameters

ac The number of atoms to try to fetch from the array of doubles. You should have at least this number of atoms allocated in *av*.

av The address to the first of an array of allocated atoms.

count The number of values in the array specified by *vals*.

vals The array from which to copy the values into the array of atoms at *av*.

Returns

A Max error code.

33.16.2.28 **t_max_err atom_setfloat (t_atom * *a*, double *b*)**

Inserts a floating point number into a [t_atom](#) and change the *t_atom*'s type to [A_FLOAT](#).

Parameters

a Pointer to a [t_atom](#) whose value and type will be modified

b Floating point value to copy into the [t_atom](#)

Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.16.2.29 t_max_err atom_setfloat_array (long ac, t_atom *av, long count, float *vals)

Assign an array of 32bit float values to an array of atoms.

Parameters

- ac** The number of atoms to try to fetch from the array of floats. You should have at least this number of atoms allocated in av.
- av** The address to the first of an array of allocated atoms.
- count** The number of values in the array specified by vals.
- vals** The array from which to copy the values into the array of atoms at av.

Returns

A Max error code.

33.16.2.30 t_max_err atom_setformat (long *ac, t_atom **av, C74_CONST char *fmt, ...)

Create an array of atoms populated with values using sprintf-like syntax.

[atom_setformat\(\)](#) supports clfdsoaCLFDOSA tokens (primitive type scalars and arrays respectively for the char, long, float, double, [t_symbol*](#), [t_object*](#), [t_atom*](#)). It also supports vbp@ tokens (obval, binbuf, parsestr, attribute).

This function allocates memory for the atoms if the ac and av parameters are NULL. Otherwise it will attempt to use any memory already allocated to av. Any allocated memory should be freed with [sysmem_freeptr\(\)](#).

Parameters

- ac** The address of a variable to hold the number of returned atoms.
- av** The address of a [t_atom](#) pointer to which memory may be allocated and atoms copied.
- fmt** An sprintf-style format string specifying values for the atoms.
- ...** One or more arguments which are to be substituted into the format string.

Returns

A Max error code.

See also

- [atom_getformat\(\)](#)
- [atom_setparse\(\)](#)

33.16.2.31 t_max_err atom_setlong (t_atom *a, long b)

Inserts an integer into a [t_atom](#) and change the [t_atom](#)'s type to [A_LONG](#).

Parameters

- a** Pointer to a [t_atom](#) whose value and type will be modified
- b** Integer value to copy into the [t_atom](#)

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.16.2.32 `t_max_err atom_setlong_array (long ac, t_atom *av, long count, long *vals)`

Assign an array of long integer values to an array of atoms.

Parameters

- ac* The number of atoms to try to fetch from the array of longs. You should have at least this number of atoms allocated in *av*.
- av* The address to the first of an array of allocated atoms.
- count* The number of values in the array specified by *vals*.
- vals* The array from which to copy the values into the array of atoms at *av*.

Returns

A Max error code.

33.16.2.33 `t_max_err atom_setobj (t_atom *a, void *b)`

Inserts a generic pointer value into a `t_atom` and change the `t_atom`'s type to `A_OBJ`.

Parameters

- a* Pointer to a `t_atom` whose value and type will be modified
- b* Pointer value to copy into the `t_atom`

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.16.2.34 `t_max_err atom_setobj_array (long ac, t_atom *av, long count, t_object **vals)`

Assign an array of `t_object*` values to an array of atoms.

Parameters

- ac* The number of atoms to try to fetch from the array of objects. You should have at least this number of atoms allocated in *av*.
- av* The address to the first of an array of allocated atoms.
- count* The number of values in the array specified by *vals*.
- vals* The array from which to copy the values into the array of atoms at *av*.

Returns

A Max error code.

33.16.2.35 `t_max_err atom_setparse (long * ac, t_atom ** av, C74_CONST char * parsestr)`

Parse a C-string into an array of atoms.

This function allocates memory for the atoms if the ac and av parameters are NULL. Otherwise it will attempt to use any memory already allocated to av. Any allocated memory should be freed with `sysmem_freeptr()`.

Parameters

- ac* The address of a variable to hold the number of returned atoms.
- av* The address of a `t_atom` pointer to which memory may be allocated and atoms copied.
- parsestr* The C-string to parse.

Returns

A Max error code.

Remarks

The following example will parse the string "foo bar 1 2 3.0" into an array of 5 atoms. The atom types will be determined automatically as 2 `A_SYM` atoms, 2 `A_LONG` atoms, and 1 `A_FLOAT` atom.

```
t_atom *av = NULL;
long ac = 0;
t_max_err err = MAX_ERR_NONE;

err = atom_setparse(&ac, &av, "foo bar 1 2 3.0");
```

33.16.2.36 `t_max_err atom_setsym (t_atom * a, t_symbol * b)`

Inserts a `t_symbol *` into a `t_atom` and change the `t_atom`'s type to `A_SYM`.

Parameters

- a* Pointer to a `t_atom` whose value and type will be modified
- b* Pointer to a `t_symbol` to copy into the `t_atom`

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.16.2.37 `t_max_err atom_setsym_array (long ac, t_atom * av, long count, t_symbol ** vals)`

Assign an array of `t_symbol*` values to an array of atoms.

Parameters

- ac* The number of atoms to try to fetch from the array of symbols. You should have at least this number of atoms allocated in av.
- av* The address to the first of an array of allocated atoms.
- count* The number of values in the array specified by vals.
- vals* The array from which to copy the values into the array of atoms at av.

Returns

A Max error code.

33.16.2.38 long atomisatomarray (*t_atom* * *a*)

Determines whether or not an atom represents a [t_atomarray](#) object.

Parameters

a The address of the atom to test.

Returns

Returns true if the [t_atom](#) contains a valid [t_atomarray](#) object.

33.16.2.39 long atomisdictionary (*t_atom* * *a*)

Determines whether or not an atom represents a [t_dictionary](#) object.

Parameters

a The address of the atom to test.

Returns

Returns true if the [t_atom](#) contains a valid [t_dictionary](#) object.

33.16.2.40 long atomisstring (*t_atom* * *a*)

Determines whether or not an atom represents a [t_string](#) object.

Parameters

a The address of the atom to test.

Returns

Returns true if the [t_atom](#) contains a valid [t_string](#) object.

33.16.2.41 void postargs (short *argc*, *t_atom* * *argv*)

Print the contents of an array of atoms to the Max window.

Parameters

argc The count of atoms in *argv*.

argv The address to the first of an array of atoms.

33.17 Atombufs

An Atombuf is an alternative to [Binbufs](#) for temporary storage of atoms.

Collaboration diagram for Atombufs:



Data Structures

- `struct t_atombuf`

The atombuf struct provides a way to pass a collection of atoms.

Functions

- `void * atombuf_new (long argc, t_atom *argv)`

Use `atombuf_new()` to create a new Atombuf from an array of t_atoms.

- `void atombuf_free (t_atombuf *x)`

Use `atombuf_free()` to dispose of the memory used by a t_atombuf.

- `void atombuf_text (t_atombuf **x, char **text, long size)`

Use `atombuf_text()` to convert text to a t_atom array in a t_atombuf.

33.17.1 Detailed Description

An Atombuf is an alternative to [Binbufs](#) for temporary storage of atoms. Its principal advantage is that the internal structure is publicly available so you can manipulate the atoms in place. The standard Max text objects (message box, object box, comment) use the Atombuf structure to store their text (each word of text is stored as a [t_symbol](#) or a number).

33.17.2 Function Documentation

33.17.2.1 void atombuf_free (t_atombuf * x)

Use `atombuf_free()` to dispose of the memory used by a t_atombuf.

Parameters

- x The t_atombuf to free.

33.17.2.2 void* atombuf_new (long argc, t_atom * argv)

Use `atombuf_new()` to create a new Atombuf from an array of t_atoms.

Parameters

argc Number of t_atoms in the argv array. May be 0.

argv Array of t_atoms. If creating an empty Atombuf, you may pass 0.

Returns

[atombuf_new\(\)](#) create a new t_atombuf and returns a pointer to it. If 0 is returned, insufficient memory was available.

33.17.2.3 void atombuf_text (t_atombuf *x*, char ***text*, long *size*)**

Use [atombuf_text\(\)](#) to convert text to a t_atom array in a t_atombuf.

To use this routine to create a new Atombuf from the text buffer, first create a new empty t_atombuf with a call to [atombuf_new\(0,NULL\)](#).

Parameters

x Pointer to existing atombuf variable. The variable will be replaced by a new Atombuf containing the converted text.

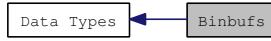
text Handle to the text to be converted. It need not be zero-terminated.

size Number of characters in the text.

33.18 Binbufs

You won't need to know about the internal structure of a Binbuf, so you can use the void * type to refer to one.

Collaboration diagram for Binbufs:



Functions

- `void * binbuf_new (void)`
Use [binbuf_new\(\)](#) to create and initialize a Binbuf.
- `void binbuf_vinsert (void *x, char *fmt,...)`
Use [binbuf_vinsert\(\)](#) to append a Max message to a Binbuf adding a semicolon.
- `void binbuf_insert (void *x, t_symbol *s, short argc, t_atom *argv)`
Use [binbuf_insert\(\)](#) to append a Max message to a Binbuf adding a semicolon.
- `void * binbuf_eval (void *x, short ac, t_atom *av, void *to)`
Use [binbuf_eval](#) to evaluate a Max message in a Binbuf, passing it arguments.
- `short binbuf_getatom (void *x, long *p1, long *p2, t_atom *ap)`
Use [binbuf_getatom](#) to retrieve a single Atom from a Binbuf.
- `short binbuf_text (void *x, char **srcText, long n)`
Use [binbuf_text\(\)](#) to convert a text handle to a Binbuf.
- `short binbuf_totext (void *x, char **dstText, long *sizep)`
Use [binbuf_totext\(\)](#) to convert a Binbuf into a text handle.
- `void binbuf_set (void *x, t_symbol *s, short argc, t_atom *argv)`
Use [binbuf_set\(\)](#) to change the entire contents of a Binbuf.
- `void binbuf_append (void *x, t_symbol *s, short argc, t_atom *argv)`
Use [binbuf_append](#) to append t_atoms to a Binbuf without modifying them.
- `short readatom (char *outstr, char **text, long *n, long e, t_atom *ap)`
Use [readatom\(\)](#) to read a single Atom from a text buffer.

33.18.1 Detailed Description

You won't need to know about the internal structure of a Binbuf, so you can use the void * type to refer to one.

33.18.2 Function Documentation

33.18.2.1 void binbuf_append (void **x*, t_symbol **s*, short *argc*, t_atom **argv*)

Use binbuf_append to append t_atoms to a Binbuf without modifying them.

Parameters

- x* Binbuf to receive the items.
- s* Ignored. Pass NULL.
- argc* Count of items in the argv array.
- argv* Array of atoms to add to the Binbuf.

33.18.2.2 void* binbuf_eval (void **x*, short *ac*, t_atom **av*, void **to*)

Use binbuf_eval to evaluate a Max message in a Binbuf, passing it arguments.

[binbuf_eval\(\)](#) is an advanced function that evaluates the message in a Binbuf with arguments in argv, and sends it to receiver.

Parameters

- x* Binbuf containing the message.
- ac* Count of items in the argv array.
- av* Array of t_atoms as the arguments to the message.
- to* Receiver of the message.

Returns

The result of sending the message.

33.18.2.3 short binbuf_getatom (void **x*, long **p1*, long **p2*, t_atom **ap*)

Use binbuf_getatom to retrieve a single Atom from a Binbuf.

Parameters

- x* Binbuf containing the desired t_atom.
- p1* Offset into the Binbuf's array of types. Modified to point to the next t_atom.
- p2* Offset into the Binbuf's array of data. Modified to point to the next t_atom.
- ap* Location of a t_atom where the retrieved data will be placed.

Returns

1 if there were no t_atoms at the specified offsets, 0 if there's a legitimate t_atom returned in result.

Remarks

To get the first t_atom, set both typeOffset and stuffOffset to 0. Here's an example of getting all the items in a Binbuf:

```
t_atom holder;
long to, so;

to = 0;
so = 0;
while (!binbuf_getatom(x, &to, &so, &holder)) {
    // do something with the t_atom
}
```

33.18.2.4 void binbuf_insert (void *x, t_symbol *s, short argc, t_atom *argv)

Use [binbuf_insert\(\)](#) to append a Max message to a Binbuf adding a semicolon.

Parameters

- x* Binbuf to receive the items.
- s* Ignored. Pass NULL.
- argc* Count of items in the argv array.
- argv* Array of t_atoms to add to the Binbuf.

Remarks

You'll use [binbuf_insert\(\)](#) instead of [binbuf_append\(\)](#) if you were saving your object into a Binbuf and wanted a semicolon at the end. If the message is part of a file that will later be evaluated, such as a Patcher file, the first argument argv[0] will be the receiver of the message and must be a Symbol. [binbuf_vinsert\(\)](#) is easier to use than [binbuf_insert\(\)](#), since you don't have to format your data into an array of Atoms first.

[binbuf_insert\(\)](#) will also convert the t_symbols #1 through #9 into \$1 through \$9. This is used for saving patcher files that take arguments; you will probably never save these symbols as part of anything you are doing.

33.18.2.5 void* binbuf_new (void)

Use [binbuf_new\(\)](#) to create and initialize a Binbuf.

Returns

Returns a new binbuf if successful, otherwise NULL.

33.18.2.6 void binbuf_set (void *x, t_symbol *s, short argc, t_atom *argv)

Use [binbuf_set\(\)](#) to change the entire contents of a Binbuf.

The previous contents of the Binbuf are destroyed.

Parameters

- x* Binbuf to receive the items.
- s* Ignored. Pass NULL.
- argc* Count of items in the argv array.
- argv* Array of t_atoms to put in the Binbuf.

33.18.2.7 short binbuf_text (void *x, char **srcText, long n)

Use [binbuf_text\(\)](#) to convert a text handle to a Binbuf.

[binbuf_text\(\)](#) parses the text in the handle srcText and converts it into binary format. Use it to evaluate a text file or text line entry into a Binbuf.

Parameters

- x* Binbuf to contain the converted text. It must have already been created with binbuf_new. Its previous contents are destroyed.
- srcText* Handle to the text to be converted. It need not be terminated with a 0.
- n* Number of characters in the text.

Returns

If binbuf_text encounters an error during its operation, a non-zero result is returned, otherwise it returns 0.

Remarks

Note: Commas, symbols containing a dollar sign followed by a number 1-9, and semicolons are identified by special pseudo-type constants for you when your text is binbuf-ized.

The following constants in the a_type field of Atoms returned by binbuf_getAtom identify the special symbols **A_SEMI**, **A_COMMA**, and **A_DOLLAR**.

For a **t_atom** of the pseudo-type **A_DOLLAR**, the a_w.w_long field of the **t_atom** contains the number after the dollar sign in the original text or symbol.

Using these pseudo-types may be helpful in separating 'sentences' and 'phrases' in the input language you design. For example, the old pop-up umenu object allowed users to have spaces in between words by requiring the menu items be separated by commas. It's reasonably easy, using [binbuf_getatom\(\)](#), to find the commas in a Binbuf in order to determine the beginning of a new item when reading the atomized text to be displayed in the menu.

If you want to use a literal comma or semicolon in a symbol, precede it with a backslash (\) character. The backslash character can be included by using two backslashes in a row.

33.18.2.8 short binbuf_totext (void *x, char **dstText, long *sizep)

Use [binbuf_totext\(\)](#) to convert a Binbuf into a text handle.

[binbuf_totext\(\)](#) converts a Binbuf into text and places it in a handle. Backslashes are added to protect literal commas and semicolons contained in symbols. The pseudo-types are converted into commas, semicolons, or dollar-sign and number, without backslashes preceding them. binbuf_text can read the output of binbuf_totext and make the same Binbuf.

Parameters

- x* Binbuf with data to convert to text.
- dstText* Pre-existing handle where the text will be placed. dstText will be resized to accomodate the text.
- sizep* Where [binbuf_totext\(\)](#) returns the number of characters in the converted text handle.

Returns

If binbuf_totext runs out of memory during its operation, it returns a non-zero result, otherwise it returns 0.

33.18.2.9 void binbuf_vinsert (void *x, char *fmt, ...)

Use [binbuf_vinsert\(\)](#) to append a Max message to a Binbuf adding a semicolon.

Parameters

- x* Binbuf containing the desired Atom.
- fmt* A C-string containing one or more letters corresponding to the types of each element of the message. s for [t_symbol*](#), l for long, or f for float.
- ... Elements of the message, passed directly to the function as Symbols, longs, or floats.

Remarks

[binbuf_vinsert\(\)](#) works somewhat like a `printf()` for Binbufs. It allows you to pass a number of arguments of different types and insert them into a Binbuf. The entire message will then be terminated with a semicolon. Only 16 items can be passed to [binbuf_vinsert\(\)](#).

The example below shows the implementation of a normal object's save method. The save method requires that you build a message that begins with N (the new object), followed by the name of your object (in this case, represented by the [t_symbol](#) myobject), followed by any arguments your instance creation function requires. In this example, we save the values of two fields m_val1 and m_val2 defined as longs.

```
void myobject_save (myObject *x, Binbuf *dstBuf)
{
    binbuf_vinsert(dstBuf, "ssll", gensym("#N"),
                   gensym("myobject"),
                   x->m_val1, x->m_val2);
}
```

Suppose that such an object had written this data into a file. If you opened the file as text, you would see the following:

```
#N myobject 10 20;
#P newobj 218 82 30 myobject;
```

The first line will result in a new myobject object to be created; the creation function receives the arguments 10 and 20. The second line contains the text of the object box. The newobj message to a patcher creates the object box user interface object and attaches it to the previously created myobject object. Normally, the newex message is used. This causes the object to be created using the arguments that were typed into the object box.

33.18.2.10 short readatom (char *outstr, char **text, long *n, long e, t_atom *ap)

Use [readatom\(\)](#) to read a single Atom from a text buffer.

Parameters

- outstr* C-string of 256 characters that will receive the next text item read from the buffer.
- text* Handle to the text buffer to be read.
- n* Starts at 0, and is modified by readatom to point to the next item in the text buffer.
- e* Number of characters in text.
- ap* Where the resulting Atom read from the text buffer is placed.

Returns

[readatom\(\)](#) returns non-zero if there is more text to read, and zero if it has reached the end of the text. Note that this return value has the opposite logic from that of [binbuf_getatom\(\)](#).

Remarks

This function provides access to the low-level Max text evaluator used by [binbuf_text\(\)](#). It is designed to operate on a handle of characters (text) and called in a loop, as in the example shown below.

```
long index = 0;
t_atom dst;
char outstr[256];

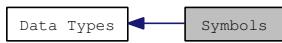
while (readatom(outstr, textHandle, &index, textLength, &dst))
{
    // do something with the resulting Atom
}
```

An alternative to using [readatom](#) is to turn your text into a Binbuf using [binbuf_text\(\)](#), then call [binbuf_getatom\(\)](#) in a loop.

33.19 Symbols

Max maintains a symbol table of all strings to speed lookup for message passing.

Collaboration diagram for Symbols:



Data Structures

- struct [t_symbol](#)

The symbol.

Functions

- [t_symbol * gensym \(C74_CONST char *s\)](#)

Given a C-string, fetch the matching [t_symbol](#) pointer from the symbol table, generating the symbol if necessary.

33.19.1 Detailed Description

Max maintains a symbol table of all strings to speed lookup for message passing. If you want to access the bang symbol for example, you'll have to use the expression `gensym("bang")`. For example, `gensym()` may be needed when sending messages directly to other Max objects such as with `object_method()` and `outlet_anything()`. These functions expect a `t_symbol*`, they don't `gensym()` character strings for you.

The `t_symbol` data structure also contains a place to store an arbitrary value. The following example shows how you can use this feature to use symbols to share values among two different external object classes. (Objects of the same class can use the code resource's global variable space to share data.) The idea is that the `s_thing` field of a `t_symbol` can be set to some value, and `gensym()` will return a reference to the Symbol. Thus, the two classes just have to agree about the character string to be used. Alternatively, each could be passed a `t_symbol` that will be used to share data.

Storing a value:

```
t_symbol *s;
s = gensym("some_weird_string");
s->s_thing = (t_object *)someValue;
```

Retrieving a value:

```
t_symbol *s;
s = gensym("some_weird_string");
someValue = s->s_thing;
```

33.19.2 Function Documentation

33.19.2.1 `t_symbol* gensym (C74_CONST char * s)`

Given a C-string, fetch the matching `t_symbol` pointer from the symbol table, generating the symbol if necessary.

Parameters

`s` A C-string to be looked up in Max's symbol table.

Returns

A pointer to the `t_symbol` in the symbol table.

33.20 Files and Folders

These routines assist your object in opening and saving files, as well as locating the user's files in the Max search path.

Data Structures

- struct [t_fileinfo](#)
Information about a file.
- struct [t_path](#)
The path data structure.
- struct [t_pathlink](#)
The pathlink data structure.

Defines

- #define [MAX_PATH_CHARS](#) 2048
The size you should use when allocating strings for full paths.
- #define [MAX_FILENAME_CHARS](#) 512
The size you should use when allocating strings for filenames.

Typedefs

- typedef void * [t_filehandle](#)
A t_filehandle is a cross-platform way of referring to an open file.

Enumerations

- enum [e_max_path_styles](#) {
 [PATH_STYLE_MAX](#) = 0,
 [PATH_STYLE_NATIVE](#),
 [PATH_STYLE_COLON](#),
 [PATH_STYLE_SLASH](#),
 [PATH_STYLE_NATIVE_WIN](#) }
Constants that determine the output of [path_nameconform\(\)](#).
- enum [e_max_path_types](#) {
 [PATH_TYPE_IGNORE](#) = 0,
 [PATH_TYPE_ABSOLUTE](#),
 [PATH_TYPE_RELATIVE](#),

```
PATH_TYPE_BOOT,
PATH_TYPE_C74,
PATH_TYPE_PATH }
```

Constants that determine the output of [path_nameconform\(\)](#).

- enum e_max_fileinfo_flags {
 PATH_FILEINFO_ALIAS = 1,
 PATH_FILEINFO_FOLDER = 2,
 PATH_FILEINFO_PACKAGE = 4 }

Flags used to represent properties of a file in a [t_fileinfo](#) struct.

- enum e_max_path_folder_flags {
 PATH_REPORTPACKAGEASFOLDER = 1,
 PATH_FOLDER_SNIFF = 2 }

Flags used by functions such as [path_foldernextfile\(\)](#) and [path_openfolder\(\)](#).

- enum e_max_openfile_permissions {
 PATH_READ_PERM = 1,
 PATH_WRITE_PERM = 2,
 PATH_RW_PERM = 3 }

Permissions or mode with which to open a file.

- enum e_max_sysfile_posmodes {
 SYSFILE_ATMARK = 0,
 SYSFILE_FROMSTART,
 SYSFILE_FROMEOF,
 SYSFILE_FROMMARK }

Modes used by [sysfile_setpos\(\)](#).

- enum e_max_sysfile_textflags {
 TEXT_LB_NATIVE = 0x00000001L,
 TEXT_LB_MAC = 0x00000002L,
 TEXT_LB_PC = 0x00000004L,
 TEXT_LB_UNIX = 0x00000008L,
 TEXT_ENCODING_USE_FILE = 0x00000100L,
 TEXT_NULL_TERMINATE = 0x00000200L }

Flags used reading and writing text files.

Functions

- short [path_getapppath](#) (void)
Retrieve the Path ID of the Max application.
- short [locatefile](#) (C74_CONST char *name, short *outvol, short *binflag)

Find a Max document by name in the search path.

- short [locatefiletype](#) (C74_CONST char *name, short *outvol, long filetype, long creator)

Find a Max document by name in the search path.

- short [locatefile_extended](#) (char *name, short *outvol, long *outtype, C74_CONST long *filetypelist, short numtypes)

Find a Max document by name in the search path.

- short [path_resolvefile](#) (char *name, C74_CONST short path, short *outpath)

Resolve a Path ID plus a (possibly extended) file name into a path that identifies the file's directory and a filename.

- short [path_fileinfo](#) (C74_CONST char *name, C74_CONST short path, void *info)

Retrieve a [t_fileinfo](#) structure from a file/path combination.

- short [path_topathname](#) (C74_CONST short path, C74_CONST char *file, char *name)

Create a fully qualified file name from a Path ID/file name combination.

- short [path_frompathname](#) (C74_CONST char *name, short *path, char *filename)

Create a filename and Path ID combination from a fully qualified file name.

- void [path_setdefault](#) (short path, short recursive)

Install a path as the default search path.

- short [path_getdefault](#) (void)

Retrieve the Path ID of the default search path.

- short [path_getmoddate](#) (short path, unsigned long *date)

Determine the modification date of the selected path.

- short [path_getfilemoddate](#) (C74_CONST char *filename, C74_CONST short path, unsigned long *date)

Determine the modification date of the selected file.

- void * [path_openfolder](#) (short path)

Prepare a directory for iteration.

- short [path_foldernextfile](#) (void *xx, long *filetype, char *name, short descend)

Get the next file in the directory.

- void [path_closefolder](#) (void *x)

Complete a directory iteration.

- short [path_opensysfile](#) (C74_CONST char *name, C74_CONST short path, [t_filehandle](#) *ref, short perm)

Open a file given a filename and Path ID.

- short [pathCreatesysfile](#) (C74_CONST char *name, C74_CONST short path, long type, [t_filehandle](#) *ref)

Create a file given a type code, a filename, and a Path ID.

- short [path_nameconform](#) (C74_CONST char *src, char *dst, long style, long type)
Convert a source path string to destination path string using the specified style and type.
- short [path_topotentialname](#) (C74_CONST short path, C74_CONST char *file, char *name, short check)
Create a fully qualified file name from a Path ID/file name combination, regardless of whether or not the file exists on disk.
- short [open_dialog](#) (char *name, short *volptr, long *typeptr, long *types, short ntypes)
Present the user with the standard open file dialog.
- short [saveas_dialog](#) (char *filename, short *path, short *binptr)
Present the user with the standard save file dialog.
- short [saveasdialog_extended](#) (char *name, short *vol, long *type, long *typelist, short numtypes)
Present the user with the standard save file dialog with your own list of file types.
- void [open_promptset](#) (C74_CONST char *s)
Use [open_promptset\(\)](#) to add a prompt message to the open file dialog displayed by [open_dialog\(\)](#).
- void [saveas_promptset](#) (C74_CONST char *s)
Use [saveas_promptset\(\)](#) to add a prompt message to the open file dialog displayed by [saveas_dialog\(\)](#) or [saveasdialog_extended\(\)](#).
- void * [filewatcher_new](#) ([t_object](#) *owner, C74_CONST short path, C74_CONST char *filename)
Create a new filewatcher.
- void [fileusage_addfile](#) (void *w, long flags, C74_CONST char *name, C74_CONST short path)
Add a file to a collective.
- long [sysfile_close](#) ([t_filehandle](#) f)
Close a file opened with [sysfile_open\(\)](#).
- long [sysfile_read](#) ([t_filehandle](#) f, long *count, void *bufptr)
Read a file from disk.
- long [sysfile_readtohandle](#) ([t_filehandle](#) f, char ***h)
Read the contents of a file into a handle.
- long [sysfile_readtoptr](#) ([t_filehandle](#) f, char **p)
Read the contents of a file into a pointer.
- long [sysfile_write](#) ([t_filehandle](#) f, long *count, const void *bufptr)
Write part of a file to disk.
- long [sysfile_seteof](#) ([t_filehandle](#) f, long logeof)
Set the size of a file handle.

- long [sysfile_geteof](#) ([t_filehandle](#) f, long *logeof)
Get the size of a file handle.
- long [sysfile_setpos](#) ([t_filehandle](#) f, long mode, long offset)
Set the current file position of a file handle.
- long [sysfile_getpos](#) ([t_filehandle](#) f, long *filepos)
Get the current file position of a file handle.
- long [sysfile_spoolcopy](#) ([t_filehandle](#) src, [t_filehandle](#) dst, long size)
Copy the contents of one file handle to another file handle.
- long [sysfile_readtextfile](#) ([t_filehandle](#) f, [t_handle](#) htext, long maxlen, long flags)
Read a text file from disk.
- long [sysfile_writetextfile](#) ([t_filehandle](#) f, [t_handle](#) htext, long flags)
Write a text file to disk.
- short [sysfile_openhandle](#) (char **h, long flags, [t_filehandle](#) *fh)
Create a [t_filehandle](#) from a pre-existing handle.
- short [sysfile_openptrsize](#) (char *p, long length, long flags, [t_filehandle](#) *fh)
Create a [t_filehandle](#) from a pre-existing pointer.

33.20.1 Detailed Description

These routines assist your object in opening and saving files, as well as locating the user's files in the Max search path. There have been a significant number of changes to these routines (as well as the addition of many functions), so some history may be useful in understanding their use.

Prior to version 4, Max used a feature of Mac OS 9 called "working directories" to specify files. When you used the [locatefile\(\)](#) service routine, you would get back a file name and a volume number. This name (converted to a Pascal string) and the volume number could be passed to FSOpen() to open the located file for reading. The [open_dialog\(\)](#) routine worked similarly.

In Mac OSX, working directories are no longer supported. In addition, the use of these "volume" numbers makes it somewhat difficult to port Max file routines to other operating systems, such as Windows XP, that specify files using complete pathnames (i.e., "C:\dir1\dir2\file.pat").

However, it is useful to be able to refer to the path and the name of the file separately. The solution involves the retention of the volume number (now called Path ID), but with a platform-independent wrapper that determines its meaning. There are now calls to locate, open, and choose files using C filename strings and Path IDs, as well as routines to convert between a "native" format for specifying a file (such as a full pathname on Windows or an FSRef on the Macintosh) to the C string and Path ID. As of Max version 5 FSSpecs, long ago deprecated by Apple, are no longer supported.

Now that paths in Max have changed to use the slash style, as opposed to the old Macintosh colon style (see the Max 4.3 documentation for a description of the file path styles), there is one function in particular that you will find useful for converting between the various ways paths can be represented, including operating system native paths. This function is [path_nameconform\(\)](#). Note that for compatibility purposes Path API functions accept paths in any number of styles, but will typically return paths, or modify paths inline to use the newer slash style. In addition to absolute paths, paths relative to the Max Folder, the "Cycling

'74" folder and the boot volume are also supported. See the conformpath.help and ext_path.h files for more information on the various styles and types of paths. See the "filebyte" SDK example project for a demonstration of how to use the path functions to convert a Max name and path ref pair to a Windows native path for use with CreateFile().

There are a large number of service routine in the Max 4 kernel that support files, but only a handful will be needed by most external objects. In addition to the descriptions that follow, you should consult the movie, folder and filedate examples included with the SDK.

33.20.2 The Sysfile API

The Sysfile API provides the means of reading and writing files opened by [path_createsysfile\(\)](#) and similar. These functions all make use of an opaque structure, [t_filehandle](#). See the path functions [path_opensysfile\(\)](#) and [path_createsysfile\(\)](#) described earlier in this chapter for more information. The Sysfile API is relatively similar to parts of the old Macintosh File Manager API, and not too different from Standard C library file functions. The "filebyte" example project in the SDK shows how to use these functions to read from a file. It is not safe to mix these routines with other file routines (e.g. don't use fopen() to open a file and [sysfile_close\(\)](#) to close it).

In addition to being able to use these routines to write cross-platform code in your max externals, another advantage of the Sysfile API is that it is able to read files stored in the collective file format on both Windows XP and Mac OSX.

33.20.3 Example: filebyte (notes from the IRCAM workshop)

33.20.3.1 Paths

- A number that specifies a file location
- Returned by [locatefile_extended\(\)](#) and [open_dialog\(\)](#)
- Supply a path when opening a file with [path_opensysfile\(\)](#)
- Can convert path to and from pathname

33.20.3.2 t_filehandle

- Returned by [path_opensysfile](#)
- Refers to an open file you want to read or write using [sysfile_read](#) / [sysfile_write](#)
- Could refer to a file in a collective

33.20.3.3 File Names

- C string
- Max 5 filenames are UTF-8
- Max 5 supports long (unicode) filenames on both Mac and Windows

33.20.3.4 File Path Names

- Max uses a platform-independent path string format: volume:/path1/path2/filename returned by path_topathname
- Can convert to platform-specific format using path_nameconform (not needed if using path_opensysfile)
- Platform-independent format must be used with path_frompathname

33.20.4 Collectives and Fileusage

Use the fileusage routines to add files to a collective when a user chooses to build a collective. Your object can respond to a "fileusage" message, which is sent by Max when the collective builder is building a collective using the following:

```
class_addmethod(c, (method)my_fileusage, "fileusage", A_CANT, 0L);
```

Where my file usage has the prototype:

```
void my_fileusage(t_myObject **x, void *w);
```

Then you can use [fileusage_addfile\(\)](#) to add any requisite files to the collective.

33.20.5 Filewatchers

Your object can watch a file or folder and be notified of changes. Use [filewatcher_new\(\)](#), [filewatcher_start\(\)](#), and [filewatcher_stop\(\)](#) to implement this functionality. You may wish to use filewatchers sparingly as they can potentially incur computational overhead in the background.

33.20.6 Define Documentation

33.20.6.1 #define MAX_FILENAME_CHARS 512

The size you should use when allocating strings for filenames.

At the time of this writing it supports up to 256 UTF chars

33.20.7 Typedef Documentation

33.20.7.1 `typedef void* t_filehandle`

A t_filehandle is a cross-platform way of referring to an open file.

It is an opaque structure, meaning you don't have access to the individual elements of the data structure. You can use a t_filehandle only with the file routines in the Sysfile API. Do not use other platform-specific file functions in conjunction with these functions. The perm parameter can be either READ_PERM, WRITE_PERM, or RW_PERM.

33.20.8 Enumeration Type Documentation

33.20.8.1 enum e_max_fileinfo_flags

Flags used to represent properties of a file in a [t_fileinfo](#) struct.

Enumerator:

PATH_FILEINFO_ALIAS alias
PATH_FILEINFO_FOLDER folder
PATH_FILEINFO_PACKAGE package (Mac-only)

33.20.8.2 enum e_max_openfile_permissions

Permissions or mode with which to open a file.

Enumerator:

PATH_READ_PERM Read mode.
PATH_WRITE_PERM Write mode.
PATH_RW_PERM Read/Write mode.

33.20.8.3 enum e_max_path_folder_flags

Flags used by functions such as [path_foldernextfile\(\)](#) and [path_openfolder\(\)](#).

Enumerator:

PATH_REPORTPACKAGEASFOLDER if not true, then a Mac OS package will be reported as a file rather than a folder.
PATH_FOLDER_SNIFF sniff

33.20.8.4 enum e_max_path_styles

Constants that determine the output of [path_nameconform\(\)](#).

See also

[e_max_path_types](#)
[path_nameconform\(\)](#)

Enumerator:

PATH_STYLE_MAX use PATH_STYLE_MAX_PLAT
PATH_STYLE_NATIVE use PATH_STYLE_NATIVE_PLAT
PATH_STYLE_COLON ':' sep, "vol:" volume, ":" relative, "^:" boot
PATH_STYLE_SLASH '/' sep, "vol://" volume, "./" relative, "/" boot
PATH_STYLE_NATIVE_WIN '\' sep, "vol:\\" volume, ".\\\" relative, "\\\" boot

33.20.8.5 enum e_max_path_types

Constants that determine the output of [path_nameconform\(\)](#).

See also

[e_max_path_styles](#)
[path_nameconform\(\)](#)

Enumerator:

PATH_TYPE_IGNORE ignore
PATH_TYPE_ABSOLUTE absolute path
PATH_TYPE_RELATIVE relative path
PATH_TYPE_BOOT boot path
PATH_TYPE_C74 Cycling '74 folder.
PATH_TYPE_PATH path

33.20.8.6 enum e_max_sysfile_posmodes

Modes used by [sysfile_setpos\(\)](#).

Enumerator:

SYSFILE_ATMARK ?
SYSFILE_FROMSTART Calculate the file position from the start of the file.
SYSFILE_FROMLEOF Calculate the file position from the logical end of the file.
SYSFILE_FROMMMARK Calculate the file position from the current file position.

33.20.8.7 enum e_max_sysfile_textflags

Flags used reading and writing text files.

Enumerator:

TEXT_LB_NATIVE Use the linebreak format native to the current platform.
TEXT_LB_MAC Use Macintosh line breaks.
TEXT_LB_PC Use Windows line breaks.
TEXT_LB_UNIX Use Unix line breaks.
TEXT_ENCODING_USE_FILE If this flag is not set then the encoding is forced to UTF8.
TEXT_NULL_TERMINATE Terminate memory returned from [sysfile_readtextfile\(\)](#) with a NULL character.

33.20.9 Function Documentation

33.20.9.1 void fileusage_addfile (void * *w*, long *flags*, C74_CONST char * *name*, C74_CONST short *path*)

Add a file to a collective.

Parameters

w Handle for the collective builder.

flags If flags == 1, copy this file to support folder of an app instead of to the collective in an app.

name The name of the file.

path The path of the file to add.

33.20.9.2 void* filewatcher_new (t_object * *owner*, C74_CONST short *path*, C74_CONST char * *filename*)

Create a new filewatcher.

The file will not be actively watched until filewatcher_start() is called. The filewatcher can be freed using [object_free\(\)](#).

Parameters

owner Your object. This object will receive the message "filechanged" when the watcher sees a change in the file or folder.

path The path in which the file being watched resides, or the path of the folder being watched.

filename The name of the file being watched, or an empty string if you are simply watching the folder specified by path.

Returns

A pointer to the new filewatcher.

Remarks

The "filechanged" method should have the prototype:

```
void myObject_filechanged(t_myObject *x, char *filename, short path);
```

33.20.9.3 short locatefile (C74_CONST char * *name*, short * *outvol*, short * *binflag*)

Find a Max document by name in the search path.

This routine performs the same function as the routine [path_getdefault\(\)](#). [locatefile\(\)](#) searches through the directories specified by the user for Patcher files and tables in the File Preferences dialog as well as the current default path (see [path_getdefault](#)) and the directory containing the Max application

Parameters

name A C string that is the name of the file to look for.

outvol The Path ID containing the location of the file if it is found.

binflag If the file found is in binary format (it's of type 'maxb') 1 is returned here; if it's in text format, 0 is returned.

Returns

If a file is found with the name specified by filename, locatefile returns 0, otherwise it returns non-zero.

Remarks

filename and vol can then be passed to binbuf_read to read and open file the file. When using MAXplay, the search path consists of all subdirectories of the directory containing the MAXplay application. locatefile only searches for files of type 'maxb' and 'TEXT.'

See also

[locatefile_extended\(\)](#)

33.20.9.4 short locatefile_extended (char * name, short * outvol, long * outtype, C74_CONST long * filerlist, short numtypes)

Find a Max document by name in the search path.

This is the preferred method for file searching since its introduction in Max version 4.

This routine performs the same function as the routine [path_getdefault\(\)](#). [locatefile\(\)](#) searches through the directories specified by the user for Patcher files and tables in the File Preferences dialog as well as the current default path (see [path_getdefault\(\)](#)) and the directory containing the Max application

Version

4.0

Parameters

name The file name for the search, receives actual filename.

outvol The Path ID of the file (if found).

outtype The file type of the file (if found).

filerlist The file type(s) that you are searching for.

numtypes The number of file types in the typelist array (1 if a single entry).

Returns

If a file is found with the name specified by filename, locatefile returns 0, otherwise it returns non-zero.

Remarks

The old file search routines [locatefile\(\)](#) and [locatefiletype\(\)](#) are still supported in Max 4, but the use of a new routine [locatefile_extended\(\)](#) is highly recommended. However, [locatefile_extended\(\)](#) has an important difference from [locatefile\(\)](#) and [locatefiletype\(\)](#) that may require some rewriting of your code. *It modifies its name parameter* in certain cases, while [locatefile\(\)](#) and [locatefiletype\(\)](#) do not. The two cases where it could modify the incoming filename string are 1) when an alias is specified, the file pointed to by the alias is returned; and 2) when a full path is specified, the output is the filename plus the path number of the folder it's in.

This is important because many people pass the s_name field of a [t_symbol](#) to [locatefile\(\)](#). If the name field of a [t_symbol](#) were to be modified, the symbol table would be corrupted. To avoid this problem, use [strncpy_zero\(\)](#) to copy the contents of a [t_symbol](#) to a character string first, as shown below:

```
char filename[MAX_FILENAME_CHARS];
strncpy_zero(filename,str->s_name, MAX_FILENAME_CHARS);
result = locatefile_extended(filename,&path,&type,typelist,1);
```

33.20.9.5 short locatefiletype (C74_CONST char * *name*, short * *outvol*, long *filetype*, long *creator*)

Find a Max document by name in the search path.

This function searches through the same directories as locatefile, but allows you to specify a type and creator of your own.

Parameters

name A C string that is the name of the file to look for.

outvol The Path ID containing the location of the file if it is found.

filetype The filetype of the file to look for. If you pass 0L, files of all filetypes are considered.

creator The creator of the file to look for. If you pass 0L, files with any creator are considered.

Returns

If a file is found with the name specified by filename, locatefile returns 0, otherwise it returns non-zero.

See also

[locatefile_extended\(\)](#)

33.20.9.6 short open_dialog (char * *name*, short * *volptr*, long * *typeptr*, long * *types*, short *ntypes*)

Present the user with the standard open file dialog.

This function is convenient wrapper for using Mac OS Navigation Services or Standard File for opening files.

The mapping of extensions to types is configured in the C74:/init/max-fileformats.txt file. The standard types to use for Max files are 'maxb' for old-format binary files, 'TEXT' for text files, and 'JSON' for newer format patchers or other .json files.

Parameters

name A C-string that will receive the name of the file the user wants to open. The C-string should be allocated with a size of at least [MAX_FILENAME_CHARS](#).

volptr Receives the Path ID of the file the user wants to open.

typeptr The file type of the file the user wants to open.

types A list of file types to display. This is not limited to 4 types as in the SFGetFile() trap. Pass NULL to display all types.

ntypes The number of file types in typelist. Pass 0 to display all types.

Returns

0 if the user clicked Open in the dialog box. If the user cancelled, [open_dialog\(\)](#) returns a non-zero value.

See also

[saveasdialog_extended\(\)](#)
[locatefile_extended\(\)](#)

33.20.9.7 void open_promptset (C74_CONST char * s)

Use [open_promptset\(\)](#) to add a prompt message to the open file dialog displayed by [open_dialog\(\)](#).

Calling this function before [open_dialog\(\)](#) permits a string to be displayed in the dialog box instructing the user as to the purpose of the file being opened. It will only apply to the call of [open_dialog\(\)](#) that immediately follows [open_promptset\(\)](#).

Parameters

s A C-string containing the prompt you wish to display in the dialog box.

Returns

Ignore.

See also

[open_dialog\(\)](#)

33.20.9.8 void path_closefolder (void * x)

Complete a directory iteration.

Parameters

x The "folder state" value originally returned by [path_openfolder\(\)](#).

33.20.9.9 short path_createssysfile (C74_CONST char * name, C74_CONST short path, long type, t_filehandle * ref)

Create a file given a type code, a filename, and a Path ID.

Parameters

name The name of the file to be opened.

path The Path ID of the file to be opened.

type The file type of the created file.

ref A [t_filehandle](#) reference to the opened file will be returned in this parameter.

Returns

An error code.

33.20.9.10 short path_fileinfo (C74_CONST char * name, C74_CONST short path, void * info)

Retrive a [t_fileinfo](#) structure from a file/path combination.

Parameters

name The file name to be queried.

path The Path ID of the file.

info The address of a [t_fileinfo](#) structure to contain the file information.

Returns

Returns 0 if successful, otherwise it returns an OS-specific error code.

33.20.9.11 short path_foldernextfile (void * xx, long * filetype, char * name, short descend)

Get the next file in the directory.

In conjunction with [path_openfolder\(\)](#) and [path_closefolder\(\)](#), this routine allows you to iterate through all of the files in a path.

Parameters

xx The "folder state" value returned by [path_openfolder\(\)](#).

filetype Contains the file type of the file type on return.

name Contains the file name of the next file on return.

descend Unused.

Returns

Returns non-zero if successful, and zero when there are no more files.

See also

[e_max_path_folder_flags](#)

33.20.9.12 short path_from pathname (C74_CONST char * name, short * path, char * filename)

Create a filename and Path ID combination from a fully qualified file name.

Note that [path_from pathname\(\)](#) does not require that the file actually exist. In this way you can use it to convert a full path you may have received as an argument to a file writing message to a form appropriate to provide to a routine such as [path_createfile\(\)](#).

Parameters

name The extended file path to be converted.

path Contains the Path ID on return.

filename Contains the file name on return.

Returns

Returns 0 if successful, otherwise it returns an OS-specific error code.

33.20.9.13 short path_getapppath (void)

Retrieve the Path ID of the Max application.

Returns

The path id.

33.20.9.14 short path_getdefault (void)

Retrieve the Path ID of the default search path.

Returns

The path id of the default search path.

33.20.9.15 short path_getfilemoddate (C74_CONST char *filename, C74_CONST short path, unsigned long *date)

Determine the modification date of the selected file.

Parameters

filename The name of the file to query.

path The Path ID of the file.

date The last modification date of the file upon return.

Returns

An error code.

33.20.9.16 short path_getmoddate (short path, unsigned long *date)

Determine the modification date of the selected path.

Parameters

path The Path ID of the directory to check.

date The last modification date of the directory.

Returns

An error code.

33.20.9.17 short path_nameconform (C74_CONST char *src, char *dst, long style, long type)

Convert a source path string to destination path string using the specified style and type.

Parameters

src A pointer to source character string to be converted.

dst A pointer to destination character string.

style The destination filepath style, as defined in [e_max_path_styles](#)

type The destination filepath type, as defined in [e_max_path_types](#)

Returns

An error code.

See also

[MAX_PATH_CHARS](#)

33.20.9.18 void* path_openfolder (short *path*)

Prepare a directory for iteration.

Parameters

path The directory Path ID to open.

Returns

The return value of this routine is an internal "folder state" structure used for further folder manipulation. It should be saved and used for calls to [path_foldernextfile\(\)](#) and [path_closefolder\(\)](#). If the folder cannot be found or accessed, [path_openfolder\(\)](#) returns 0.

33.20.9.19 short path_opensysfile (C74_CONST char * *name*, C74_CONST short *path*, t_filehandle * *ref*, short *perm*)

Open a file given a filename and Path ID.

Parameters

name The name of the file to be opened.

path The Path ID of the file to be opened.

ref A [t_filehandle](#) reference to the opened file will be returned in this parameter.

perm The permission for the opened file as defined in [e_max_openfile_permissions](#).

Returns

An error code.

33.20.9.20 short path_resolvefile (char * *name*, C74_CONST short *path*, short * *outpath*)

Resolve a Path ID plus a (possibly extended) file name into a path that identifies the file's directory and a filename.

This routine converts a name and Path ID to a standard form in which the name has no path information and does not refer to an aliased file.

Parameters

- name*** A file name (which may be fully or partially qualified), will contain the file name on return.
path The Path ID to be resolved.
outpath The Path ID of the returned file name.

Returns

Returns 0 if successful.

33.20.9.21 void path_setdefault (short *path*, short *recursive*)

Install a path as the default search path.

The default path is searched before the Max search path. For instance, when loading a patcher from a directory outside the search path, the patcher's directory is searched for files before the search path. [path_setdefault\(\)](#) allows you to set a path as the default.

Parameters

- path*** The path to use as the search path. If path is already part of the Max Search path, it will not be added (since, by default, it will be searched during file searches).
recursive If non-zero, all subdirectories will be installed in the default search list. Be very careful with the use of the recursive argument. It has the capacity to slow down file searches dramatically as the list of folders is being built. Max itself never creates a hierarchical default search path.

33.20.9.22 short path_topathname (C74_CONST short *path*, C74_CONST char **file*, char **name*)

Create a fully qualified file name from a Path ID/file name combination.

Unlike [path_topotentialname\(\)](#), this routine will only convert a pathname pair to a valid path string if the path exists.

Parameters

- path*** The path to be used.
file The file name to be used.
name Loaded with the fully extended file name on return.

Returns

Returns 0 if successful, otherwise it returns an OS-specific error code.

33.20.9.23 short path_topotentialname (C74_CONST short *path*, C74_CONST char **file*, char **name*, short *check*)

Create a fully qualified file name from a Path ID/file name combination, regardless of whether or not the file exists on disk.

Parameters

- path*** The path to be used.

file The file name to be used.

name Loaded with the fully extended file name on return.

check Flag to check if a file with the given path exists.

Returns

Returns 0 if successful, otherwise it returns an OS-specific error code.

See also

[path_topathname\(\)](#)

33.20.9.24 short saveas_dialog (char *filename, short *path, short *binptr)

Present the user with the standard save file dialog.

The mapping of extensions to types is configured in the C74:/init/max-fileformats.txt file. The standard types to use for Max files are 'maxb' for old-format binary files, 'TEXT' for text files, and 'JSON' for newer format patchers or other .json files.

Parameters

filename A C-string containing a default name for the file to save. If the user decides to save a file, its name is returned here. The C-string should be allocated with a size of at least [MAX_FILENAME_CHARS](#).

path If the user decides to save the file, the Path ID of the location chosen is returned here.

binptr Pass NULL for this parameter. This parameter was used in Max 4 to allow the choice of saving binary or text format patchers.

Returns

0 if the user choose to save the file. If the user cancelled, returns a non-zero value.

See also

[open_dialog\(\)](#)
[saveasdialog_extended\(\)](#)
[locatefile_extended\(\)](#)

33.20.9.25 void saveas_promptset (C74_CONST char * s)

Use [saveas_promptset\(\)](#) to add a prompt message to the open file dialog displayed by [saveas_dialog\(\)](#) or [saveasdialog_extended\(\)](#).

Calling this function before [saveasdialog_extended\(\)](#) permits a string to be displayed in the dialog box instructing the user as to the purpose of the file being opened. It will only apply to the call of [saveasdialog_extended\(\)](#) that immediately follows [saveas_promptset\(\)](#).

Parameters

s A C-string containing the prompt you wish to display in the dialog box.

Returns

Ignore.

See also[open_dialog\(\)](#)**33.20.9.26 short saveasdialog_extended (char * name, short * vol, long * type, long * typelist, short numtypes)**

Present the user with the standard save file dialog with your own list of file types.

[saveasdialog_extended\(\)](#) is similar to [saveas_dialog\(\)](#), but allows the additional feature of specifying a list of possible types. These will be displayed in a pop-up menu.

File types found in the typelist argument that match known Max types will be displayed with descriptive text. Unmatched types will simply display the type name (for example, "foXx" is not a standard type so it would be shown in the pop-up menu as foXx)

Known file types include:

- TEXT: text file
- maxb: Max binary patcher
- maxc: Max collective
- Midi: MIDI file
- Sd2f: Sound Designer II audio file
- NxTS: NeXT/Sun audio file
- WAVE: WAVE audio file.
- AIFF: AIFF audio file
- mP3f: Max preference file
- PICT: PICT graphic file
- MooV: Quicktime movie file
- aPcs: VST plug-in
- AFxP: VST effect patch data file
- AFxB: VST effect bank data file
- DATA: Raw data audio file
- ULAW: NeXT/Sun audio file

Parameters

name A C-string containing a default name for the file to save. If the user decides to save a file, its name is returned here. The C-string should be allocated with a size of at least [MAX_FILENAME_CHARS](#).

vol If the user decides to save the file, the Path ID of the location chosen is returned here.

type Returns the type of file chosen by the user.

typelist The list of types provided to the user.

numtypes The number of file types in typelist.

Returns

0 if the user choose to save the file. If the user cancelled, returns a non-zero value.

See also

[open_dialog\(\)](#)
[locatefile_extended\(\)](#)

33.20.9.27 long sysfile_close (*t_filehandle f*)

Close a file opened with sysfile_open().

This function is similar to FSClose() or fclose(). It should be used instead of system-specific file closing routines in order to make max external code that will compile cross-platform.

Parameters

f The [*t_filehandle*](#) structure of the file the user wants to close.

Returns

An error code.

33.20.9.28 long sysfile_geteof (*t_filehandle f, long * logeof*)

Get the size of a file handle.

This function is similar to and should be used instead of GetEOF(). The function gets the size of file handle in bytes, and places it in "logeof".

Parameters

f The file's [*t_filehandle*](#) structure.

logeof The file size in bytes is returned to this value.

Returns

An error code.

33.20.9.29 long sysfile_getpos (*t_filehandle f, long *filepos*)

Get the current file position of a file handle.

This function is similar to and should be used instead of GetFPos(). The function gets the current file position of file handle in bytes, and places it in "filepos".

Parameters

f The file's [*t_filehandle*](#) structure.

filepos The address of a variable to hold the current file position of file handle in bytes.

Returns

An error code.

33.20.9.30 short sysfile_openhandle (char ***h*, long*flags*, t_filehandle **fh*)

Create a t_filehandle from a pre-existing handle.

Parameters

- h* A handle for some data.
- flags* Pass 0 (additional flags are private).
- fh* The address of a t_filehandle which will be allocated.

Returns

An error code.

33.20.9.31 short sysfile_openptrsize (char **p*, long*length*, long*flags*, t_filehandle **fh*)

Create a t_filehandle from a pre-existing pointer.

Parameters

- p* A pointer to some data.
- length* The size of p.
- flags* Pass 0 (additional flags are private).
- fh* The address of a t_filehandle which will be allocated.

Returns

An error code.

33.20.9.32 long sysfile_read (t_filehandle*f*, long **count*, void **bufptr*)

Read a file from disk.

This function is similar to FSRead() or fread(). It should be used instead of these functions (or other system-specific file reading routines) in order to make max external code that will compile cross-platform. It reads “count” bytes from file handle at current file position into “bufptr”. The byte count actually read is set in “count”, and the file position is updated by the actual byte count read.

Parameters

- f* The t_filehandle structure of the file the user wants to open.
- count* Pointer to the number of bytes that will be read from the file at the current file position. The byte count actually read is returned to this value.
- bufptr* Pointer to the buffer that the data will be read into.

Returns

An error code.

33.20.9.33 long sysfile_readtextfile (t_filehandle *f*, t_handle *htext*, long *maxlen*, long *flags*)

Read a text file from disk.

This function reads up to the maximum number of bytes given by maxlen from file handle at current file position into the htext handle, performing linebreak translation if set in flags.

Parameters

f The [t_filehandle](#) structure of the text file the user wants to open.

htext Handle that the data will be read into.

maxlen The maximum length in bytes to be read into the handle. Passing the value 0L indicates no maximum (i.e. read the entire file).

flags Flags to set linebreak translation as defined in [e_max_sysfile_textflags](#).

Returns

An error code.

33.20.9.34 long sysfile_readtohandle (t_filehandle *f*, char * *h*)**

Read the contents of a file into a handle.

Parameters

f The open [t_filehandle](#) structure to read into the handle.

h The address of a handle into which the file will be read.

Returns

An error code.

Remarks

You should free the pointer, when you are done with it, using [sysmem_freehandle\(\)](#).

33.20.9.35 long sysfile_readtoptr (t_filehandle *f*, char ** *p*)

Read the contents of a file into a pointer.

Parameters

f The open [t_filehandle](#) structure to read into the handle.

p The address of a pointer into which the file will be read.

Returns

An error code.

Remarks

You should free the pointer, when you are done with it, using [sysmem_freetoptr\(\)](#).

33.20.9.36 long sysfile_seteof (*t_filehandle f*, *long logeof*)

Set the size of a file handle.

This function is similar to and should be used instead of SetEOF(). The function sets the size of file handle in bytes, specified by “*logeof*”.

Parameters

- f* The file’s [t_filehandle](#) structure.
- logeof* The file size in bytes.

Returns

An error code.

33.20.9.37 long sysfile_setpos (*t_filehandle f*, *long mode*, *long offset*)

Set the current file position of a file handle.

This function is similar to and should be used instead of SetFPos(). It is used to set the current file position of file handle to by the given number of offset bytes relative to the mode used, as defined in [e_max_sysfile_posmodes](#).

Parameters

- f* The file’s [t_filehandle](#) structure.
- mode* Mode from which the offset will be calculated, as defined in [e_max_sysfile_posmodes](#).
- offset* The offset in bytes relative to the mode.

Returns

An error code.

33.20.9.38 long sysfile_spoolcopy (*t_filehandle src*, *t_filehandle dst*, *long size*)

Copy the contents of one file handle to another file handle.

Parameters

- src* The file handle from which to copy.
- dst* The file handle to which the copy is written.
- size* The number of bytes to copy. If 0 the size of src will be used.

Returns

An error code.

33.20.9.39 long sysfile_write (t_filehandle *f*, long * *count*, const void * *bufptr*)

Write part of a file to disk.

This function is similar to FSWrite() or fwrite(). It should be used instead of these functions (or other system-specific file reading routines) in order to make max external code that will compile cross-platform. The function writes “count” bytes from “bufptr” into file handle at current file position. The byte count actually written is set in “count”, and the file position is updated by the actual byte count written.

Parameters

f The t_filehandle structure of the file to which the user wants to write.

count Pointer to the number of bytes that will be written to the file at the current file position. The byte count actually written is returned to this value.

bufptr Pointer to the buffer that the data will be read from.

Returns

An error code.

33.20.9.40 long sysfile_writetextfile (t_filehandle *f*, t_handle *htext*, long *flags*)

Write a text file to disk.

This function writes a text handle to a text file performing linebreak translation if set in flags.

Parameters

f The t_filehandle structure of the text file to which the user wants to write.

htext Handle that the data that will be read from.

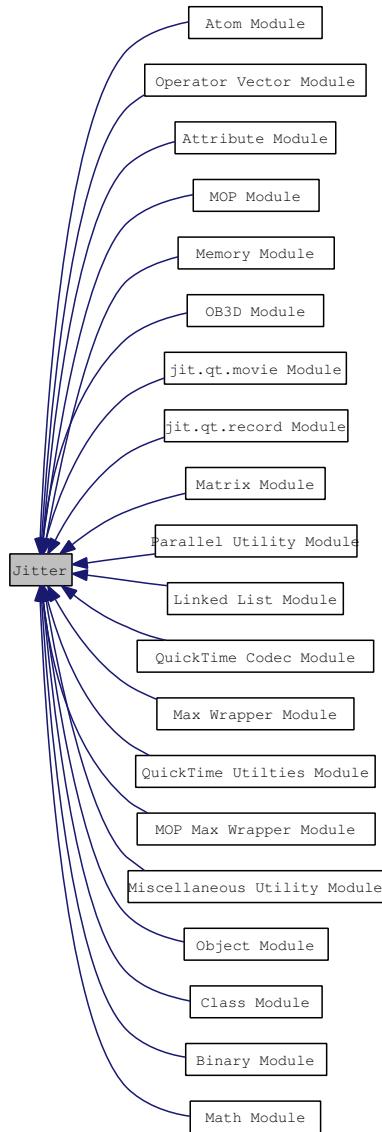
flags Flags to set linebreak translation as defined in [e_max_sysfile_textflags](#).

Returns

An error code.

33.21 Jitter

Collaboration diagram for Jitter:



Modules

- Atom Module
- Attribute Module
- Binary Module
- Class Module
- Object Module
- Miscellaneous Utility Module
- Linked List Module
- Math Module

- Matrix Module
- Max Wrapper Module
- Memory Module
- MOP Module
- Parallel Utility Module
- MOP Max Wrapper Module
- OB3D Module
- Operator Vector Module
- QuickTime Codec Module
- jit.qt.movie Module
- jit.qt.record Module
- QuickTime Utilities Module

Defines

- #define **JIT_ATTR_GET_OPAQUE** 0x00000001
private getter (all)
- #define **JIT_ATTR_SET_OPAQUE** 0x00000002
private setter (all)
- #define **JIT_ATTR_GET_OPAQUE_USER** 0x00000100
private getter (user)
- #define **JIT_ATTR_SET_OPAQUE_USER** 0x00000200
private setter (user)
- #define **JIT_ATTR_GET_DEFER** 0x00010000
defer getter (deprecated)
- #define **JIT_ATTR_GET_USURP** 0x00020000
usurp getter (deprecated)
- #define **JIT_ATTR_GET_DEFER_LOW** 0x00040000
defer getter
- #define **JIT_ATTR_GET_USURP_LOW** 0x00080000
usurp getter
- #define **JIT_ATTR_SET_DEFER** 0x01000000
defer setter (deprecated)
- #define **JIT_ATTR_SET_USURP** 0x02000000
usurp setter (deprecated)
- #define **JIT_ATTR_SET_DEFER_LOW** 0x04000000
defer setter
- #define **JIT_ATTR_SET_USURP_LOW** 0x08000000

usurp setter

- #define **JIT_MATRIX_DATA_HANDLE** 0x00000002
data is handle
- #define **JIT_MATRIX_DATA_REFERENCE** 0x00000004
data is reference to outside memory
- #define **JIT_MATRIX_DATA_PACK_TIGHT** 0x00000008
data is tightly packed (doesn't use standard 16 byte alignment)
- #define **JIT_MATRIX_DATA_FLAGS_USE** 0x00008000
necessary if using handle/reference data flags when creating jit_matrix, however, it is never stored in matrix
- #define **JIT_MATRIX_MAX_DIMCOUNT** 32
maximum dimension count
- #define **JIT_MATRIX_MAX_PLANECOUNT** 32
maximum plane count
- #define **JIT_MATRIX_CONVERT_CLAMP** 0x00000001
not currently used
- #define **JIT_MATRIX_CONVERT_INTERP** 0x00000002
use interpolation
- #define **JIT_MATRIX_CONVERT_SRCDIM** 0x00000004
use source dimensions
- #define **JIT_MATRIX_CONVERT_DSTDIM** 0x00000008
use destination dimensions
- #define **JIT_OB3D_NO_ROTATION_SCALE** 1 << 0
ob3d flag
- #define **JIT_OB3D_NO_POLY_VARS** 1 << 1
ob3d flag
- #define **JIT_OB3D_NO_BLEND** 1 << 2
ob3d flag
- #define **JIT_OB3D_NO_TEXTURE** 1 << 3
ob3d flag
- #define **JIT_OB3D_NO_MATRIXOUTPUT** 1 << 4
ob3d flag
- #define **JIT_OB3D_AUTO_ONLY** 1 << 5
ob3d flag

- #define **JIT_OB3D_DOES_UI** 1 << 6
ob3d flag
- #define **JIT_OB3D_NO_DEPTH** 1 << 7
ob3d flag
- #define **JIT_OB3D_NO_ANTIALIAS** 1 << 8
ob3d flag
- #define **JIT_OB3D_NO_FOG** 1 << 9
ob3d flag
- #define **JIT_OB3D_NO_LIGHTING_MATERIAL** 1 << 10
ob3d flag
- #define **JIT_OB3D_HAS_LIGHTS** 1 << 11
ob3d flag
- #define **JIT_OB3D_HAS_CAMERA** 1 << 12
ob3d flag
- #define **JIT_OB3D_IS_RENDERER** 1 << 13
ob3d flag
- #define **JIT_OB3D_NO_COLOR** 1 << 14
ob3d flag
- #define **JIT_OB3D_IS_SLAB** 1 << 15
ob3d flag
- #define **MAX_JIT_MOP_FLAGS_NONE** 0x00000000
mop flag
- #define **MAX_JIT_MOP_FLAGS_OWN_ALL** 0xFFFFFFFF
mop flag
- #define **MAX_JIT_MOP_FLAGS_OWN_JIT_MATRIX** 0x00000001
mop flag
- #define **MAX_JIT_MOP_FLAGS_OWN_BANG** 0x00000002
mop flag
- #define **MAX_JIT_MOP_FLAGS_OWN_OUTPUTMATRIX** 0x00000004
mop flag
- #define **MAX_JIT_MOP_FLAGS_OWN_NAME** 0x00000008
mop flag
- #define **MAX_JIT_MOP_FLAGS_OWN_TYPE** 0x00000010
mop flag

- #define **MAX_JIT_MOP_FLAGS_OWN_DIM** 0x00000020
mop flag
- #define **MAX_JIT_MOP_FLAGS_OWN_PLANECOUNT** 0x00000040
mop flag
- #define **MAX_JIT_MOP_FLAGS_OWN_CLEAR** 0x00000080
mop flag
- #define **MAX_JIT_MOP_FLAGS_OWN_NOTIFY** 0x00000100
mop flag
- #define **MAX_JIT_MOP_FLAGS_OWN_ADAPT** 0x00000200
mop flag
- #define **MAX_JIT_MOP_FLAGS_OWN_OUTPUTMODE** 0x00000400
mop flag
- #define **MAX_JIT_MOP_FLAGS_ONLY_MATRIX_PROBE** 0x10000000
mop flag
- #define **JIT_MOP_INPUT** 1
mop flag
- #define **JIT_MOP_OUTPUT** 2
mop flag

Typedefs

- typedef **t_object t_jit_object**
object header

Variables

- **JIT_EX_DATA t_symbol * _jit_sym_codec_raw**
cached t_symbol
- **JIT_EX_DATA t_symbol * _jit_sym_codec_cinepak**
cached t_symbol
- **JIT_EX_DATA t_symbol * _jit_sym_codec_graphics**
cached t_symbol
- **JIT_EX_DATA t_symbol * _jit_sym_codec_animation**
cached t_symbol

- JIT_EX_DATA `t_symbol * _jit_sym_codec_video`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_componentvideo`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_jpeg`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_mjpega`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_mjpegb`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_sgi`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_planarrgb`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_maqpaint`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_gif`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_photocd`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_qdgx`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_avrjpeg`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_opendmljpeg`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_bmp`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_winraw`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_vector`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_qd`
cached t_symbol

- JIT_EX_DATA `t_symbol * _jit_sym_codec_h261`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_h263`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_dvntsc`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_dvpal`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_dvprontsc`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_dvpropal`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_flc`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_targa`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_png`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_tiff`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_componentvideosigned`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_componentvideounsigned`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_cmyk`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_microsoft`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_sorenson`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_sorenson3`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_indeo4`

cached t_symbol

- JIT_EX_DATA *t_symbol * _jit_sym_codec_argb64*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_rgb48*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_alphagrey32*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_grey16*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_mpegyuv420*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_yuv420*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_sorensonyuv9*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_mpeg4*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_yuv422*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_v308*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_v408*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_v216*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_v210*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_v410*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_r408*
cached t_symbol
- JIT_EX_DATA *t_symbol * _jit_sym_codec_jpeg2000*
cached t_symbol

- JIT_EX_DATA `t_symbol * _jit_sym_codec_pixlet`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_h264`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_lossless`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_max`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_min`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_low`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_normal`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_high`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_none`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_raw`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_big16`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_little16`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_float32`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_float64`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_24bit`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_32bit`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_little32`
cached t_symbol

- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_mace3`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_mace6`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_cdx4`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_cdx2`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_ima`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_ulaw`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_alaw`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_adpcm`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_dviima`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_dvaudio`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_qdesign`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_qdesign2`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_qualcomm`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_mp3`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_vdva`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_codec_a_mpeg4`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_nothing`

cached t_symbol

- `JIT_EX_DATA t_symbol * _jit_sym_new`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_free`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_classname`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_getname`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_getmethod`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_get`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_set`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_register`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_char`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_long`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_float32`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_float64`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_symbol`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_pointer`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_object`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_atom`
cached t_symbol

- `JIT_EX_DATA t_symbol * _jit_sym_list`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_type`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_dim`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_planecount`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_val`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_plane`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_cell`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_jit_matrix`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_class_jit_matrix`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_togworld`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_fromgworld`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_frommatrix`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_class_jit_attribute`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_jit_attribute`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_jit_attr_offset`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_jit_attr_offset_array`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_rebuilding`
cached t_symbol

- JIT_EX_DATA `t_symbol * _jit_sym_modified`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_lock`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_setinfo`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_setinfo_ex`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_getinfo`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_data`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_getdata`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_outputmatrix`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_clear`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_clear_custom`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_err_calculate`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_max_jit_classex`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_setall`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_chuck`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_getsize`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_getindex`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_objptr2index`

cached t_symbol

- JIT_EX_DATA *t_symbol* * `_jit_sym_append`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_insertindex`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_deleteindex`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_chuckindex`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_makearray`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_reverse`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_rotate`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_shuffle`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_swap`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_findfirst`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_findall`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_methodall`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_methodindex`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_sort`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_matrix_calc`
cached t_symbol
- JIT_EX_DATA *t_symbol* * `_jit_sym_genframe`
cached t_symbol

- JIT_EX_DATA `t_symbol * _jit_sym_filter`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_jit_mop`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_newcopy`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_jit_linklist`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_inputcount`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_outputcount`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_getinput`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_getoutput`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_getinputlist`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_getoutputlist`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_ioname`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_matrixname`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_outputmode`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_matrix`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_getmatrix`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_typealink`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_dimlink`
cached t_symbol

- `JIT_EX_DATA t_symbol * _jit_sym_planelink`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_restrict_type`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_restrict_planeCount`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_restrict_dim`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_special`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_getspecial`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_adapt`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_decorator`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_frommatrix_trunc`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_ioproc`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_getioproc`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_name`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_types`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_minplaneCount`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_maxplaneCount`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_mindimCount`
cached t_symbol
- `JIT_EX_DATA t_symbol * _jit_sym_maxdimCount`

cached t_symbol

- JIT_EX_DATA `t_symbol * _jit_sym_mindim`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_maxdim`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gl_points`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gl_point_sprite`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gl_lines`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gl_line_strip`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gl_line_loop`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gl_triangles`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gl_tri_strip`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gl_tri_fan`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gl_quads`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gl_quad_strip`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gl_polygon`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gl_tri_grid`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gl_quad_grid`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_err_lockout_stack`
cached t_symbol

- JIT_EX_DATA `t_symbol * _jit_sym_class_jit_namespace`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_jit_namespace`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_findsize`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_attach`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_detach`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_add`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_replace`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_gettype`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_ob_sym`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_resolve_name`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_resolve_raw`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_notifyall`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_block`
cached t_symbol
- JIT_EX_DATA `t_symbol * _jit_sym_unblock`
cached t_symbol

33.22 Memory Management

In the past, Max has provided two separate APIs for memory management.

Defines

- `#define MM_UNIFIED`

This macro being defined means that getbytes and sysmem APIs for memory management are unified.

Functions

- `char * getbytes (short size)`

Allocate small amounts of non-relocatable memory.

- `void freebytes (void *b, short size)`

Free memory allocated with `getbytes()`.

- `char * getbytes16 (short size)`

Use `getbytes16()` to allocate small amounts of non-relocatable memory that is aligned on a 16-byte boundary for use with vector optimization.

- `void freebytes16 (char *mem, short size)`

Free memory allocated with `getbytes16()`.

- `char ** newhandle (long size)`

Allocate relocatable memory.

- `short growhandle (void *h, long size)`

Change the size of a handle.

- `void disposhandle (char **h)`

Free the memory used by a handle you no longer need.

- `t_ptr sysmem_newptr (long size)`

Allocate memory.

- `t_ptr sysmem_newptrclear (long size)`

Allocate memory and set it to zero.

- `t_ptr sysmem_resizeptr (void *ptr, long newsize)`

Resize an existing pointer.

- `t_ptr sysmem_resizeptrclear (void *ptr, long newsize)`

Resize an existing pointer and clear it.

- `long sysmem_ptrsize (void *ptr)`

Find the size of a pointer.

- void **sysmem_freeptr** (void *ptr)
Free memory allocated with [sysmem_newptr\(\)](#).
- void **sysmem_copyptr** (const void *src, void *dst, long bytes)
Copy memory the contents of one pointer to another pointer.
- **t_handle sysmem_newhandle** (long size)
Allocate a handle (a pointer to a pointer).
- **t_handle sysmem_newhandleclear** (unsigned long size)
Allocate a handle (a pointer to a pointer) whose memory is set to zero.
- long **sysmem_resizehandle** (**t_handle** handle, long newsize)
Resize an existing handle.
- long **sysmem_handlesize** (**t_handle** handle)
Find the size of a handle.
- void **sysmem_freehandle** (**t_handle** handle)
Free memory allocated with [sysmem_newhandle\(\)](#).
- long **sysmem_lockhandle** (**t_handle** handle, long lock)
Set the locked/unlocked state of a handle.
- long **sysmem_ptrandhand** (void *p, **t_handle** h, long size)
Add memory to an existing handle and copy memory to the resized portion from a pointer.
- long **sysmem_ptrbeforehand** (void *p, **t_handle** h, unsigned long size)
Add memory to an existing handle and copy memory to the resized portion from a pointer.
- long **sysmem_nullterminatehandle** (**t_handle** h)
Add a null terminator to a handle.

33.22.1 Detailed Description

In the past, Max has provided two separate APIs for memory management. One for allocating memory on the stack so that it was interrupt safe, including the [getbytes\(\)](#) and [freebytes\(\)](#) functions. The other, the "sysmem" API, were for allocating memory on the heap where larger amounts of memory were needed and the code could be guaranteed to operate at non-interrupt level.

Many things have changed in the environment of recent operating systems (MacOS X and Windows XP/Vista), the memory routines function differently, and the scheduler is no longer directly triggered by a hardware interrupt. In Max 5, the sysmem and getbytes API's have been unified, and thus may be used interchangeably.

The memory management unification can be switched on and off in the header files if needed, to compile code for older versions of Max for example, by changing the use of [MM_UNIFIED](#) in the Max headers.

33.22.2 Sysmem API

The Sysmem API provides a number of utilities for allocating and managing memory. It is relatively similar to some of the Macintosh Memory Manager API, and not too different from Standard C library memory functions. It is *not* safe to mix these routines with other memory routines (e.g. don't use malloc() to allocate a pointer, and [sysmem_freeptr\(\)](#) to free it).

33.22.3 Define Documentation

33.22.3.1 #define MM_UNIFIED

This macro being defined means that getbytes and sysmem APIs for memory management are unified.

This is correct for Max 5, but should be commented out when compiling for old max targets.

33.22.4 Function Documentation

33.22.4.1 void disposthandle (char ** *h*)

Free the memory used by a handle you no longer need.

Parameters

h The handle to dispose.

See also

[sysmem_freehandle\(\)](#)

33.22.4.2 void freebytes (void * *b*, short *size*)

Free memory allocated with [getbytes\(\)](#).

As of Max 5 it is unified with [sysmem_newptr\(\)](#), which is the preferred method for allocating memory.

Parameters

b A pointer to the block of memory previously allocated that you want to free.

size The size the block specified (as parameter b) in bytes.

33.22.4.3 void freebytes16 (char * *mem*, short *size*)

Free memory allocated with [getbytes16\(\)](#).

As of Max 5 it is unified with [sysmem_newptr\(\)](#), which is the preferred method for allocating memory.

Parameters

mem A pointer to the block of memory previously allocated that you want to free.

size The size the block specified (as parameter b) in bytes.

Remarks

Note that [freebytes16\(\)](#) will cause memory corruption if you pass it memory that was allocated with [getbytes\(\)](#). Use it only with memory allocated with [getbytes16\(\)](#).

33.22.4.4 char* getbytes (short size)

Allocate small amounts of non-relocatable memory.

As of Max 5 it is unified with [sysmem_newptr\(\)](#), which is the preferred method for allocating memory.

Parameters

size The size to allocate in bytes (up to 32767 bytes).

Returns

A pointer to the allocated memory.

33.22.4.5 char* getbytes16 (short size)

Use [getbytes16\(\)](#) to allocate small amounts of non-relocatable memory that is aligned on a 16-byte boundary for use with vector optimization.

Parameters

size The size to allocate in bytes (up to 32767 bytes).

Returns

A pointer to the allocated memory.

Remarks

[getbytes16\(\)](#) is identical to [getbytes](#) except that it returns memory that is aligned to a 16-byte boundary. This allows you to allocate storage for vector-optimized memory at interrupt level. Note that any memory allocated with [getbytes16\(\)](#) must be freed with [freebytes16\(\)](#), not [freebytes\(\)](#).

33.22.4.6 short growhandle (void * h, long size)

Change the size of a handle.

Parameters

h The handle to resize.

size The new size to allocate in bytes.

Returns

Ignored.

See also

[sysmem_resizehandle\(\)](#)

33.22.4.7 char newhandle (long size)**

Allocate relocatable memory.

Parameters

size The size to allocate in bytes.

Returns

The allocated handle.

See also

[sysmem_newhandle\(\)](#)

33.22.4.8 void sysmem_copyptr (const void * src, void * dst, long bytes)

Copy memory the contents of one pointer to another pointer.

This function is similar to BlockMove() or memcpy(). It copies the contents of the memory from the source to the destination pointer.

Parameters

src A pointer to the memory whose bytes will be copied.

dst A pointer to the memory where the data will be copied.

bytes The size in bytes of the data to be copied.

33.22.4.9 void sysmem_freehandle (t_handle handle)

Free memory allocated with [sysmem_newhandle\(\)](#).

Parameters

handle The handle whose memory will be freed.

33.22.4.10 void sysmem_freetr (void * ptr)

Free memory allocated with [sysmem_newptr\(\)](#).

This function is similar to DisposePtr or free. It frees the memory that had been allocated to the given pointer.

Parameters

ptr The pointer whose memory will be freed.

33.22.4.11 long sysmem_handlesize (t_handle handle)

Find the size of a handle.

This function is similar to GetHandleSize().

Parameters

handle The handle whose size will be queried.

Returns

The number of bytes allocated to the specified handle.

33.22.4.12 long sysmem_lockhandle (t_handle handle, long lock)

Set the locked/unlocked state of a handle.

This function is similar to HLock or HUnlock. It sets the lock state of a handle, using a zero or non-zero number.

Parameters

handle The handle that will be locked.

lock The new lock state of the handle.

Returns

The previous lock state.

33.22.4.13 t_handle sysmem_newhandle (long size)

Allocate a handle (a pointer to a pointer).

This function is similar to NewHandle(). It allocates a handle of a given number of bytes and returns a [t_handle](#).

Parameters

size The size of the handle in bytes that will be allocated.

Returns

A new [t_handle](#).

33.22.4.14 t_handle sysmem_newhandleclear (unsigned long size)

Allocate a handle (a pointer to a pointer) whose memory is set to zero.

Parameters

size The size of the handle in bytes that will be allocated.

Returns

A new `t_handle`.

See also

[sysmem_newhandle\(\)](#)

33.22.4.15 t_ptr sysmem_newptr (long size)

Allocate memory.

This function is similar to NewPtr() or malloc(). It allocates a pointer of a given number of bytes and returns a pointer to the memory allocated.

Parameters

`size` The amount of memory to allocate.

Returns

A pointer to the allocated memory, or NULL if the allocation fails.

33.22.4.16 t_ptr sysmem_newptrclear (long size)

Allocate memory and set it to zero.

This function is similar to NewPtrClear() or calloc(). It allocates a pointer of a given number of bytes, zeroing all memory, and returns a pointer to the memory allocated.

Parameters

`size` The amount of memory to allocate.

Returns

A pointer to the allocated memory, or NULL if the allocation fails.

33.22.4.17 long sysmem_nullterminatehandle (t_handle h)

Add a null terminator to a handle.

Parameters

`h` A handle to null terminate.

Returns

An error code.

33.22.4.18 long sysmem_ptrandhand (void **p*, t_handle *h*, long *size*)

Add memory to an existing handle and copy memory to the resized portion from a pointer.

This function is similar to `PtrAndHand()`. It resizes an existing handle by adding a given number of bytes to it and copies data from a pointer into those bytes.

Parameters

- p* The existing pointer whose data will be copied into the resized handle.
- h* The handle which will be enlarged by the size of the pointer.
- size* The size in bytes that will be added to the handle.

Returns

The number of bytes allocated to the specified handle.

33.22.4.19 long sysmem_ptrbeforehand (void **p*, t_handle *h*, unsigned long *size*)

Add memory to an existing handle and copy memory to the resized portion from a pointer.

Unlike `sysmem_ptrandhand()`, however, this copies the ptr before the previously existing handle data.

Parameters

- p* The existing pointer whose data will be copied into the resized handle.
- h* The handle which will be enlarged by the size of the pointer.
- size* The size in bytes that will be added to the handle.

Returns

An error code.

33.22.4.20 long sysmem_ptrsize (void **ptr*)

Find the size of a pointer.

This function is similar to `_msize()`.

Parameters

- ptr* The pointer whose size will be queried

Returns

The number of bytes allocated to the pointer specified.

33.22.4.21 long sysmem_resizehandle (t_handle *handle*, long *newsiz*)

Resize an existing handle.

This function is similar to `SetHandleSize()`. It resizes an existing handle to the size specified.

Parameters

handle The handle that will be resized.

newsize The new size of the handle in bytes.

Returns

The number of bytes allocated to the specified handle.

33.22.4.22 t_ptr sysmem_resizeptr (void *ptr, long newsize)

Resize an existing pointer.

This function is similar to realloc(). It resizes an existing pointer and returns a new pointer to the resized memory.

Parameters

ptr The pointer to the memory that will be resized.

newsize The new size of the pointer in bytes.

Returns

A pointer to the resized memory, or NULL if the allocation fails.

33.22.4.23 t_ptr sysmem_resizeptrclear (void *ptr, long newsize)

Resize an existing pointer and clear it.

Parameters

ptr The pointer to the memory that will be resized.

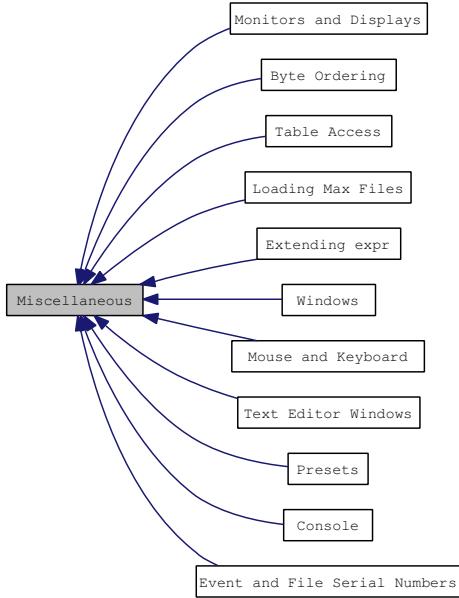
newsize The new size of the pointer in bytes.

Returns

A pointer to the resized memory, or NULL if the allocation fails.

33.23 Miscellaneous

Collaboration diagram for Miscellaneous:



Modules

- [Console](#)
- [Byte Ordering](#)

Utilities for swapping the order of bytes to match the Endianness of the required platform.

- [Extending expr](#)

If you want to use C-like variable expressions that are entered by a user of your object, you can use the "guts" of Max's expr object in your object.

- [Table Access](#)

You can use these functions to access named table objects.

- [Text Editor Windows](#)

Max has a simple built-in text editor object that can display and edit text in conjunction with your object.

- [Presets](#)

Max contains a preset object that has the ability to send preset messages to some or all of the objects (clients) in a Patcher window.

- [Event and File Serial Numbers](#)

If you call `outlet_int()`, `outlet_float()`, `outlet_list()`, or `outlet_anything()` inside a Qelem or during some idle or interrupt time, you should increment Max's Event Serial Number beforehand.

- [Loading Max Files](#)

Several high-level functions permit you to load patcher files.

- [Monitors and Displays](#)

Functions for finding our information about the environment.

- [Windows](#)

- [Mouse and Keyboard](#)

Defines

- `#define InRange(v, lo, hi) ((v)<=(hi)&&(v)>=(lo))`

If a value is within the specified range, then return true.

- `#define MAX(a, b) ((a)>(b)?(a):(b))`

Return the higher of two values.

- `#define MIN(a, b) ((a)<(b)?(a):(b))`

Return the lower of two values.

- `#define CLIP(a, lo, hi) ((a)>(lo)?((a)<(hi)?(a):(hi)):((lo)))`

Limit values to within a specified range.

- `#define calcoffset(x, y) ((long)(&(((x *)0L)->y)))`

Find byte offset of a named member of a struct, relative to the beginning of that struct.

- `#define BEGIN_USING_C_LINKAGE`

Ensure that any definitions following this macro use a C-linkage, not a C++ linkage.

- `#define END_USING_C_LINKAGE`

Close a definition section that was opened using [BEGIN_USING_C_LINKAGE](#).

Enumerations

- `enum e_max_errorcodes {`

`MAX_ERR_NONE = 0,`

`MAX_ERR_GENERIC = -1,`

`MAX_ERR_INVALID_PTR = -2,`

`MAX_ERR_DUPLICATE = -3,`

`MAX_ERR_OUT_OF_MEM = -4 }`

Standard values returned by function calls with a return type of [t_max_err](#).

- `enum e_max_wind_advise_result {`

`aaYes = 1,`

`aaNo,`

`aaCancel }`

Returned values from [wind_advise\(\)](#).

Functions

- `void * globalsymbol_reference (t_object *x, C74_CONST char *name, C74_CONST char *classname)`

Get a reference to an object that is bound to a [t_symbol](#).
- `void globalsymbol_dereference (t_object *x, C74_CONST char *name, C74_CONST char *classname)`

Stop referencing an object that is bound to a [t_symbol](#), previously referenced using [globalsymbol_reference\(\)](#).
- `t_max_err globalsymbol_bind (t_object *x, C74_CONST char *name, long flags)`

Bind an object to a [t_symbol](#).
- `void globalsymbol_unbind (t_object *x, C74_CONST char *name, long flags)`

Remove an object from being bound to a [t_symbol](#).
- `long method_true (void *x)`

A method that always returns true.
- `long method_false (void *x)`

A method that always returns false.
- `t_symbol * symbol_unique ()`

*Generates a unique [t_symbol](#) *.*
- `void error_sym (void *x, t_symbol *s)`

Posts an error message to the Max window.
- `void post_sym (void *x, t_symbol *s)`

Posts a message to the Max window.
- `t_max_err symbolarray_sort (long ac, t_symbol **av)`

*Performs an ASCII sort on an array of [t_symbol](#) *s.*
- `void object_obex_quickref (void *x, long *numitems, t_symbol **items)`

Developers do not need to directly use the [object_obex_quickref\(\)](#) function.
- `void error_subscribe (t_object *x)`

Receive messages from the error handler.
- `void error_unsubscribe (t_object *x)`

Remove an object as an error message recipient.
- `void quittask_install (method m, void *a)`

Register a function that will be called when Max exits.
- `void quittask_remove (method m)`

Unregister a function previously registered with [quittask_install\(\)](#).

- short [maxversion](#) (void)
Determine version information about the current Max environment.
- char * [strncpy_zero](#) (char *dst, const char *src, long size)
Copy the contents of one string to another, in a manner safer than the standard strcpy() or strncpy().
- char * [strncat_zero](#) (char *dst, const char *src, long size)
Concatenate the contents of one string onto the end of another, in a manner safer than the standard strcat() or strncat().
- int [snprintf_zero](#) (char *buffer, size_t count, const char *format,...)
Copy the contents of a string together with value substitutions, in a manner safer than the standard sprintf() or snprintf().
- short [wind_advise](#) ([t_object](#) *w, char *s,...)
Throw a dialog which may have text and up to three buttons.
- void [wind_setcursor](#) (short which)
Change the cursor.

33.23.1 Define Documentation

33.23.1.1 #define BEGIN_USING_C_LINKAGE

Ensure that any definitions following this macro use a C-linkage, not a C++ linkage.

The Max API uses C-linkage. This is important for objects written in C++ or that use a C++ compiler. This macro must be balanced with the [END_USING_C_LINKAGE](#) macro.

33.23.1.2 #define calcoffset(x, y) ((long)(&(((x *)0L)->y)))

Find byte offset of a named member of a struct, relative to the beginning of that struct.

Parameters

- x* The name of the struct
- y* The name of the member

Returns

A long integer representing the number of bytes into the struct where the member begins.

33.23.1.3 #define CLIP(a, lo, hi) ((a)>(lo)?((a)<(hi)?(a):(hi)): (lo))

Limit values to within a specified range.

Parameters

- a* The value to constrain.
- lo* The low bound for the range.

hi The high bound for the range.

Returns

Returns the value a constrained to the range specified by lo and hi.

33.23.1.4 #define InRange(v, lo, hi) ((v)<=(hi)&&(v)>=(lo))

If a value is within the specified range, then return true.

Otherwise return false.

Parameters

v The value to test.

lo The low bound for the range.

hi The high bound for the range.

Returns

Returns true if within range, otherwise false.

33.23.1.5 #define MAX(a, b) ((a)>(b)?(a):(b))

Return the higher of two values.

Parameters

a The first value to compare.

b The second value to compare.

Returns

Returns the higher of a or b.

33.23.1.6 #define MIN(a, b) ((a)<(b)?(a):(b))

Return the lower of two values.

Parameters

a The first value to compare.

b The second value to compare.

Returns

Returns the lower of a or b.

33.23.2 Enumeration Type Documentation

33.23.2.1 enum e_max_errorcodes

Standard values returned by function calls with a return type of [t_max_err](#).

Enumerator:

- MAX_ERR_NONE*** No error.
- MAX_ERR_GENERIC*** Generic error.
- MAX_ERR_INVALID_PTR*** Invalid Pointer.
- MAX_ERR_DUPLICATE*** Duplicate.
- MAX_ERR_OUT_OF_MEM*** Out of memory.

33.23.2.2 enum e_max_wind_advise_result

Returned values from [wind_advise\(\)](#).

Enumerator:

- aaYes*** Yes button was chosen.
- aaNo*** No button was chosen.
- aaCancel*** Cancel button was chosen.

33.23.3 Function Documentation

33.23.3.1 void error_subscribe (t_object *x)

Receive messages from the error handler.

Parameters

x The object to be subscribed to the error handler.

Remarks

[error_subscribe\(\)](#) enables your object to receive a message (error), followed by the list of atoms in the error message posted to the Max window.

Prior to calling [error_subscribe\(\)](#), you should bind the error message to an internal error handling routine:

```
addmess( (method)myobject_error, "error", A_GIMME, 0);
```

Your error handling routine should be declared as follows:

```
void myobject_error(t_myobject *x, t_symbol *s, short argc, t_atom *argv);
```

33.23.3.2 void error_sym (void *x, t_symbol *s)

Posts an error message to the Max window.

This function is interrupt safe.

Parameters

- x* The object's pointer
- s* Symbol to be posted as an error in the Max window

33.23.3.3 void error_unsubscribe (t_object *x)

Remove an object as an error message recipient.

Parameters

- x* The object to unsubscribe.

33.23.3.4 t_max_err globalsymbol_bind (t_object *x, C74_CONST char *name, long flags)

Bind an object to a [t_symbol](#).

Parameters

- x* The object to bind to the [t_symbol](#).
- name* The name of the [t_symbol](#) to which the object will be bound.
- flags* Pass 0.

Returns

A Max error code.

33.23.3.5 void globalsymbol_dereference (t_object *x, C74_CONST char *name, C74_CONST char *classname)

Stop referencing an object that is bound to a [t_symbol](#), previously referenced using [globalsymbol_reference\(\)](#).

Parameters

- x* The object that is getting the reference to the symbol.
- name* The name of the symbol to reference.
- classname* The name of the class of which the object we are referencing should be an instance.

See also

[globalsymbol_reference\(\)](#)

33.23.3.6 void* globalsymbol_reference (t_object *x, C74_CONST char *name, C74_CONST char *classname)

Get a reference to an object that is bound to a [t_symbol](#).

Parameters

- x* The object that is getting the reference to the symbol.
- name* The name of the symbol to reference.
- classname* The name of the class of which the object we are referencing should be an instance.

Returns

The s_thing of the [t_symbol](#).

Remarks

An example of real-world use is to get the buffer~ object associated with a symbol.

```
// the struct of our object
typedef struct _myobject {
    t_object obj;
    t_symbol *buffer_name;
    t_buffer *buffer_object;
} t_myobject;

void myobject_setbuffer(t_myobject *x, t_symbol *s, long argc, t_atom *argv)
{
    if(s != x->buffer_name) {
        // Reference the buffer associated with the incoming name
        x->buffer_object = (t_buffer *)globalsymbol_reference((t_object *)x, s->s_name, "buffer~");

        // If we were previously referencing another buffer, we should not longer
        // reference it.
        globalsymbol_dereference((t_object *)x, x->buffer_name->s_name, "buffer~");
    }

    x->buffer_name = s;
}
```

33.23.3.7 void globalsymbol_unbind (t_object *x, C74_CONST char *name, long flags)

Remove an object from being bound to a [t_symbol](#).

Parameters

- x* The object from which to unbind the [t_symbol](#).
- name* The name of the [t_symbol](#) from which the object will be unbound.
- flags* Pass 0.

33.23.3.8 short maxversion (void)

Determine version information about the current Max environment.

This function returns the version number of Max. In Max versions 2.1.4 and later, this number is the version number of the Max kernel application in binary-coded decimal. Thus, 2.1.4 would return 214 hex or 532 decimal. Version 3.0 returns 300 hex.

Use this to check for the existence of particular function macros that are only present in more recent Max versions. Versions before 2.1.4 returned 1, except for versions 2.1.1 - 2.1.3 which returned 2.

Bit 14 (counting from left) will be set if Max is running as a standalone application, so you should mask the lower 12 bits to get the version number.

Returns

The Max environment's version number.

33.23.3.9 void object_obex_quickref (void **x*, long **numitems*, t_symbol *items*)**

Developers do not need to directly use the [object_obex_quickref\(\)](#) function.

It was used in Max 4 to add support for attributes to the quickref, but this is automatic in Max 5.

33.23.3.10 void post_sym (void **x*, t_symbol **s*)

Posts a message to the Max window.

This function is interrupt safe.

Parameters

x The object's pointer

s Symbol to be posted in the Max window

33.23.3.11 void quittask_install (method *m*, void **a*)

Register a function that will be called when Max exits.

Parameters

m A function that will be called on Max exit.

a Argument to be used with method *m*.

Remarks

[quittask_install\(\)](#) provides a mechanism for your external to register a routine to be called prior to Max shutdown. This is useful for objects that need to provide disk-based persistance outside the standard Max storage mechanisms, or need to shut down hardware or their connection to system software and cannot do so in the termination routine of a code fragment.

33.23.3.12 void quittask_remove (method *m*)

Unregister a function previously registered with [quittask_install\(\)](#).

Parameters

m Function to be removed as a shutdown method.

33.23.3.13 int snprintf_zero (char * *buffer*, size_t *count*, const char * *format*, ...)

Copy the contents of a string together with value substitutions, in a manner safer than the standard sprintf() or snprintf().

This is the preferred function to use for this operation in Max.

Parameters

- buffer* The destination string (already allocated) for the copy.
- count* The number of chars allocated to the buffer string.
- format* The source string that will be copied, which may include sprintf() formatting codes for substitutions.
- ... An array of arguments to be substituted into the format string.

33.23.3.14 char* strncat_zero (char * *dst*, const char * *src*, long *size*)

Concatenate the contents of one string onto the end of another, in a manner safer than the standard strcat() or strncat().

This is the preferred function to use for this operation in Max.

Parameters

- dst* The destination string onto whose end the src string will be appended.
- src* The source string that will be copied.
- size* The number of chars allocated to the dst string.

33.23.3.15 char* strncpy_zero (char * *dst*, const char * *src*, long *size*)

Copy the contents of one string to another, in a manner safer than the standard strcpy() or strncpy().

This is the preferred function to use for this operation in Max.

Parameters

- dst* The destination string (already allocated) for the copy.
- src* The source string that will be copied.
- size* The number of chars allocated to the dst string.

33.23.3.16 t_symbol* symbol_unique ()

Generates a unique [t_symbol](#) *.

The symbol will be formatted somewhat like "u123456789".

Returns

This function returns a unique [t_symbol](#) *.

33.23.3.17 `t_max_err symbolarray_sort (long ac, t_symbol **av)`

Performs an ASCII sort on an array of `t_symbol` *s.

Parameters

- ac*** The count of `t_symbol` *s in `av`
- av*** An array of `t_symbol` *s to be sorted

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.23.3.18 `short wind_advise (t_object *w, char *s, ...)`

Throw a dialog which may have text and up to three buttons.

For example, this can be used to ask "Save changes before..."

Parameters

- w*** The window with which this dialog is associated.
- s*** A string with any sprintf()-like formatting to be displayed.
- ...** Any variables that should be substituted in the string defined by `s`.

Returns

One of the values defined in `e_max_wind_advise_result`, depending on what the user selected.

33.23.3.19 `void wind_setcursor (short which)`

Change the cursor.

Parameters

- which*** One of the following predefined cursors:

```
#define C_ARROW      1
#define C_WATCH     2
#define C_IBeam     3
#define C_HAND      4
#define C_CROSS     5
#define C_PENCIL    6
#define C_GROW      8
```

Remarks

`wind_setcursor()` keeps track of what the cursor was previously set to, so if something else has changed the cursor, you may not see a new cursor if you set it to the previous argument to `wind_setcursor()`.

The solution is to call `wind_setcursor(0)` before calling it with the desired cursor constant. Use `wind_setcursor(-1)` to tell Max you'll set the cursor to directly.

33.24 Console

Collaboration diagram for Console:



Functions

- void [post](#) (C74_CONST char *fmt,...)
Print text to the Max window.
- void [cpost](#) (C74_CONST char *fmt,...)
Print text to the system console.
- void [error](#) (C74_CONST char *fmt,...)
Print an error to the Max window.
- void [ouchstring](#) (C74_CONST char *s,...)
Put up an error or advisory alert box on the screen.
- void [postatom](#) ([t_atom](#) *ap)
Print multiple items in the same line of text in the Max window.
- void [object_post](#) ([t_object](#) *x, C74_CONST char *s,...)
Print text to the Max window, linked to an instance of your object.
- void [object_error](#) ([t_object](#) *x, C74_CONST char *s,...)
Print text to the Max window, linked to an instance of your object, and flagged as an error (highlighted with a red background).
- void [object_warn](#) ([t_object](#) *x, C74_CONST char *s,...)
Print text to the Max window, linked to an instance of your object, and flagged as a warning (highlighted with a yellow background).
- void [object_error_obtrusive](#) ([t_object](#) *x, char *s,...)
Print text to the Max window, linked to an instance of your object, and flagged as an error (highlighted with a red background), and grab the user's attention by displaying a banner in the patcher window.

33.24.1 Function Documentation

33.24.1.1 void [cpost](#) (C74_CONST char **fmt*, ...)

Print text to the system console.

On the Mac this post will be visible by launching Console.app in the /Applications/Utilities folder. On Windows this post will be visible by launching the dbgView.exe program, which is a free download as a part of Microsoft's SysInternals.

Parameters

fmt A C-string containing text and printf-like codes specifying the sizes and formatting of the additional arguments.

... Arguments of any type that correspond to the format codes in *fmtString*.

Remarks

Particularly on MacOS 10.5, posting to Console.app can be a computationally expensive operation. Use with care.

See also

[post\(\)](#)
[object_post\(\)](#)

33.24.1.2 void error (C74_CONST char **fmt*, ...)

Print an error to the Max window.

Max 5 introduced [object_error\(\)](#), which provides several enhancements to [error\(\)](#) where a valid [t_object](#) pointer is available.

[error\(\)](#) is very similar to [post\(\)](#), thought it offers two additional features:

- the post to the Max window is highlighted (with a red background).
- the post can be trapped with the error object in a patcher.

Parameters

fmt A C-string containing text and printf-like codes specifying the sizes and formatting of the additional arguments.

... Arguments of any type that correspond to the format codes in *fmtString*.

See also

[object_post\(\)](#)
[post\(\)](#)
[cpost\(\)](#)

33.24.1.3 void object_error ([t_object](#) **x*, C74_CONST char **s*, ...)

Print text to the Max window, linked to an instance of your object, and flagged as an error (highlighted with a red background).

Max window rows which are generated using [object_post\(\)](#) or [object_error\(\)](#) can be double-clicked by the user to have Max assist with locating the object in a patcher. Rows created with [object_post\(\)](#) and [object_error\(\)](#) will also automatically provide the name of the object's class in the correct column in the Max window.

Parameters

x A pointer to your object.

- s* A C-string containing text and printf-like codes specifying the sizes and formatting of the additional arguments.
- ... Arguments of any type that correspond to the format codes in fmtString.

See also

[object_post\(\)](#)
[object_warn\(\)](#)

33.24.1.4 void object_error_obtrusive (*t_object* **x*, *char* **s*, ...)

Print text to the Max window, linked to an instance of your object, and flagged as an error (highlighted with a red background), and grab the user's attention by displaying a banner in the patcher window.

This function should be used exceedingly sparingly, with preference given to [object_error\(\)](#) when a problem occurs.

Parameters

- x* A pointer to your object.
- s* A C-string containing text and printf-like codes specifying the sizes and formatting of the additional arguments.
- ... Arguments of any type that correspond to the format codes in fmtString.

See also

[object_post\(\)](#)
[object_error\(\)](#)

33.24.1.5 void object_post (*t_object* **x*, C74_CONST *char* **s*, ...)

Print text to the Max window, linked to an instance of your object.

Max window rows which are generated using [object_post\(\)](#) or [object_error\(\)](#) can be double-clicked by the user to have Max assist with locating the object in a patcher. Rows created with [object_post\(\)](#) and [object_error\(\)](#) will also automatically provide the name of the object's class in the correct column in the Max window.

Parameters

- x* A pointer to your object.
- s* A C-string containing text and printf-like codes specifying the sizes and formatting of the additional arguments.
- ... Arguments of any type that correspond to the format codes in fmtString.

Remarks

Example:

```
void myMethod(myObject *x, long someArgument)
{
    object_post((t_object*)x, "This is my argument: %ld", someArgument);
}
```

See also

[object_error\(\)](#)

33.24.1.6 void object_warn (*t_object* **x*, C74_CONST char **s*, ...)

Print text to the Max window, linked to an instance of your object, and flagged as a warning (highlighted with a yellow background).

Max window rows which are generated using [object_post\(\)](#), [object_error\(\)](#), or [object_warn](#) can be double-clicked by the user to have Max assist with locating the object in a patcher. Rows created with [object_post\(\)](#), [object_error\(\)](#), or [object_warn\(\)](#) will also automatically provide the name of the object's class in the correct column in the Max window.

Parameters

- x* A pointer to your object.
- s* A C-string containing text and printf-like codes specifying the sizes and formatting of the additional arguments.
- ... Arguments of any type that correspond to the format codes in fmtString.

See also

- [object_post\(\)](#)
- [object_error\(\)](#)

33.24.1.7 void ouchstring (C74_CONST char **s*, ...)

Put up an error or advisory alert box on the screen.

Don't use this function. Instead use [error\(\)](#), [object_error\(\)](#), or [object_error_obtrusive\(\)](#).

This function performs an sprintf() on fmtstring and items, then puts up an alert box. [ouchstring\(\)](#) will queue the message to a lower priority level if it's called in an interrupt and there is no alert box request already pending.

Parameters

- s* A C-string containing text and printf-like codes specifying the sizes and formatting of the additional arguments.
- ... Arguments of any type that correspond to the format codes in fmtString.

See also

- [error\(\)](#)
- [object_error\(\)](#)
- [object_error_obtrusive\(\)](#)

33.24.1.8 void post (C74_CONST char **fmt*, ...)

Print text to the Max window.

Max 5 introduced [object_post\(\)](#), which provides several enhancements to [post\(\)](#) where a valid *t_object* pointer is available.

[post\(\)](#) is a printf() for the Max window. It even works from non-main threads, queuing up multiple lines of text to be printed when the main thread processing resumes. [post\(\)](#) can be quite useful in debugging your external object.

Parameters

- fmt* A C-string containing text and printf-like codes specifying the sizes and formatting of the additional arguments.
- ... Arguments of any type that correspond to the format codes in `fmtString`.

Remarks

Note that `post` only passes 16 bytes of arguments to `sprintf`, so if you want additional formatted items on a single line, use [postatom\(\)](#).

Example:

```
short whatIsIt;  
  
whatIsIt = 999;  
post ("the variable is %ld", (long)whatIsIt);
```

Remarks

The Max Window output when this code is executed.

```
the variable is 999
```

See also

[object_post\(\)](#)
[error\(\)](#)
[cpost\(\)](#)

33.24.1.9 void postatom (*t_atom * ap*)

Print multiple items in the same line of text in the Max window.

This function prints a single [`t_atom`](#) on a line in the Max window without a carriage return afterwards, as [post\(\)](#) does. Each [`t_atom`](#) printed is followed by a space character.

Parameters

- ap* The address of a [`t_atom`](#) to print.

See also

[object_post\(\)](#)
[post\(\)](#)
[cpost\(\)](#)

33.25 Byte Ordering

Utilities for swapping the order of bytes to match the Endianness of the required platform.

Collaboration diagram for Byte Ordering:



Defines

- `#define C74_LITTLE_ENDIAN 1`
A macro that indicates whether or not the current architecture uses Little-endian byte ordering (such as is used on an i386 processor).
- `#define C74_BIG_ENDIAN 0`
A macro that indicates whether or not the current architecture uses Big-endian byte ordering (such as is used on a PPC processor).
- `#define BYTEORDER_SWAPW16(x) (((short)((((unsigned short)(x))>>8)&0x00ff)+(((unsigned short)(x))<<8)&0xff00)))`
Switch the byte ordering of a short integer.
- `#define BYTEORDER_SWAPW32(x)`
Switch the byte ordering of an integer.
- `#define BYTEORDER_SWAPF32 byteorder_swapf32`
Switch the byte ordering of a float.
- `#define BYTEORDER_SWAPF64 byteorder_swapf64`
Switch the byte ordering of a double.
- `#define BYTEORDER_LSBW16(x) (x)`
Switch the byte ordering of a short integer from the native swapping to Little-endian (Least Significant Byte).
- `#define BYTEORDER_LSBW32(x) (x)`
Switch the byte ordering of an integer from the native swapping to Little-endian (Least Significant Byte).
- `#define BYTEORDER_LSBF32(x) (x)`
Switch the byte ordering of a float from the native swapping to Little-endian (Least Significant Byte).
- `#define BYTEORDER_LSBF64(x) (x)`
Switch the byte ordering of a double from the native swapping to Little-endian (Least Significant Byte).
- `#define BYTEORDER_MSBW16(x) BYTEORDER_SWAPW16(x)`
Switch the byte ordering of a short integer from the native swapping to Big-endian (Most Significant Byte).
- `#define BYTEORDER_MSBW32(x) BYTEORDER_SWAPW32(x)`
Switch the byte ordering of an integer from the native swapping to Big-endian (Most Significant Byte).

- #define **BYTEORDER_MSBF32(x)** BYTEORDER_SWAPF32(x)
Switch the byte ordering of a float from the native swapping to Big-endian (Most Significant Byte).
- #define **BYTEORDER_MSBF64(x)** BYTEORDER_SWAPF64(x)
Switch the byte ordering of a double from the native swapping to Big-endian (Most Significant Byte).

33.25.1 Detailed Description

Utilities for swapping the order of bytes to match the Endianness of the required platform. An introduction to the issue of endianness can be found at <http://en.wikipedia.org/wiki/Endianness>.

Of particular relevance is that a Macintosh with a PPC processor uses a Big-endian byte ordering, whereas an Intel processor in a Mac or Windows machine will use a Little-endian byte ordering.

These utilities are defined to assist with cases where byte ordering needs to be manipulated for floats or ints. Note that floats are subject to the same byte ordering rules as integers. While the IEEE defines the bits, the machine still defines how the bits are arranged with regard to bytes.

33.25.2 Define Documentation

33.25.2.1 #define **BYTEORDER_LSBF32(x) (x)**

Switch the byte ordering of a float from the native swapping to Little-endian (Least Significant Byte).

If the current environment is already Little-endian, then the returned value is not byteswapped.

Parameters

x A float.

Returns

A float with the byte-ordering swapped if necessary.

33.25.2.2 #define **BYTEORDER_LSBF64(x) (x)**

Switch the byte ordering of a double from the native swapping to Little-endian (Least Significant Byte).

If the current environment is already Little-endian, then the returned value is not byteswapped.

Parameters

x A double.

Returns

A double with the byte-ordering swapped if necessary.

33.25.2.3 #define **BYTEORDER_LSBW16(x) (x)**

Switch the byte ordering of a short integer from the native swapping to Little-endian (Least Significant Byte).

If the current environment is already Little-endian, then the returned value is not byteswapped.

Parameters

x A short integer.

Returns

A short integer with the byte-ordering swapped if necessary.

33.25.2.4 #define BYTEORDER_LSBW32(x) (x)

Switch the byte ordering of an integer from the native swapping to Little-endian (Least Significant Byte). If the current environment is already Little-endian, then the returned value is not byteswapped.

Parameters

x An integer.

Returns

An integer with the byte-ordering swapped if necessary.

33.25.2.5 #define BYTEORDER_MSBF32(x) BYTEORDER_SWAPF32(x)

Switch the byte ordering of a float from the native swapping to Big-endian (Most Significant Byte). If the current environment is already Big-endian, then the returned value is not byteswapped.

Parameters

x A float.

Returns

A float with the byte-ordering swapped if necessary.

33.25.2.6 #define BYTEORDER_MSBF64(x) BYTEORDER_SWAPF64(x)

Switch the byte ordering of a double from the native swapping to Big-endian (Most Significant Byte). If the current environment is already Big-endian, then the returned value is not byteswapped.

Parameters

x A double.

Returns

A double with the byte-ordering swapped if necessary.

33.25.2.7 #define BYTEORDER_MSBW16(x) BYTEORDER_SWAPW16(x)

Switch the byte ordering of a short integer from the native swapping to Big-endian (Most Significant Byte). If the current environment is already Big-endian, then the returned value is not byteswapped.

Parameters

x A short integer.

Returns

A short integer with the byte-ordering swapped if necessary.

33.25.2.8 #define BYTEORDER_MSBW32(x) BYTEORDER_SWAPW32(x)

Switch the byte ordering of an integer from the native swapping to Big-endian (Most Significant Byte). If the current environment is already Big-endian, then the returned value is not byteswapped.

Parameters

x An integer.

Returns

An integer with the byte-ordering swapped if necessary.

33.25.2.9 #define BYTEORDER_SWAPF32 byteorder_swapf32

Switch the byte ordering of a float.

Parameters

x A float.

Returns

A float with the byte-ordering swapped.

33.25.2.10 #define BYTEORDER_SWAPF64 byteorder_swapf64

Switch the byte ordering of a double.

Parameters

x A double.

Returns

A double.

33.25.2.11 #define BYTEORDER_SWAPW16(x) (((short)((((unsigned short)(x)>>8)&0x00ff)+(((unsigned short)(x))<<8)&0xff00)))

Switch the byte ordering of a short integer.

Parameters

x A short integer.

Returns

A short integer with the byte-ordering swapped.

33.25.2.12 #define BYTEORDER_SWAPW32(x)

Value:

```
((long) (((((unsigned long) (x))>>24L)&0x000000ff)+((((unsigned long) (x))>>8L)&0x0000ff00)+\n         (((unsigned long) (x))<<24L)&0xff000000)+(((unsigned long) (x))<<8L)&0x00ff0000))
```

Switch the byte ordering of an integer.

Parameters

x An integer.

Returns

An integer with the byte-ordering swapped.

33.25.2.13 #define C74_BIG_ENDIAN 0

A macro that indicates whether or not the current architecture uses Big-endian byte ordering (such as is used on a PPC processor).

Note that this macro is always defined; it will be either a 0 or a 1.

33.25.2.14 #define C74_LITTLE_ENDIAN 1

A macro that indicates whether or not the current architecture uses Little-endian byte ordering (such as is used on an i386 processor).

Note that this macro is always defined; it will be either a 0 or a 1.

33.26 Extending expr

If you want to use C-like variable expressions that are entered by a user of your object, you can use the "guts" of Max's expr object in your object.

Collaboration diagram for Extending expr:



Data Structures

- struct [Ex_ex](#)
ex_ex.
- struct [t_expr](#)
Struct for an instance of expr.

Defines

- #define [ex_int](#) ex_cont.v_int
shortcut for accessing members of an Ex_ex struct's ex_cont union.
- #define [ex_flt](#) ex_cont.v_flt
shortcut for accessing members of an Ex_ex struct's ex_cont union.
- #define [ex_op](#) ex_cont.op
shortcut for accessing members of an Ex_ex struct's ex_cont union.
- #define [ex_ptr](#) ex_cont.ptr
shortcut for accessing members of an Ex_ex struct's ex_cont union.

Enumerations

- enum [e_max_expr_types](#) {

 ET_INT = 0x1,

 ET_FLT = 0x2,

 ET_OP = 0x3,

 ET_STR = 0x4,

 ET_TBL = 0x5,

 ET_FUNC = 0x6,

 ET_SYM = 0x7,

 ET_VSYM = 0x8,

 ET_LP = 0x9,

 ET_LB = 0x10,
 }

```

ET_HI = 0x11,
ET_FL = 0x12,
ET_SI = 0x13 }

Defines for ex_type.

```

Functions

- void * **expr_new** (short argc, **t_atom** *argv, **t_atom** *types)
Create a new expr object.
- short **expr_eval** (**t_expr** *x, short argc, **t_atom** *argv, **t_atom** *result)
Evaluate an expression in an expr object.

33.26.1 Detailed Description

If you want to use C-like variable expressions that are entered by a user of your object, you can use the "guts" of Max's expr object in your object. For example, the if object uses expr routines for evaluating a conditional expression, so it can decide whether to send the message after the words then or else. The following functions provide an interface to expr.

33.26.2 Enumeration Type Documentation

33.26.2.1 enum e_max_expr_types

Defines for ex_type.

We treat parenthesis and brackets special to keep a pointer to their match in the content.

Enumerator:

```

ET_INT an int
ETFLT a float
ETOP operator
ETSTR string
ETTBL a table, the content is a pointer
ETFUNC a function
ETSYM symbol ("string")
ETVSYM variable symbol ("$s?")
ETLP left parenthesis
ETLB left bracket
ETII and integer inlet
ETFI float inlet
ETSI string inlet

```

33.26.3 Function Documentation

33.26.3.1 short expr_eval (*t_expr *x*, *short argc*, *t_atom *argv*, *t_atom *result*)

Evaluate an expression in an expr object.

Parameters

- x* The expr object to evaluate.
- argc* Count of arguments in argv.
- argv* Array of nine Atoms that will be substituted for variable arguments (such as \$i1) in the expression. Unused arguments should be of type [A NOTHING](#).
- result* A pre-existing Atom that will hold the type and value of the result of evaluating the expression.

Returns

Remarks

Evaluates the expression in an expr object with arguments in argv and returns the type and value of the evaluated expression as a *t_atom* in result. result need only point to a single *t_atom*, but argv should contain at least argc Atoms. If, as in the example shown above under [expr_new\(\)](#), there are “gaps” between arguments, they should be filled in with *t_atom* of type [A NOTHING](#).

33.26.3.2 void* expr_new (*short argc*, *t_atom *argv*, *t_atom *types*)

Create a new expr object.

Parameters

- argc* Count of arguments in argv.
- argv* Arguments that are used to create the expr. See the example below for details.
- types* A pre-existing array of nine *t_atoms*, that will hold the types of any variable arguments created in the expr. The types are returned in the *a_type* field of each *t_atom*. If an argument was not present, [A NOTHING](#) is returned.

Returns

[expr_new\(\)](#) creates an expr object from the arguments in argv and returns the type of any expr-style arguments contained in argv (i.e. \$i1, etc.) in atoms in an array pointed to by types.

Remarks

types should already exist as an array of nine Atoms, all of which will be filled in by [expr_new\(\)](#). If an argument was not present, it will set to type [A NOTHING](#). For example, suppose argv pointed to the following atoms:

```
$i1 (A_SYM)
+
$f3 (A_SYM)
+
3 (A_LONG)
```

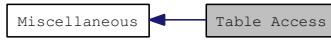
After calling [expr_new](#), types would contain the following:

Index	Argument	Type	Value
0	1 (\$i1)	A_LONG	0
1	2	A_NOTHING	0
2	3 (\$f3)	A_FLOAT	0.0
3	4	A_NOTHING	0
4	5	A_NOTHING	0
5	6	A_NOTHING	0
6	7	A_NOTHING	0
7	8	A_NOTHING	0
8	9	A_NOTHING	0

33.27 Table Access

You can use these functions to access named table objects.

Collaboration diagram for Table Access:



Functions

- short `table_get (t_symbol *s, long ***hp, long *sp)`
Get a handle to the data in a named table object.
- short `table_dirty (t_symbol *s)`
Mark a table object as having changed data.

33.27.1 Detailed Description

You can use these functions to access named table objects. Tables have names when the user creates a table with an argument.

The scenario for knowing the name of a table but not the object itself is if you were passed a `t_symbol`, either as an argument to your creation function or in some message, with the implication being "do your thing with the data in the table named norris."

33.27.2 Function Documentation

33.27.2.1 short `table_dirty (t_symbol * s)`

Mark a table object as having changed data.

Parameters

`s` Symbol containing the name of a table object.

Returns

If no table is associated with tableName, table_dirty returns a non-zero result.

33.27.2.2 short `table_get (t_symbol * s, long *** hp, long * sp)`

Get a handle to the data in a named table object.

Parameters

`s` Symbol containing the name of the table object to find.

`hp` Address of a handle where the table's data will be returned if the named table object is found.

`sp` Number of elements in the table (its size in longs).

Returns

If no table object is associated with the symbol tableName, [table_get\(\)](#) returns a non-zero result.

Remarks

table_get searches for a table associated with the [t_symbol](#) tableName. If one is found, a Handle to its elements (stored as an array of long integers) is returned and the function returns 0. Never count on a table to exist across calls to one of your methods. Call table_get and check the result each time you wish to use a table.

Here is an example of retrieving the 40th element of a table:

```
long **storage,size,value;
if (!table_get(gensym("somename"),&storage,&size)) {
    if (size > 40)
        value = *((*storage)+40);
}
```

33.28 Text Editor Windows

Max has a simple built-in text editor object that can display and edit text in conjunction with your object.

Collaboration diagram for Text Editor Windows:



Max has a simple built-in text editor object that can display and edit text in conjunction with your object. The routines described here let you create a text editor.

When the editor window is about to be closed, your object could receive as many as three messages. The first one, okclose, will be sent if the user has changed the text in the window. This is the standard okclose message that is sent to all "dirty" windows when they are about to be closed, but the text editor window object passes it on to you instead of doing anything itself. Refer to the section on Window Messages for a description of how to write a method for the okclose message. It's not required that you write one—if you don't, the behavior of the window will be determined by the setting of the window's w_scratch bit. If it's set, no confirmation will be asked when a dirty window is closed (and no okclose message will be sent to the text editor either). The second message, edclose, requires a method that should be added to your object at initialization time. The third message, edSave, allows you to gain access to the text before it is saved, or save it yourself.

See also

[Showing a Text Editor](#)

33.29 Presets

Max contains a preset object that has the ability to send preset messages to some or all of the objects (clients) in a Patcher window.

Collaboration diagram for Presets:



Functions

- void [preset_store](#) (char *fmt,...)

Give the preset object a general message to restore the current state of your object.

- void [preset_set](#) ([t_object](#) *obj, long val)

Restore the state of your object with a set message.

- void [preset_int](#) (void *x, long n)

Restore the state of your object with an int message.

33.29.1 Detailed Description

Max contains a preset object that has the ability to send preset messages to some or all of the objects (clients) in a Patcher window. The preset message, sent when the user is storing a preset, is just a request for your object to tell the preset object how to restore your internal state to what it is now. Later, when the user executes a preset, the preset object will send you back the message you had previously said you wanted.

The dialog goes something like this:

- During a store... preset object to Client object(s): hello, this is the preset message—tell me how to restore your state
- Client object to preset object: send me int 34 (for example)
- During an execute... preset object to Client object: int 34

The client object won't know the difference between receiving int 34 from a preset object and receiving a 34 in its leftmost inlet.

It's not mandatory for your object to respond to the preset message, but it is something that will make users happy. All Max user interface objects currently respond to preset messages. Note that if your object is not a user interface object and implements a preset method, the user will need to connect the outlet of the preset object to its leftmost inlet in order for it to be sent a preset message when the user stores a preset.

Here's an example of using [preset_store\(\)](#) that specifies that the object would like to receive a set message. We assume it has one field, myvalue, which it would like to save and restore.

```

void myobject_preset(myobject *x)
{
    preset_store("ossl", x, ob_sym(x), gensym("set"), x->myvalue);
}

```

When this preset is executed, the object will receive a set message whose argument will be the value of myvalue. Note that the same thing can be accomplished more easily with [preset_set\(\)](#) and [preset_int\(\)](#).

Don't pass more than 12 items to [preset_store\(\)](#). If you want to store a huge amount of data in a preset, use [binbuf_insert\(\)](#).

The following example locates the Binbuf into which the preset data is being collected, then calls [binbuf_insert\(\)](#) on a previously prepared array of Atoms. It assumes that the state of your object can be restored with a set message.

```
void myobject_preset(myObject *x)
{
    void *preset_buf; // Binbuf that stores the preset
    short atomCount; // number of atoms you're storing
    t_atom atomArray[SOMESIZE]; // array of atoms to be stored

    // 1. prepare the preset "header" information
    atom_setobj(atomArray,x);
    atom_setsym(atomArray+1,ob_sym(x));
    atom_setsym(atomArray+2,gensym("set"));
    // fill atomArray+3 with object's state here and set atomCount

    // 2. find the Binbuf
    preset_buf = gensym("_preset")->s_thing;

    // 3. store the data
    if (preset_buf) {
        binbuf_insert(preset_buf,NIL,atomCount,atomArray);
    }
}
```

33.29.2 Function Documentation

33.29.2.1 void preset_int (void * *x*, long *n*)

Restore the state of your object with an int message.

This function causes an int message with the argument value to be sent to your object from the preset object when the user executes a preset. All of the existing user interface objects use the int message for restoring their state when a preset is executed.

Parameters

- x* Your object.
- n* Current value of your object.

33.29.2.2 void preset_set (t_object * *obj*, long *val*)

Restore the state of your object with a set message.

This function causes a set message with the argument value to be sent to your object from the preset object when the user executes a preset.

Parameters

- obj* Your object.
- val* Current value of your object.

33.29.2.3 void preset_store (char **fmt*, ...)

Give the preset object a general message to restore the current state of your object.

This is a general preset function for use when your object's state cannot be restored with a simple int or set message. The example below shows the expected format for specifying what your current state is to a preset object. The first thing you supply is your object itself, followed by the symbol that is the name of your object's class (which you can retrieve from your object using the macro ob_sym, declared in ext_mess.h). Next, supply the symbol that specifies the message you want receive (a method for which had better be defined in your class), followed by the arguments to this message—the current values of your object's fields.

Parameters

fmt C string containing one or more letters corresponding to the types of each element of the message. s for Symbol, l for long, or f for float.

... Elements of the message used to restore the state of your object, passed directly to the function as Symbols, longs, or floats. See below for an example that conforms to what the preset object expects.

33.30 Event and File Serial Numbers

If you call `outlet_int()`, `outlet_float()`, `outlet_list()`, or `outlet_anything()` inside a Qelem or during some idle or interrupt time, you should increment Max's Event Serial Number beforehand.

Collaboration diagram for Event and File Serial Numbers:



Functions

- `void evnum_incr (void)`
Increment the event serial number.
- `long evnum_get (void)`
Get the current value of the event serial number.
- `long serialno (void)`
Get a unique number for each Patcher file saved.

33.30.1 Detailed Description

If you call `outlet_int()`, `outlet_float()`, `outlet_list()`, or `outlet_anything()` inside a Qelem or during some idle or interrupt time, you should increment Max's Event Serial Number beforehand. This number can be read by objects that want to know if two messages they have received occurred at the same logical "time" (in response to the same event). Max increments the serial number for each tick of the clock, each key press, mouse click, and MIDI event. Note that this is different from the file serial number returned by the `serialno()` function. The file serial number is only incremented when patchers are saved in files. If more than one patcher is saved in a file, the file serial number will change but the event serial number will not.

33.30.2 Using Event Serial Numbers

Here is a Max patch that includes an object called simul that would use the information returned by `evnum_get` to return a 1 if the right and left inlets receive messages at the same time, 0 if not. The number boxes below show the results of clicking on the button objects or typing a key.

33.30.3 Function Documentation

33.30.3.1 long evnum_get (void)

Get the current value of the event serial number.

Returns

The current value of the event serial number.

33.30.3.2 long serialno (void)

Get a unique number for each Patcher file saved.

This function returns a serial number that is incremented each time a Patcher file is saved. This routine is useful for objects like table and coll that have multiple objects that refer to the same data, and can embed the data inside a Patcher file. If the serial number hasn't changed since your object was last saved, you can detect this and avoid saving multiple copies of the object's data.

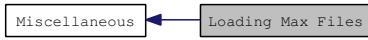
Returns

The serial number.

33.31 Loading Max Files

Several high-level functions permit you to load patcher files.

Collaboration diagram for Loading Max Files:



Functions

- short `readtohandle` (C74_CONST char *name, short volume, char ***h, long *sizep)

Load a data file into a handle.
- void * `fileload` (C74_CONST char *name, short vol)

Load a patcher file by name and volume reference number.
- void * `intload` (C74_CONST char *name, short volume, t_symbol *s, short ac, t_atom *av, short couldedit)

Pass arguments to Max files when you open them.
- void * `stringload` (C74_CONST char *name)

Load a patcher file located in the Max search path by name.

33.31.1 Detailed Description

Several high-level functions permit you to load patcher files. These can be used in sophisticated objects that use Patcher objects to perform specific tasks.

33.31.2 Function Documentation

33.31.2.1 void* `fileload` (C74_CONST char * *name*, short *vol*)

Load a patcher file by name and volume reference number.

Parameters

name Filename of the patcher file to load (C string).

vol Path ID specifying the location of the file.

Returns

If the file is found, `fileload` tries to open the file, evaluate it, open a window, and bring it to the front. A pointer to the newly created Patcher is returned if loading is successful, otherwise, if the file is not found or there is insufficient memory, zero is returned.

33.31.2.2 void* intload (C74_CONST char * *name*, short *volume*, t_symbol * *s*, short *ac*, t_atom * *av*, short *couldedit*)

Pass arguments to Max files when you open them.

This function loads the specified file and returns a pointer to the created object. Historically, [intload\(\)](#) is used to open patcher files, whether they are in text or Max binary format. It could also open table files whose contents begin with the word "table".

Parameters

name Name of the file to open.

volume Path ID specifying the location of the file.

s A symbol.

ac Count of t_atoms in av. To properly open a patcher file, ac should be 9.

av Array of t_atoms that will replace the changeable arguments 1-9. The default behavior could be to set all these to t_atoms of type [A_LONG](#) with a value of 0.

couldedit If non-zero and the file is not a patcher file, the file is opened as a text file.

Returns

If couldedit is non-zero and the file is not a patcher file, it is made into a text editor, and intload() returns 0. If couldedit is non-zero, [intload\(\)](#) will alert the user to an error and return 0. If there is no error, the value returned will be a pointer to a patcher or table object.

33.31.2.3 short readtohandle (C74_CONST char * *name*, short *volume*, char * *h*, long * *sizep*)**

Load a data file into a handle.

This is a low-level routine used for reading text and data files. You specify the file's name and Path ID, as well as a pointer to a Handle.

Parameters

name Name of the patcher file to load.

volume Path ID specifying the location of the file.

h Pointer to a handle variable that will receive the handle that contains the data in the file.

sizep Size of the handle returned in h.

Returns

If the file is found, readtohandle creates a Handle, reads all the data in the file into it, assigns the handle to the variable hp, and returns the size of the data in size. readtohandle returns 0 if the file was opened and read successfully, and non-zero if there was an error.

33.31.2.4 void* stringload (C74_CONST char * *name*)

Load a patcher file located in the Max search path by name.

This function searches for a patcher file, opens it, evaluates it as a patcher file, opens a window for the patcher and brings it to the front. You need only specify a filename and Max will look through its search path for the file. The search path begins with the current 'default volume' that is often the volume of the last opened patcher file, then the folders specified in the File Preferences dialog, searched depth first, then finally the folder that contains the Max application.

Parameters

name Filename of the patcher file to load (C string).

Returns

If [stringload\(\)](#) returns a non-zero result, you can later use [freeobject\(\)](#) to close the patcher, or just let users do it themselves. If [stringload\(\)](#) returns zero, no file with the specified name was found or there was insufficient memory to open it.

33.32 Monitors and Displays

Functions for finding our information about the environment.

Collaboration diagram for Monitors and Displays:



Functions

- long `jmonitor_getnumdisplays ()`
Return the number of monitors on which can be displayed.
- void `jmonitor_getdisplayrect (long workarea, long displayindex, t_rect *rect)`
Return the `t_rect` for a given display.
- void `jmonitor_getdisplayrect_foralldisplays (long workarea, t_rect *rect)`
Return a union of all display rects.
- void `jmonitor_getdisplayrect_forpoint (long workarea, t_pt pt, t_rect *rect)`
Return the `t_rect` for the display on which a point exists.

33.32.1 Detailed Description

Functions for finding our information about the environment.

33.32.2 Function Documentation

33.32.2.1 void `jmonitor_getdisplayrect (long workarea, long displayindex, t_rect * rect)`

Return the `t_rect` for a given display.

Parameters

- `workarea` Set workarea non-zero to clip out things like dock / task bar.
`displayindex` The index number for a monitor. The primary monitor has an index of 0.
`rect` The address of a valid `t_rect` whose values will be filled-in upon return.

33.32.2.2 void `jmonitor_getdisplayrect_foralldisplays (long workarea, t_rect * rect)`

Return a union of all display rects.

Parameters

- `workarea` Set workarea non-zero to clip out things like dock / task bar.
`rect` The address of a valid `t_rect` whose values will be filled-in upon return.

33.32.2.3 void jmonitor_getdisplayrect_forpoint (long *workarea*, t_pt *pt*, t_rect * *rect*)

Return the [t_rect](#) for the display on which a point exists.

Parameters

workarea Set workarea non-zero to clip out things like dock / task bar.

pt A point, for which the monitor will be determined and the rect returned.

rect The address of a valid [t_rect](#) whose values will be filled-in upon return.

33.32.2.4 long jmonitor_getnumdisplays ()

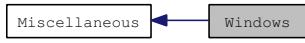
Return the number of monitors on which can be displayed.

Returns

The number of monitors.

33.33 Windows

Collaboration diagram for Windows:



Functions

- **t_object * jwind_getactive (void)**
Get the current window, if any.
- **long jwind_getcount (void)**
Determine how many windows exist.
- **t_object * jwind_getat (long index)**
Return a pointer to the window with a given index.

33.33.1 Function Documentation

33.33.1.1 t_object* jwind_getactive (void)

Get the current window, if any.

Returns

A pointer to the current window, if there is one. Otherwise returns NULL.

33.33.1.2 t_object* jwind_getat (long index)

Return a pointer to the window with a given index.

Parameters

index Get window at index (0 to count-1).

Returns

A pointer to a window object.

33.33.1.3 long jwind_getcount (void)

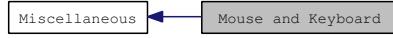
Determine how many windows exist.

Returns

The number of windows.

33.34 Mouse and Keyboard

Collaboration diagram for Mouse and Keyboard:



Enumerations

- enum t_modifiers {
 eCommandKey = 1,
 eShiftKey = 2,
 eControlKey = 4,
 eAltKey = 8,
 eLeftButton = 16,
 eRightButton = 32,
 eMiddleButton = 64,
 ePopupMenu = 128,
 eCapsLock = 256,
 eAutoRepeat = 512 }

Bit mask values for various meta-key presses on the keyboard.

- enum t_jmouse_cursortype {
 JMOUSE_CURSOR_NONE,
 JMOUSE_CURSOR_ARROW,
 JMOUSE_CURSOR_WAIT,
 JMOUSE_CURSOR_IBEAM,
 JMOUSE_CURSOR_CROSSHAIR,
 JMOUSE_CURSOR_COPYING,
 JMOUSE_CURSOR_POINTINGHAND,
 JMOUSE_CURSOR_DRAGGINGHAND,
 JMOUSE_CURSOR_RESIZE_LEFTRIGHT,
 JMOUSE_CURSOR_RESIZE_UPDOWN,
 JMOUSE_CURSOR_RESIZE_FOURWAY,
 JMOUSE_CURSOR_RESIZE_TOPEDGE,
 JMOUSE_CURSOR_RESIZE_BOTTOMEDGE,
 JMOUSE_CURSOR_RESIZE_LEFTEDGE,
 JMOUSE_CURSOR_RESIZE_RIGHTEDGE,
 JMOUSE_CURSOR_RESIZE_TOPLEFTCORNER,
 JMOUSE_CURSOR_RESIZE_TOPRIGHTCORNER,
 JMOUSE_CURSOR_RESIZE_BOTTOMLEFTCORNER,
 JMOUSE_CURSOR_RESIZE_BOTTOMRIGHTCORNER }

Mouse cursor types.

Functions

- **t_modifiers jkeyboard_getcurrentmodifiers ()**
Return the last known combination of modifier keys being held by the user.
- **t_modifiers jkeyboard_getcurrentmodifiers_realtime ()**
Return the current combination of modifier keys being held by the user.
- **void jmouse_getposition_global (int *x, int *y)**
Get the position of the mouse cursor in screen coordinates.
- **void jmouse_setposition_global (int x, int y)**
Set the position of the mouse cursor in screen coordinates.
- **void jmouse_setposition_view (t_object *patcherview, double cx, double cy)**
Set the position of the mouse cursor relative to the patcher canvas coordinates.
- **void jmouse_setposition_box (t_object *patcherview, t_object *box, double bx, double by)**
Set the position of the mouse cursor relative to a box within the patcher canvas coordinates.
- **void jmouse_setcursor (t_object *patcherview, t_object *box, t_jmouse_cursortype type)**
Set the mouse cursor.

33.34.1 Enumeration Type Documentation

33.34.1.1 enum t_jmouse_cursortype

Mouse cursor types.

Enumerator:

- JMOUSE_CURSOR_NONE** None.
- JMOUSE_CURSOR_ARROW** Arrow.
- JMOUSE_CURSOR_WAIT** Wait.
- JMOUSE_CURSOR_IBEAM** I-Beam.
- JMOUSE_CURSOR_CROSSHAIR** Crosshair.
- JMOUSE_CURSOR_COPYING** Copying.
- JMOUSE_CURSOR_POINTINGHAND** Pointing Hand.
- JMOUSE_CURSOR_DRAGGINGHAND** Dragging Hand.
- JMOUSE_CURSOR_RESIZE_LEFTRIGHT** Left-Right.
- JMOUSE_CURSOR_RESIZE_UPDOWN** Up-Down.
- JMOUSE_CURSOR_RESIZE_FOURWAY** Four Way.
- JMOUSE_CURSOR_RESIZE_TOPEDGE** Top Edge.
- JMOUSE_CURSOR_RESIZE_BOTTOMEDGE** Bottom Edge.
- JMOUSE_CURSOR_RESIZE_LEFTEDGE** Left Edge.
- JMOUSE_CURSOR_RESIZE_RIGHTEDGE** Right Edge.

JMOUSE_CURSOR_RESIZE_TOPLEFTCORNER Top-Left Corner.
JMOUSE_CURSOR_RESIZE_TOPRIGHTCORNER Top-Right Corner.
JMOUSE_CURSOR_RESIZE_BOTTOMLEFTCORNER Bottom-Left Corner.
JMOUSE_CURSOR_RESIZE_BOTTOMRIGHTCORNER Bottom-Right Corner.

33.34.1.2 enum t_modifiers

Bit mask values for various meta-key presses on the keyboard.

Enumerator:

eCommandKey Command Key.
eShiftKey Shift Key.
eControlKey Control Key.
eAltKey Alt Key.
eLeftButton Left mouse button.
eRightButton Right mouse button.
eMiddleButton Middle mouse button.
ePopupMenu Popup Menu (contextual menu requested).
eCapsLock Caps lock.
eAutoRepeat Key is generated by key press auto-repeat.

33.34.2 Function Documentation

33.34.2.1 t_modifiers jkeyboard_getcurrentmodifiers ()

Return the last known combination of modifier keys being held by the user.

Returns

The current modifier keys that are activated.

33.34.2.2 t_modifiers jkeyboard_getcurrentmodifiers_realtime ()

Return the current combination of modifier keys being held by the user.

Returns

The current modifier keys that are activated.

33.34.2.3 void jmouse_getposition_global (int * x, int * y)

Get the position of the mouse cursor in screen coordinates.

Parameters

- x** The address of a variable to hold the x-coordinate upon return.
- y** The address of a variable to hold the y-coordinate upon return.

33.34.2.4 void jmouse_setcursor (t_object *patcherview, t_object *box, t_jmouse_cursortype type)

Set the mouse cursor.

Parameters

patcherview The patcherview for which the cursor should be applied.

box The box for which the cursor should be applied.

type The type of cursor for the mouse to use.

33.34.2.5 void jmouse_setposition_box (t_object *patcherview, t_object *box, double bx, double by)

Set the position of the mouse cursor relative to a box within the patcher canvas coordinates.

Parameters

patcherview The patcherview containing the box upon which the mouse coordinates are based.

box The box upon which the mouse coordinates are based.

bx The new x-coordinate of the mouse cursor position.

by The new y-coordinate of the mouse cursor position.

33.34.2.6 void jmouse_setposition_global (int x, int y)

Set the position of the mouse cursor in screen coordinates.

Parameters

x The new x-coordinate of the mouse cursor position.

y The new y-coordinate of the mouse cursor position.

33.34.2.7 void jmouse_setposition_view (t_object *patcherview, double cx, double cy)

Set the position of the mouse cursor relative to the patcher canvas coordinates.

Parameters

patcherview The patcherview upon which the mouse coordinates are based.

cx The new x-coordinate of the mouse cursor position.

cy The new y-coordinate of the mouse cursor position.

33.35 Example Projects

Files

- file [attrtester.c](#)
attrtester - a max object shell jeremy bernstein - jeremy@bootsquad.com
- file [collect.cpp](#)
collect - collect numbers and operate on them.
- file [dbcuelist.c](#)
dbcuelist - demonstrate use of a sqlite database
- file [dbviewer.c](#)
dbviewer - demonstrate use of database views for sqlite and jdataview
- file [delay2.c](#)
delay2 - an ITM-based delay
- file [dspstress~.c](#)
dspstress~ - very simple msp object that does nothing except eat up a specified % of processor time
- file [dummy.c](#)
dummy - a dummy object jeremy bernstein - jeremy@bootsquad.com
- file [filebyte.c](#)
filebyte - similar to filein.
- file [iterate2.c](#)
iterate2 - object that iterates through a patcher and its subpatchers jeremy bernstein - jeremy@bootsquad.com
- file [iterator.c](#)
iterator - patch iterator jeremy bernstein - jeremy@bootsquad.com
- file [linky.c](#)
linky - Demonstrates functions of the `t_linklist` object.
- file [pictmeter~.c](#)
pictmeter~ - audio meter that works by resizing an image
- file [plussz.c](#)
plussz.c - one of the simplest max objects you can make -rdd 2001 (plussz is/was the name of a Hungarian vitamin C tablet-drink from the early 90s)
- file [plussz2.c](#)
plussz2.c - a version of plussz2 that demonstrates the use of proxy inlets.
- file [plussz~.c](#)
plussz~ - a very simple example of a basic MSP object

- file [scripto.c](#)

scripto - patcher scripting from C example scripto makes a custom UI object and then puts it in a window -- so it is similar to the old "kalim" example

- file [sheep.c](#)

sheep - demonstrates object registration and notification, and some hashtab techniques for use with the "shepherd" object jeremy bernstein - jeremy@bootsquad.com

- file [shepherd.c](#)

shepherd - demonstrates object registration and notification, and some hashtab techniques for use with the "sheep" object jeremy bernstein - jeremy@bootsquad.com

- file [jit.simple.cpp](#)

jit.simple - simple example of a Jitter external + multiplies an incoming matrix by a constant + demonstrates some oft-requested example code for using C++ in an extern

- file [max.jit.simple.c](#)

max.jit.simple - simple example of a Jitter external multiplies an incoming matrix by a constant

- file [jit.simple~.cpp](#)

max.jit.simple~ - simple example of an MSP+Jitter combination external.

- file [max.jit.simple~.c](#)

jit.simple - simple example of a Jitter external multiplies an incoming matrix by a constant

- file [simplejs.c](#)

simplejs - a max object used as a JavaScript class

- file [simplemax.c](#)

simplemax - a max object shell jeremy bernstein - jeremy@bootsquad.com

- file [simplemsp~.c](#)

simplemsp - an MSP object shell jeremy bernstein - jeremy@bootsquad.com

- file [simpletext.c](#)

simpletext - show use of text reading and editing

- file [simpwave~.c](#)

simpwave~ - a simple wavetable oscillator using buffer~

- file [uioptimized.c](#)

uioptimized - demonstrate the drawing of various objects using jgraphics

- file [uisimp2.c](#)

uisimp - a very simple ui object - step 4

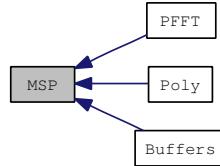
- file [uisimp3.c](#)

uisimp - a very simple ui object - step 5

- file [uisimp4.c](#)
uisimp - a very simple ui object - step 6
- file [uisimp5.c](#)
uisimp - a very simple ui object - step 7
- file [uitester.c](#)
uitester - demonstrate the drawing of various objects using jgraphics
- file [uitextfield.c](#)
uitextfield - demonstrate the textfield with keyboard input
- file [urner.c](#)
urner - a max object shell jeremy bernstein - jeremy@bootsquad.com
- file [whosyourdaddy.c](#)
whosyourdaddy - who's the parent patcher jeremy bernstein - jeremy@bootsquad.com
- file [windowwatcher.c](#)
windowwatcher - Demonstrate how to get notifications about the window for a patcher in which an object exists.

33.36 MSP

Collaboration diagram for MSP:



Data Structures

- struct [t_pxobject](#)
Header for any non-ui signal processing object.
- struct [t_signal](#)
The signal data structure.
- struct [t_pxjbox](#)
Header for any ui signal processing object.

Modules

- [Buffers](#)
Your object can access shared data stored in an MSP buffer~ object.
- [PFFT](#)
When an object is instantiated, it is possible to determine if it is being created in pfft~ context in the new method.
- [Poly](#)
If your object is instantiated as a voice of a poly~ object, it is possible both to determine this context and to determine information about the specific voice.

Defines

- #define [Z_NO_INPLACE](#) 1
flag indicating the object doesn't want signals in place
- #define [Z_PUT_LAST](#) 2
when list of ugens is resorted, put this object at end
- #define [Z_PUT_FIRST](#) 4
when list of ugens is resorted, put this object at beginning
- #define [PI](#) 3.14159265358979323846

The pi constant.

- #define **TWOPI** 6.28318530717958647692

Twice the pi constant.

- #define **PIOVERTWO** 1.57079632679489661923

Half of the pi constant.

- #define **dsp_setup** **z_dsp_setup**

This is commonly used rather than directly calling [z_dsp_setup\(\)](#) in MSP objects.

- #define **dsp_free** **z_dsp_free**

This is commonly used rather than directly calling [z_dsp_free\(\)](#) in MSP objects.

Typedefs

- typedef int **t_int**

An integer.

- typedef float **t_float**

A float.

- typedef float **t_sample**

A sample value.

- typedef **t_int** **(* t_perfroutine)**(**t_int** *args)

A function pointer for the audio perform routine used by MSP objects to process blocks of samples.

- typedef void * **t_vptr**

A void pointer.

- typedef void * **vptr**

A void pointer.

Enumerations

- enum {

SYS_MAXBLKSIZE = 2048,

SYS_MAXSIGS = 250 }

MSP System Properties.

Functions

- int **sys_getmaxblksize** (void)
Query MSP for the maximum global vector (block) size.
- int **sys_getblksize** (void)
Query MSP for the current global vector (block) size.
- float **sys_getsr** (void)
Query MSP for the global sample rate.
- int **sys_getdspstate** (void)
Query MSP to determine whether or not it is running.
- int **sys_getdspobjdspstate** (**t_object** *o)
Query MSP to determine whether or not a given audio object is in a running dsp chain.
- void **dsp_add** (**t_perfroutine** f, int n,...)
Call this function in your MSP object's dsp method.
- void **dsp_adadv** (**t_perfroutine** f, int n, void **vector)
Call this function in your MSP object's dsp method.
- void **z_dsp_setup** (**t_pxobject** *x, long nsignals)
*Call this routine after creating your object in the new instance routine with **object_alloc()**.*
- void **z_dsp_free** (**t_pxobject** *x)
*This function disposes of any memory used by proxies allocated by **dsp_setup()**.*
- void **class_dspinit** (**t_class** *c)
This routine must be called in your object's initialization routine.
- void **class_dspinitjbox** (**t_class** *c)
This routine must be called in your object's initialization routine.

33.36.1 Define Documentation

33.36.1.1 #define PI 3.14159265358979323846

The pi constant.

33.36.1.2 #define PIOVERTWO 1.57079632679489661923

Half of the pi constant.

33.36.1.3 #define TWOPI 6.28318530717958647692

Twice the pi constant.

33.36.2 Typedef Documentation

33.36.2.1 `typedef float t_float`

A float.

33.36.2.2 `typedef int t_int`

An integer.

33.36.2.3 `typedef t_int*(* t_perfroutine)(t_int *args)`

A function pointer for the audio perform routine used by MSP objects to process blocks of samples.

33.36.2.4 `typedef float t_sample`

A sample value.

33.36.2.5 `typedef void* t_vptr`

A void pointer.

33.36.2.6 `typedef void* vptr`

A void pointer.

33.36.3 Enumeration Type Documentation

33.36.3.1 `anonymous enum`

MSP System Properties.

Enumerator:

`SYS_MAXBLKSIZE` a good number for a maximum signal vector size

`SYS_MAXSIGS` number of signal inlets you can have in an object

33.36.4 Function Documentation

33.36.4.1 `void class_dspinit (t_class * c)`

This routine must be called in your object's initialization routine.

It adds a set of methods to your object's class that are called by MSP to build the DSP call chain. These methods function entirely transparently to your object so you don't have to worry about them. However, you should avoid binding anything to their names: signal, userconnect, nsiginlets, and enable.

This routine is for non-user-interface objects only (where the first item in your object's struct is a `t_pxobject`). It must be called prior to calling [class_register\(\)](#) for your class.

Parameters

c The class to make dsp-ready.

See also

[class_dspinitjbox\(\)](#)

33.36.4.2 void class_dspinitjbox (*t_class * c*)

This routine must be called in your object's initialization routine.

It adds a set of methods to your object's class that are called by MSP to build the DSP call chain. These methods function entirely transparently to your object so you don't have to worry about them. However, you should avoid binding anything to their names: signal, userconnect, nsiginlets, and enable.

This routine is for user-interface objects only (where the first item in your object's struct is a [*t_jbox*](#)).

Parameters

c The class to make dsp-ready.

See also

[class_dspinit\(\)](#)

33.36.4.3 void dsp_add (*t_perfroutine f, int n, ...*)

Call this function in your MSP object's dsp method.

This function adds your object's perform method to the DSP call chain and specifies the arguments it will be passed. *n*, the number of arguments to your perform method, should be followed by *n* additional arguments, all of which must be the size of a pointer or a long.

Parameters

f The perform routine to use for processing audio.

n The number of arguments that will follow

... The arguments that will be passed to the perform routine.

See also

[The DSP Method and Perform Routine](#)

[Using Connection Information](#)

33.36.4.4 void dsp_addv (*t_perfroutine f, int n, void ** vector*)

Call this function in your MSP object's dsp method.

Use [dsp_addv\(\)](#) to add your object's perform routine to the DSP call chain and specify its arguments in an array rather than as arguments to a function.

Parameters

f The perform routine to use for processing audio.

- n* The number of arguments that will follow in the vector parameter.
- vector* The arguments that will be passed to the perform routine.

See also

[The DSP Method and Perform Routine](#)
[Using Connection Information](#)

33.36.4.5 int sys_getblksize (void)

Query MSP for the current global vector (block) size.

Returns

The current global vector size for the MSP environment.

33.36.4.6 int sys_getdspobjdspstate (t_object * o)

Query MSP to determine whether or not a given audio object is in a running dsp chain.

This is preferable over [sys_getdspstate\(\)](#) since global audio can be on but an object could be in a patcher that is not running.

Returns

Returns true if the MSP object is in a patcher that has audio on, otherwise returns false.

33.36.4.7 int sys_getdspstate (void)

Query MSP to determine whether or not it is running.

Returns

Returns true if the DSP is turned on, otherwise returns false.

33.36.4.8 int sys_getmaxblksize (void)

Query MSP for the maximum global vector (block) size.

Returns

The maximum global vector size for the MSP environment.

33.36.4.9 float sys_getsr (void)

Query MSP for the global sample rate.

Returns

The global sample rate of the MSP environment.

33.36.4.10 void z_dsp_free (*t_pxobject* * *x*)

This function disposes of any memory used by proxies allocated by [dsp_setup\(\)](#).

It also notifies the signal compiler that the DSP call chain needs to be rebuilt if signal processing is active. You should be sure to call this before de-allocating any memory that might be in use by your object's perform routine, in the event that signal processing is on when your object is freed.

Parameters

x The object to free.

See also

[dsp_free](#)

33.36.4.11 void z_dsp_setup (*t_pxobject* * *x*, long *nsignals*)

Call this routine after creating your object in the new instance routine with [object_alloc\(\)](#).

Cast your object to [*t_pxobject*](#) as the first argument, then specify the number of signal inputs your object will have. [dsp_setup\(\)](#) initializes fields of the [*t_pxobject*](#) header and allocates any proxies needed (if num_signal_inputs is greater than 1).

Some signal objects have no inputs; you should pass 0 for num_signal_inputs in this case. After calling [dsp_setup\(\)](#), you can create additional non-signal inlets using [intin\(\)](#), [floatin\(\)](#), or [inlet_new\(\)](#).

Parameters

x Your object's pointer.

nsignals The number of signal/proxy inlets to create for the object.

See also

[dsp_setup](#)

33.37 Buffers

Your object can access shared data stored in an MSP buffer~ object.

Collaboration diagram for Buffers:



Data Structures

- struct [t_buffer](#)
Data structure for the buffer~ object.

33.37.1 Detailed Description

Your object can access shared data stored in an MSP buffer~ object. Similar to table and coll objects, buffer~ objects are bound to a [t_symbol](#) from which you can gain access to the [t_buffer](#) struct. Consider the following example.

```

t_symbol *s;
t_object *o;

s = gensym("foo");
o = s->s_thing;

// if an object is bound to the symbol "foo", then o is that object.
if (ob_sym(o) == gensym("buffer~")) {
    // that object is a buffer~, so we can use it
    x->x_buffer = (t_buffer*)o;

}

```

Having stored a pointer to the buffer~ is the first step toward working with its data. However, you must not go accessing the data directly without taking some precautions regarding thread-safety.

To access the data in a buffer you first increment the `b_inuse` member of the [t_buffer](#)'s struct. Then you perform the requisite operations on the data, which is stored in the `b_samples` member. When you are done you decrement the `b_inuse` member to return it to the state in which you found it.

In the past you may have set the buffer's `b_inuse` flag directly and cleared it when you were done. This is no longer good enough, and you must instead use the threadsafe macros [ATOMIC_INCREMENT](#) and [ATOMIC_DECREMENT](#) for modifying the `b_inuse` flag. The example below demonstrates what this might look like in an MSP object's `perform` routine. Notice that extra care has been taken to ensure that the [ATOMIC_INCREMENT](#) is always balanced with an [ATOMIC_DECREMENT](#) call.

```

ATOMIC_INCREMENT(&x->w_buf->b_inuse);
if (!x->w_buf->b_valid) {
    ATOMIC_DECREMENT(&x->w_buf->b_inuse);
    goto byebye;
}

// do something with the buffer

ATOMIC_DECREMENT(&x->w_buf->b_inuse);
byebye:
    return (w + 7);

```

A class that accesses buffer~ objects is the simpwave~ object that is included with Max 5 SDK example projects.

33.38 PFFT

When an object is instantiated, it is possible to determine if it is being created in pfft~ context in the new method.

Collaboration diagram for PFFT:



Data Structures

- struct [t_pfftpub](#)
Public FFT Patcher struct.

33.38.1 Detailed Description

When an object is instantiated, it is possible to determine if it is being created in pfft~ context in the new method. In the new method (and only at this time), you can check the s_thing member of the [t_symbol](#) '[__pfft~__](#)'. If this is non-null, then you will have a pointer to a [t_pfftpub](#) struct.

```

t_pfftpub *pfft_parent = (t_pfftpub*) gensym("__pfft~__")->s_thing;

if (pfft_parent) {
    // in a pfft~ context
}
else {
    // not in a pfft~
}
  
```

33.39 Poly

If your object is instantiated as a voice of a poly~ object, it is possible both to determine this context and to determine information about the specific voice.

Collaboration diagram for Poly:



If your object is instantiated as a voice of a poly~ object, it is possible both to determine this context and to determine information about the specific voice. This is done by querying the patcher in which your object exists for an associated object, and then calling methods on that object.

```
t_object *patcher = NULL;
t_max_err err = MAX_ERR_NONE;
t_object *assoc = NULL;
method m = NULL;
long voices = -1;
long index = -1;

err = object_obex_lookup(x, gensym("#P"), &patcher);
if (err == MAX_ERR_NONE) {
    object_method(patcher, gensym("getassoc"), &assoc);
    if (assoc) {
        post("found %s", object_classname(assoc)->s_name);

        voices = object_attr_getlong(assoc, gensym("voices"));
        post("total amount of voices: %ld", voices);

        if(m = zgetfn(assoc, gensym("getindex")))
            index = (long)(*m)(assoc, patcher);
        post("index: %ld", index);
    }
}
```

33.40 Objects

Data Structures

- struct [t_messlist](#)

A list of symbols and their corresponding methods, complete with typechecking information.

- struct [t_tinyobject](#)

The tiny object structure sits at the head of any object to which you may pass messages (and which you may feed to [freeobject\(\)](#)).

- struct [t_object](#)

The structure for the head of any object which wants to have inlets or outlets, or support attributes.

Defines

- #define [MAGIC](#) 1758379419L

Magic number used to determine if memory pointed to by a [t_object](#) is valid.*

- #define [NOGOOD](#)(x) (((struct object *)x)->o_magic != MAGIC)

Returns true if a pointer is not a valid object.

- #define [MAXARG](#) 7

Maximum number of arguments that can be passed as a typed-list rather than using [A_GIMME](#).

Functions

- [t_object * newobject_sprintf](#) ([t_object](#) *patcher, C74_CONST char *fmt,...)

Create a new object in a specified patcher with values using a combination of attribute and sprintf syntax.

- [t_object * newobject_fromdictionary](#) ([t_object](#) *patcher, [t_dictionary](#) *d)

Place a new object into a patcher.

- long [object_classname_compare](#) (void *x, [t_symbol](#) *name)

Determines if a particular object is an instance of a given class.

- void * [object_alloc](#) ([t_class](#) *c)

Allocates the memory for an instance of an object class and initialize its object header.

- void * [object_new](#) ([t_symbol](#) *name_space, [t_symbol](#) *classname,...)

Allocates the memory for an instance of an object class and initialize its object header internal to Max.

- void * [object_new_typed](#) ([t_symbol](#) *name_space, [t_symbol](#) *classname, long ac, [t_atom](#) *av)

Allocates the memory for an instance of an object class and initialize its object header internal to Max.

- [t_max_err object_free](#) (void *x)

Call the free function and release the memory for an instance of an internal object class previously instantiated using `object_new()`, `object_new_typed()` or other new-style object constructor functions (e.g.

- `void * object_method (void *x, t_symbol *s,...)`
Sends an untyped message to an object.
- `t_max_err object_method_typed (void *x, t_symbol *s, long ac, t_atom *av, t_atom *rv)`
Sends a type-checked message to an object.
- `t_max_err object_method_typedfun (void **x, t_messlist *mp, t_symbol *s, long ac, t_atom *av, t_atom *rv)`
Currently undocumented.
- `method object_getmethod (void *x, t_symbol *s)`
Retrieves an object's `method` for a particular message selector.
- `t_symbol * object_classname (void *x)`
Retrieves an object instance's class name.
- `void * object_register (t_symbol *name_space, t_symbol *s, void *x)`
Registers an object in a namespace.
- `void * object_findregistered (t_symbol *name_space, t_symbol *s)`
Determines a registered object's pointer, given its namespace and name.
- `t_max_err object_findregisteredbyptr (t_symbol **name_space, t_symbol **s, void *x)`
Determines the namespace and/or name of a registered object, given the object's pointer.
- `void * object_attach (t_symbol *name_space, t_symbol *s, void *x)`
Attaches a client to a registered object.
- `t_max_err object_detach (t_symbol *name_space, t_symbol *s, void *x)`
Detach a client from a registered object.
- `t_max_err object_attach_byptr (void *x, void *registeredobject)`
Attaches a client to a registered object.
- `t_max_err object_attach_byptr_register (void *x, void *object_to_attach, t_symbol *reg_name_space)`
A convenience function wrapping `object_register()` and `object_attach_byptr()`.
- `t_max_err object_detach_byptr (void *x, void *registeredobject)`
Detach a client from a registered object.
- `t_max_err object_unregister (void *x)`
Removes a registered object from a namespace.
- `t_max_err object_notify (void *x, t_symbol *s, void *data)`
Broadcast a message (with an optional argument) from a registered object to any attached client objects.

- `t_class * object_class (void *x)`
Determines the class of a given object.
- `t_max_err object_getvalueof (void *x, long *ac, t_atom **av)`
Retrieves the value of an object which supports the getvalueof/setvalueof interface.
- `t_max_err object_setvalueof (void *x, long ac, t_atom *av)`
Sets the value of an object which supports the getvalueof/setvalueof interface.
- `t_max_err object_obex_lookup (void *x, t_symbol *key, t_object **val)`
Retrieves the value of a data stored in the obex.
- `t_max_err object_obex_store (void *x, t_symbol *key, t_object *val)`
Stores data in the object's obex.
- `void object_obex_dumpout (void *x, t_symbol *s, long argc, t_atom *argv)`
Sends data from the object's dumpout outlet.
- `t_dictionary * object_dictionaryarg (long ac, t_atom *av)`
Retrieve a pointer to a dictionary passed in as an atom argument.
- `t_max_err object_method_parse (t_object *x, t_symbol *s, C74_CONST char *parsestr, t_atom *rv)`
Convenience wrapper for `object_method_typed()` that uses `atom_setparse()` to define the arguments.
- `t_max_err object_method_format (t_object *x, t_symbol *s, t_atom *rv, C74_CONST char *fmt,...)`
Convenience wrapper for `object_method_typed()` that uses `atom_setformat()` to define the arguments.
- `t_max_err object_method_char (t_object *x, t_symbol *s, unsigned char v, t_atom *rv)`
Convenience wrapper for `object_method_typed()` that passes a single char as an argument.
- `t_max_err object_method_long (t_object *x, t_symbol *s, long v, t_atom *rv)`
Convenience wrapper for `object_method_typed()` that passes a single long integer as an argument.
- `t_max_err object_method_float (t_object *x, t_symbol *s, float v, t_atom *rv)`
Convenience wrapper for `object_method_typed()` that passes a single 32bit float as an argument.
- `t_max_err object_method_double (t_object *x, t_symbol *s, double v, t_atom *rv)`
Convenience wrapper for `object_method_typed()` that passes a single 64bit float as an argument.
- `t_max_err object_method_sym (t_object *x, t_symbol *s, t_symbol *v, t_atom *rv)`
Convenience wrapper for `object_method_typed()` that passes a single `t_symbol` as an argument.*
- `t_max_err object_method_obj (t_object *x, t_symbol *s, t_object *v, t_atom *rv)`
Convenience wrapper for `object_method_typed()` that passes a single `t_object` as an argument.*
- `t_max_err object_method_char_array (t_object *x, t_symbol *s, long ac, unsigned char *av, t_atom *rv)`
Convenience wrapper for `object_method_typed()` that passes an array of char values as an argument.

- `t_max_err object_method_long_array (t_object *x, t_symbol *s, long ac, long *av, t_atom *rv)`
Convenience wrapper for `object_method_typed()` that passes an array of long integers values as an argument.
- `t_max_err object_method_float_array (t_object *x, t_symbol *s, long ac, float *av, t_atom *rv)`
Convenience wrapper for `object_method_typed()` that passes an array of 32bit floats values as an argument.
- `t_max_err object_method_double_array (t_object *x, t_symbol *s, long ac, double *av, t_atom *rv)`
Convenience wrapper for `object_method_typed()` that passes an array of 64bit float values as an argument.
- `t_max_err object_method_sym_array (t_object *x, t_symbol *s, long ac, t_symbol **av, t_atom *rv)`
Convenience wrapper for `object_method_typed()` that passes an array of `t_symbol` values as an argument.*
- `t_max_err object_method_obj_array (t_object *x, t_symbol *s, long ac, t_object **av, t_atom *rv)`
Convenience wrapper for `object_method_typed()` that passes an array of `t_object` values as an argument.*
- `void object_openhelp (t_object *x)`
Open the help patcher for a given instance of an object.
- `void object_openrefpage (t_object *x)`
Open the reference page for a given instance of an object.
- `void object_openquery (t_object *x)`
Open a search in the file browser for files with the name of the given object.
- `void classname_openhelp (char *classname)`
Open the help patcher for a given object class name.
- `void classname_openrefpage (char *classname)`
Open the reference page for a given object class name.
- `void classname_openquery (char *classname)`
Open a search in the file browser for files with the name of the given class.

33.40.1 Detailed Description

See also

<http://www.cycling74.com/twiki/bin/view/ProductDocumentation/JitterSdkObjectModel>
<http://www.cycling74.com/twiki/bin/view/ProductDocumentation/JitterSdkRegNotify>

33.40.2 Define Documentation

33.40.2.1 #define MAXARG 7

Maximum number of arguments that can be passed as a typed-list rather than using `A_GIMME`.

It is generally recommended to use `A_GIMME`.

33.40.3 Function Documentation

33.40.3.1 void classname_openhelp (char * *classname*)

Open the help patcher for a given object class name.

Parameters

classname The class name for which to open the help patcher.

33.40.3.2 void classname_openquery (char * *classname*)

Open a search in the file browser for files with the name of the given class.

Parameters

classname The class name for which to query.

33.40.3.3 void classname_openrefpage (char * *classname*)

Open the reference page for a given object class name.

Parameters

classname The class name for which to open the reference page.

33.40.3.4 t_object* newobject_fromdictionary (t_object * *patcher*, t_dictionary * *d*)

Place a new object into a patcher.

The new object will be created based on a specification contained in a [Dictionary](#).

Create a new dictionary populated with values using a combination of attribute and sprintf syntax.

Parameters

patcher An instance of a patcher object.

d A dictionary containing an object specification.

Returns

A pointer to the newly created object instance, or NULL if creation of the object fails.

Remarks

Max attribute syntax is used to define key-value pairs. For example,

```
"@key1 value @key2 another_value"
```

The example below creates a new object that in a patcher whose object pointer is stored in a variable called "aPatcher".

```
t_dictionary *d;
t_object *o;
char text[4];

strncpy_zero(text, "foo", 4);

d = dictionary_sprintf("@maxclass comment @varname _name \
@text \"%s\" @patching_rect %.2f %.2f %.2f %.2f \
@fontsize %f @textcolor %f %f %f 1.0 \
@fontname %s @bgcolor 0.001 0.001 0.001 0.",
text, 20.0, 20.0, 200.0, 24.0,
18, 0.9, 0.9, 0.9, "Arial");

o = newobject_fromdictionary(aPatcher, d);
```

See also

[newobject_sprintf\(\)](#)
[newobject_fromdictionary\(\)](#)
[atom_setparse\(\)](#)

33.40.3.5 t_object* newobject_sprintf (t_object *patcher, C74_CONST char *fmt, ...)

Create a new object in a specified patcher with values using a combination of attribute and sprintf syntax.

Parameters

patcher An instance of a patcher object.
fmt An sprintf-style format string specifying key-value pairs with attribute nomenclature.
... One or more arguments which are to be substituted into the format string.

Returns

A pointer to the newly created object instance, or NULL if creation of the object fails.

Remarks

Max attribute syntax is used to define key-value pairs. For example,

```
"@key1 value @key2 another_value"
```

The example below creates a new object that in a patcher whose object pointer is stored in a variable called "aPatcher".

```
t_object *my_comment;
char text[4];

strncpy_zero(text, "foo", 4);

my_comment = newobject_sprintf(aPatcher, "@maxclass comment @varname _name \
@text \"%s\" @patching_rect %.2f %.2f %.2f %.2f \
@fontsize %f @textcolor %f %f %f 1.0 \
@fontname %s @bgcolor 0.001 0.001 0.001 0.",
text, 20.0, 20.0, 200.0, 24.0,
18, 0.9, 0.9, 0.9, "Arial");
```

See also

[dictionary_sprintf\(\)](#)
[newobject_fromdictionary\(\)](#)
[atom_setparse\(\)](#)

33.40.3.6 void* object_alloc (*t_class * c*)

Allocates the memory for an instance of an object class and initialize its object header.

It is used like the traditional function newobject, inside of an object's new method, but its use is required with obex-class objects.

Parameters

c The class pointer, returned by [class_new\(\)](#)

Returns

This function returns a new instance of an object class if successful, or NULL if unsuccessful.

33.40.3.7 void* object_attach (*t_symbol * name_space, t_symbol * s, void * x*)

Attaches a client to a registered object.

Once attached, the object will receive notifications sent from the registered object (via the [object_notify\(\)](#) function), if it has a `notify` method defined and implemented.

Parameters

name_space The namespace of the registered object. This should be the same value used in [object_register\(\)](#) to register the object. If you don't know the registered object's namespace, the [object_findregisteredbyptr\(\)](#) function can be used to determine it.

s The name of the registered object in the namespace. If you don't know the name of the registered object, the [object_findregisteredbyptr\(\)](#) function can be used to determine it.

x The client object to attach. Generally, this is the pointer to your Max object.

Returns

This function returns a pointer to the registered object (to the object referred to by the combination of *name_space* and *s* arguments) if successful, or NULL if unsuccessful.

Remarks

You should not attach an object to itself if the object is a UI object. UI objects automatically register and attach to themselves in [jbox_new\(\)](#).

See also

[object_notify\(\)](#)
[object_detach\(\)](#)
[object_attach_byptr\(\)](#)
[object_register\(\)](#)

33.40.3.8 *t_max_err object_attach_byptr (void * x, void * registeredobject)*

Attaches a client to a registered object.

Unlike [object_attach\(\)](#), the client is specified by providing a pointer to that object rather than the registered name of that object.

Once attached, the object will receive notifications sent from the registered object (via the [object_notify\(\)](#) function), if it has a `notify` method defined and implemented.

Parameters

- x* The attaching client object. Generally, this is the pointer to your Max object.
- registeredobject* A pointer to the registered object to which you wish to attach.

Returns

A Max error code.

Remarks

You should not attach an object to itself if the object is a UI object. UI objects automatically register and attach to themselves in [jbox_new\(\)](#).

See also

[object_notify\(\)](#)
[object_detach\(\)](#)
[object_attach\(\)](#)
[object_register\(\)](#)
[object_attach_byptr_register\(\)](#)

33.40.3.9 t_max_err object_attach_byptr_register (void * *x*, void * *object_to_attach*, t_symbol * *reg_name_space*)

A convenience function wrapping [object_register\(\)](#) and [object_attach_byptr\(\)](#).

Parameters

- x* The attaching client object. Generally, this is the pointer to your Max object.
- object_to_attach* A pointer to the object to which you wish to registered and then to which to attach.
- reg_name_space* The namespace in which to register the *object_to_attach*.

Returns

A Max error code.

See also

[object_register\(\)](#)
[object_attach_byptr\(\)](#)

33.40.3.10 t_class* object_class (void * *x*)

Determines the class of a given object.

Parameters

- x* The object to test

Returns

This function returns the [t_class](#) * of the object's class, if successful, or NULL, if unsuccessful.

33.40.3.11 t_symbol* object_classname (void *x)

Retrieves an object instance's class name.

Parameters

x The object instance whose class name is being queried

Returns

The classname, or NULL if unsuccessful.

33.40.3.12 long object_classname_compare (void *x, t_symbol *name)

Determines if a particular object is an instance of a given class.

Parameters

x The object to test

name The name of the class to test this object against

Returns

This function returns 1 if the object is an instance of the named class. Otherwise, 0 is returned.

Remarks

For instance, to determine whether an unknown object pointer is a pointer to a print object, one would call:

```
long isprint = object_classname_compare(x, gensym("print"));
```

33.40.3.13 t_max_err object_detach (t_symbol *name_space, t_symbol *s, void *x)

Detach a client from a registered object.

Parameters

name_space The namespace of the registered object. This should be the same value used in [object_register\(\)](#) to register the object. If you don't know the registered object's namespace, the [object_findregisteredbyptr\(\)](#) function can be used to determine it.

s The name of the registered object in the namespace. If you don't know the name of the registered object, the [object_findregisteredbyptr\(\)](#) function can be used to determine it.

x The client object to attach. Generally, this is the pointer to your Max object.

Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.40.3.14 `t_max_err object_detach_byptr (void *x, void *registeredobject)`

Detach a client from a registered object.

Parameters

- x* The attaching client object. Generally, this is the pointer to your Max object.
- registeredobject* The object from which to detach.

Returns

A Max error code.

See also

- [object_detach\(\)](#)
- [object_attach_byptr\(\)](#)

33.40.3.15 `t_dictionary* object_dictionaryarg (long ac, t_atom *av)`

Retrieve a pointer to a dictionary passed in as an atom argument.

Use this function when working with classes that have dictionary constructors to fetch the dictionary.

Parameters

- ac* The number of atoms.
- av* A pointer to the first atom in the array.

Returns

The dictionary retrieved from the atoms.

See also

- [attr_dictionary_process\(\)](#)

33.40.3.16 `void* object_findregistered (t_symbol *name_space, t_symbol *s)`

Determines a registered object's pointer, given its namespace and name.

Parameters

- name_space* The namespace of the registered object
- s* The name of the registered object in the namespace

Returns

This function returns the pointer of the registered object, if successful, or NULL, if unsuccessful.

33.40.3.17 `t_max_err object_findregisteredbyptr (t_symbol ** name_space, t_symbol ** s, void * x)`

Determines the namespace and/or name of a registered object, given the object's pointer.

Parameters

- name_space* Pointer to a `t_symbol *`, to receive the namespace of the registered object
- s* Pointer to a `t_symbol *`, to receive the name of the registered object within the namespace
- x* Pointer to the registered object

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.40.3.18 `t_max_err object_free (void * x)`

Call the free function and release the memory for an instance of an internal object class previously instantiated using `object_new()`, `object_new_typed()` or other new-style object constructor functions (e.g.

`hashtab_new()`). It is, at the time of this writing, a wrapper for the traditional function `freeobject()`, but its use is suggested with obex-class objects.

Parameters

- x* The pointer to the object to be freed.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

33.40.3.19 `method object_getmethod (void * x, t_symbol * s)`

Retrieves an object's `method` for a particular message selector.

Parameters

- x* The object whose method is being queried
- s* The message selector

Returns

This function returns the `method` if successful, or 0 if unsuccessful.

33.40.3.20 `t_max_err object_getvalueof (void * x, long * ac, t_atom ** av)`

Retrieves the value of an object which supports the `getvalueof/setvalueof` interface.

See part 2 of the pattr SDK for more information on this interface.

Parameters

- x* The object whose value is of interest
- ac* Pointer to a long variable to receive the count of arguments in *av*. The long variable itself should be set to 0 previous to calling this function.
- av* Pointer to a `t_atom` *, to receive object data. The `t_atom` * itself should be set to NULL previous to calling this function.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

Remarks

Calling the `object_getvalueof()` function allocates memory for any data it returns. It is the developer's responsibility to free it, using the `freebytes()` function.
Developers wishing to design objects which will support this function being called on them must define and implement a special method, `getvalueof`, like so:

```
class_addmethod(c, (method)myobject_getvalueof, "getvalueof", A_CANT, 0);
```

The `getvalueof` method should be prototyped as:

```
t_max_err myobject_getvalueof(t_myobject *x, long *ac, t_atom **av);
```

And implemented, generally, as:

```
t_max_err myobj_getvalueof(t_myobj *x, long *ac, t_atom **av)
{
    if (ac && av) {
        if (*ac && *av) {
            // memory has been passed in; use it.
        } else {
            // allocate enough memory for your data
            *av = (t_atom *)getbytes(sizeof(t_atom));
        }
        *ac = 1; // our data is a single floating point value
        atom_setfloat(*av, x->objvalue);
    }
    return MAX_ERR_NONE;
}

@remark      By convention, and to permit the interoperability of objects using
the obex API,
            developers should allocate memory in their <tt>getvalueof</tt> met
hods using the getbytes() function.
```

33.40.3.21 void* object_method (void *x, t_symbol *s, ...)

Sends an untyped message to an object.

Parameters

- x* The object that will receive the message
- s* The message selector
- ... Any arguments to the message

Returns

If the receiver object can respond to the message, `object_method()` returns the result. Otherwise, the function will return 0.

Remarks

Example: To send the message bang to the object bang_me:

```
void *bang_result;
bang_result = object_method(bang_me, gensym("bang"));
```

33.40.3.22 t_max_err object_method_char (t_object *x, t_symbol *s, unsigned char v, t_atom *rv)

Convenience wrapper for [object_method_typed\(\)](#) that passes a single char as an argument.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- v* An argument to pass to the method.
- rv* The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

33.40.3.23 t_max_err object_method_char_array (t_object *x, t_symbol *s, long ac, unsigned char *av, t_atom *rv)

Convenience wrapper for [object_method_typed\(\)](#) that passes an array of char values as an argument.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- ac* The number of arguments to pass to the method.
- av* The address of the first of the array of arguments to pass to the method.
- rv* The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

33.40.3.24 t_max_err object_method_double (t_object *x, t_symbol *s, double v, t_atom *rv)

Convenience wrapper for [object_method_typed\(\)](#) that passes a single 64bit float as an argument.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- v* An argument to pass to the method.
- rv* The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

33.40.3.25 t_max_err object_method_double_array (t_object *x, t_symbol *s, long ac, double *av, t_atom *rv)

Convenience wrapper for [object_method_typed\(\)](#) that passes an array of 64bit float values as an argument.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- ac* The number of arguments to pass to the method.
- av* The address of the first of the array of arguments to pass to the method.
- rv* The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

33.40.3.26 t_max_err object_method_float (t_object *x, t_symbol *s, float v, t_atom *rv)

Convenience wrapper for [object_method_typed\(\)](#) that passes a single 32bit float as an argument.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- v* An argument to pass to the method.
- rv* The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

33.40.3.27 t_max_err object_method_float_array (t_object *x, t_symbol *s, long ac, float *av, t_atom *rv)

Convenience wrapper for [object_method_typed\(\)](#) that passes an array of 32bit floats values as an argument.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- ac* The number of arguments to pass to the method.
- av* The address of the first of the array of arguments to pass to the method.
- rv* The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

33.40.3.28 t_max_err object_method_format (t_object *x, t_symbol *s, t_atom *rv, C74_CONST char *fmt, ...)

Convenience wrapper for [object_method_typed\(\)](#) that uses [atom_setformat\(\)](#) to define the arguments.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- rv* The address of an atom to hold a return value.
- fmt* An sprintf-style format string specifying values for the atoms.
- ... One or more arguments which are to be substituted into the format string.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)
[atom_setformat\(\)](#)

33.40.3.29 t_max_err object_method_long (t_object *x, t_symbol *s, long v, t_atom *rv)

Convenience wrapper for [object_method_typed\(\)](#) that passes a single long integer as an argument.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- v* An argument to pass to the method.
- rv* The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

33.40.3.30 t_max_err object_method_long_array (t_object *x, t_symbol *s, long ac, long *av, t_atom *rv)

Convenience wrapper for [object_method_typed\(\)](#) that passes an array of long integers values as an argument.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- ac* The number of arguments to pass to the method.
- av* The address of the first of the array of arguments to pass to the method.
- rv* The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

33.40.3.31 t_max_err object_method_obj (t_object *x, t_symbol *s, t_object *v, t_atom *rv)

Convenience wrapper for [object_method_typed\(\)](#) that passes a single [t_object*](#) as an argument.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- v* An argument to pass to the method.
- rv* The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

33.40.3.32 t_max_err object_method_obj_array (t_object *x, t_symbol *s, long ac, t_object **av, t_atom *rv)

Convenience wrapper for [object_method_typed\(\)](#) that passes an array of `t_object*` values as an argument.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- ac* The number of arguments to pass to the method.
- av* The address of the first of the array of arguments to pass to the method.
- rv* The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

33.40.3.33 t_max_err object_method_parse (t_object *x, t_symbol *s, C74_CONST char *parsestr, t_atom *rv)

Convenience wrapper for [object_method_typed\(\)](#) that uses [atom_setparse\(\)](#) to define the arguments.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- parsestr* A C-string to parse into an array of atoms to pass to the method.
- rv* The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)
[atom_setparse\(\)](#)

33.40.3.34 t_max_err object_method_sym (t_object *x, t_symbol *s, t_symbol *v, t_atom *rv)

Convenience wrapper for [object_method_typed\(\)](#) that passes a single `t_symbol*` as an argument.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- v* An argument to pass to the method.
- rv* The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

33.40.3.35 t_max_err object_method_sym_array (t_object *x, t_symbol *s, long ac, t_symbol **av, t_atom *rv)

Convenience wrapper for [object_method_typed\(\)](#) that passes an array of `t_symbol*` values as an argument.

Parameters

- x* The object to which the message will be sent.
- s* The name of the method to call on the object.
- ac* The number of arguments to pass to the method.
- av* The address of the first of the array of arguments to pass to the method.
- rv* The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

33.40.3.36 t_max_err object_method_typed (void *x, t_symbol *s, long ac, t_atom *av, t_atom *rv)

Sends a type-checked message to an object.

Parameters

- x* The object that will receive the message
- s* The message selector
- ac* Count of message arguments in *av*
- av* Array of `t_atoms`; the message arguments

rv Return value of function, if available

Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

Remarks

If the receiver object can respond to the message, [object_method_typed\(\)](#) returns the result in *rv*. Otherwise, *rv* will contain an [A NOTHING](#) atom.

33.40.3.37 [**t_max_err object_method_typedfun \(void *x, t_messlist *mp, t_symbol *s, long ac, t_atom *av, t_atom *rv\)**](#)

Currently undocumented.

Parameters

- x* The object that will receive the message
- mp* Undocumented
- s* The message selector
- ac* Count of message arguments in *av*
- av* Array of *t_atoms*; the message arguments
- rv* Return value of function, if available

Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

Remarks

If the receiver object can respond to the message, [object_method_typedfun\(\)](#) returns the result in *rv*. Otherwise, *rv* will contain an [A NOTHING](#) atom.

33.40.3.38 [**void* object_new \(t_symbol *name_space, t_symbol *classname, ...\)**](#)

Allocates the memory for an instance of an object class and initialize its object header *internal to Max*.

It is used similarly to the traditional function [newinstance\(\)](#), but its use is required with obex-class objects.

Parameters

- name_space* The desired object's name space. Typically, either the constant [CLASS_BOX](#), for obex classes which can instantiate inside of a Max patcher (e.g. boxes, UI objects, etc.), or the constant [CLASS_NOBOX](#), for classes which will only be used internally. Developers can define their own name spaces as well, but this functionality is currently undocumented.
- classname* The name of the class of the object to be created
- ... Any arguments expected by the object class being instantiated

Returns

This function returns a new instance of the object class if successful, or NULL if unsuccessful.

33.40.3.39 `void* object_new_typed (t_symbol * name_space, t_symbol * classname, long ac, t_atom * av)`

Allocates the memory for an instance of an object class and initialize its object header *internal to Max*.

It is used similarly to the traditional function `newinstance()`, but its use is required with obex-class objects. The `object_new_typed()` function differs from `object_new()` by its use of an atom list for object arguments—in this way, it more resembles the effect of typing something into an object box from the Max interface.

Parameters

name_space The desired object's name space. Typically, either the constant `CLASS_BOX`, for obex classes which can instantiate inside of a Max patcher (e.g. boxes, UI objects, etc.), or the constant `CLASS_NOBOX`, for classes which will only be used internally. Developers can define their own name spaces as well, but this functionality is currently undocumented.

classname The name of the class of the object to be created

ac Count of arguments in `av`

av Array of `t_atoms`; arguments to the class's instance creation function.

Returns

This function returns a new instance of the object class if successful, or `NULL` if unsuccessful.

33.40.3.40 `t_max_err object_notify (void *x, t_symbol *s, void *data)`

Broadcast a message (with an optional argument) from a registered object to any attached client objects.

Parameters

x Pointer to the registered object

s The message to send

data An optional argument which will be passed with the message. Sets this argument to `NULL` if it will be unused.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in `e_max_errorcodes` if unsuccessful.

Remarks

In order for client objects to receive notifications, they must define and implement a special method, `notify`, like so:

```
class_addmethod(c, (method)myobject_notify, "notify", A_CANT, 0);
```

The `notify` method should be prototyped as:

```
void myobject_notify(t_myobject *x, t_symbol *s, t_symbol *msg, void *sender,
                     void *data);
```

where `x` is the pointer to the receiving object, `s` is the name of the sending (registered) object in its namespace, `msg` is the sent message, `sender` is the pointer to the sending object, and `data` is an optional argument sent with the message. This value corresponds to the `data` argument in the `object_notify()` method.

33.40.3.41 void object_obex_dumpout (void *x, t_symbol *s, long argc, t_atom *argv)

Sends data from the object's dumpout outlet.

The dumpout outlet is stored in the obex using the [object_obex_store\(\)](#) function (see above). It is used approximately like [outlet_anything\(\)](#).

Parameters

- x* The object pointer. This function should only be called on instantiated objects (i.e. in the `new` method or later), not directly on classes (i.e. in `main()`).
- s* The message selector [t_symbol *](#)
- argc* Number of elements in the argument list in *argv*
- argv* [t_atoms](#) constituting the message arguments

Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.40.3.42 t_max_err object_obex_lookup (void *x, t_symbol *key, t_object **val)

Retrieves the value of a data stored in the obex.

Parameters

- x* The object pointer. This function should only be called on instantiated objects (i.e. in the `new` method or later), not directly on classes (i.e. in `main()`).
- key* The symbolic name for the data to be retrieved
- val* A pointer to a [t_object](#) *, to be filled with the data retrieved from the obex.

Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

Remarks

By default, pointers to the object's containing patcher and box objects are stored in the obex, under the keys 'P' and 'B', respectively. To retrieve them, the developer could do something like the following:

```
void post_containers(t_obexobj **x)
{
    t_patcher *p;
    t_box *b;
    t_max_err err;

    err = object_obex_lookup(x, gensym("#P"), (t_object **) &p);
    err = object_obex_lookup(x, gensym("#B"), (t_object **) &b);

    post("my patcher is located at 0x%X", p);
    post("my box is located at 0x%X", b);
}
```

33.40.3.43 t_max_err object_obex_store (void *x, t_symbol *key, t_object *val)

Stores data in the object's obex.

Parameters

- x* The object pointer. This function should only be called on instantiated objects (i.e. in the new method or later), not directly on classes (i.e. in main()).
- key* A symbolic name for the data to be stored
- val* A [t_object](#) *, to be stored in the obex, referenced under the key.

Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

Remarks

Most developers will need to use this function for the specific purpose of storing the dumpout outlet in the obex (the dumpout outlet is used by attributes to report data in response to 'get' queries). For this, the developer should use something like the following in the object's new method:

```
object_obex_store(x, _sym_dumpout, outlet_new(x, NULL));
```

33.40.3.44 void object_openhelp (t_object *x)

Open the help patcher for a given instance of an object.

Parameters

- x* The object instance for which to open the help patcher.

33.40.3.45 void object_openquery (t_object *x)

Open a search in the file browser for files with the name of the given object.

Parameters

- x* The object instance for which to query.

33.40.3.46 void object_openrefpage (t_object *x)

Open the reference page for a given instance of an object.

Parameters

- x* The object instance for which to open the reference page.

33.40.3.47 void* object_register (t_symbol * name_space, t_symbol * s, void * x)

Registers an object in a namespace.

Parameters

- name_space*** The namespace in which to register the object. The namespace can be any symbol. If the namespace does not already exist, it is created automatically.
- s*** The name of the object in the namespace. This name will be used by other objects to attach and detach from the registered object.
- x*** The object to register

Returns

The function returns a pointer to the registered object. Under some circumstances, `object_register` will *duplicate* the object, and return a pointer to the duplicate—the developer should not assume that the pointer passed in is the same pointer that has been registered. To be safe, the returned pointer should be stored and used with the `object_unregister()` function.

Remarks

You should not register an object if the object is a UI object. UI objects automatically register and attach to themselves in [jbox_new\(\)](#).

33.40.3.48 t_max_err object_setvalueof (void * x, long ac, t_atom * av)

Sets the value of an object which supports the `getvalueof/setvalueof` interface.

Parameters

- x*** The object whose value is of interest
- ac*** The count of arguments in `av`
- av*** Array of `t_atoms`; the new desired data for the object

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

Remarks

Developers wishing to design objects which will support this function being called on them must define and implement a special method, `setvalueof`, like so:

```
class_addmethod(c, (method)myobject_setvalueof, "setvalueof", A_CANT, 0);
```

The `setvalueof` method should be prototyped as:

```
t_max_err myobject_setvalueof(t_myobject **x, long *ac, t_atom **av);
```

And implemented, generally, as:

```
t_max_err myobject_setvalueof(t_myobject **x, long ac, t_atom **av)
{
    if (ac && av) {
        // simulate receipt of a float value
        myobject_float(x, atom_getfloat(av));
    }
    return MAX_ERR_NONE;
}
```

33.40.3.49 `t_max_err object_unregister (void *x)`

Removes a registered object from a namespace.

Parameters

x The object to unregister. This should be the pointer returned from the [object_register\(\)](#) function.

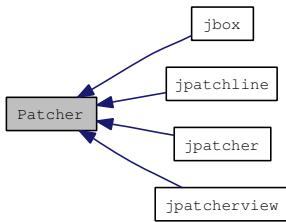
Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in [e_max_errorcodes](#) if unsuccessful.

33.41 Patcher

Max's patcher represents a graph of objects that communicate with messages.

Collaboration diagram for Patcher:



Data Structures

- struct [t_jbox](#)

The [t_jbox](#) struct provides the header for a Max user-interface object.

Modules

- [jpatcher](#)

The patcher.

- [jbox](#)

A box in the patcher.

- [jpatchline](#)

A patch cord.

- [jpatcherview](#)

A view of a patcher.

Typedefs

- typedef [t_object](#) [t_patcher](#)

A patcher.

- typedef [t_object](#) [t_box](#)

A box.

Enumerations

- enum {
 [PI_DEEP](#) = 1,
 }

```
PI_REQUIREFIRSTIN = 2,  
PI_WANTBOX = 4 }  
patcher iteration flags
```

33.41.1 Detailed Description

Max's patcher represents a graph of objects that communicate with messages. This is the public interface to the jpatcher -- the new patcher object in Max 5. The jpatcher is fully controllable via obex attributes and methods.

The jpatcher_api.h header defines constants, enumerations, symbols, structs, and functions for working with the jpatcher. It also includes utility functions for getting/setting attributes and for calling methods. These utilities are just wrapping the obex interface and thus loosely connect your code to the jpatcher implementation.

Finally methods are defined for implementing your own boxes.

33.41.2 Typedef Documentation

33.41.2.1 `typedef t_object t_box`

A box.

As of Max 5, the box struct is opaque. Messages can be sent to a box using `object_method()` or `object_method_typed()`, or by using [Attributes](#) accessors.

33.41.2.2 `typedef t_object t_patcher`

A patcher.

As of Max 5, the patcher struct is opaque. Messages can be sent to a patcher using `object_method()` or `object_method_typed()`, or by using [Attributes](#) accessors.

33.41.3 Enumeration Type Documentation

33.41.3.1 `anonymous enum`

patcher iteration flags

Enumerator:

`PI_DEEP` descend into subpatchers (not used by audio library)

`PI_REQUIREFIRSTIN` if b->b_firstin is NULL, do not call function

`PI_WANTBOX` instead, of b->b_firstin, pass b to function, whether or not b->b_firstin is NULL

33.42 jpatcher

The patcher.

Collaboration diagram for jpatcher:



Functions

- `int jpatcher_is_patcher (t_object *p)`
Determine if a `t_object` is a patcher object.*
- `t_object * jpatcher_get_box (t_object *p)`
If a patcher is inside a box, return its box.
- `long jpatcher_get_count (t_object *p)`
Determine the number of boxes in a patcher.
- `t_max_err jpatcher_set_locked (t_object *p, char c)`
Lock or unlock a patcher.
- `char jpatcher_get_presentation (t_object *p)`
Determine whether a patcher is currently in presentation mode.
- `t_max_err jpatcher_set_presentation (t_object *p, char c)`
Set a patcher to presentation mode.
- `t_object * jpatcher_get_firstobject (t_object *p)`
Get the first box in a patcher.
- `t_object * jpatcher_get_lastobject (t_object *p)`
Get the last box in a patcher.
- `t_object * jpatcher_get_firstline (t_object *p)`
Get the first line (patch-cord) in a patcher.
- `t_object * jpatcher_get_firstview (t_object *p)`
Get the first view (jpatcherview) for a given patcher.
- `t_symbol * jpatcher_get_title (t_object *p)`
Retrieve a patcher's title.
- `t_max_err jpatcher_set_title (t_object *p, t_symbol *ps)`
Set a patcher's title.
- `t_symbol * jpatcher_get_name (t_object *p)`
Retrieve a patcher's name.

- `t_symbol * jpatcher_get_filepath (t_object *p)`
Retrieve a patcher's file path.
- `t_symbol * jpatcher_get_filename (t_object *p)`
Retrieve a patcher's file name.
- `char jpatcher_get_dirty (t_object *p)`
Determine whether a patcher's dirty bit has been set.
- `t_max_err jpatcher_set_dirty (t_object *p, char c)`
Set a patcher's dirty bit.
- `char jpatcher_get_bglocked (t_object *p)`
Determine whether a patcher's background layer is locked.
- `t_max_err jpatcher_set_bglocked (t_object *p, char c)`
Set whether a patcher's background layer is locked.
- `char jpatcher_get_bghidden (t_object *p)`
Determine whether a patcher's background layer is hidden.
- `t_max_err jpatcher_set_bghidden (t_object *p, char c)`
Set whether a patcher's background layer is hidden.
- `char jpatcher_get_fghidden (t_object *p)`
Determine whether a patcher's foreground layer is hidden.
- `t_max_err jpatcher_set_fghidden (t_object *p, char c)`
Set whether a patcher's foreground layer is hidden.
- `t_max_err jpatcher_get_editing_bgcolor (t_object *p, t_jrgba *prgba)`
Retrieve a patcher's editing background color.
- `t_max_err jpatcher_set_editing_bgcolor (t_object *p, t_jrgba *prgba)`
Set a patcher's editing background color.
- `t_max_err jpatcher_get_bgcolor (t_object *p, t_jrgba *prgba)`
Retrieve a patcher's locked background color.
- `t_max_err jpatcher_set_bgcolor (t_object *p, t_jrgba *prgba)`
Set a patcher's locked background color.
- `t_max_err jpatcher_get_gridsize (t_object *p, double *gridsizeX, double *gridsizeY)`
Retrieve a patcher's grid size.
- `t_max_err jpatcher_set_gridsize (t_object *p, double gridsizeX, double gridsizeY)`
Set a patcher's grid size.
- `void jpatcher_deleteobj (t_object *p, t_jbox *b)`
Delete an object that is in a patcher.

- `t_object * jpatcher_get_parentpatcher (t_object *p)`
Given a patcher, return its parent patcher.
- `t_object * jpatcher_get_toppatcher (t_object *p)`
Given a patcher, return the top-level patcher for the tree in which it exists.
- `t_max_err jpatcher_get_rect (t_object *p, t_rect *pr)`
Query a patcher to determine its location and size.
- `t_max_err jpatcher_set_rect (t_object *p, t_rect *pr)`
Set a patcher's location and size.
- `t_max_err jpatcher_get_defrect (t_object *p, t_rect *pr)`
Query a patcher to determine the location and dimensions of its window when initially opened.
- `t_max_err jpatcher_set_defrect (t_object *p, t_rect *pr)`
Set a patcher's default location and size.
- `t_symbol * jpatcher_uniqueboxname (t_object *p, t_symbol *classname)`
Generate a unique name for a box in patcher.
- `t_symbol * jpatcher_get_default_fontname (t_object *p)`
Return the name of the default font used for new objects in a patcher.
- `float jpatcher_get_default_fontsize (t_object *p)`
Return the size of the default font used for new objects in a patcher.
- `long jpatcher_get_default_fontface (t_object *p)`
Return the index of the default font face used for new objects in a patcher.
- `long jpatcher_get_fileversion (t_object *p)`
Return the file version of the patcher.
- `long jpatcher_get_currentfileversion (void)`
Return the file version for any new patchers, e.g.

33.42.1 Detailed Description

The patcher.

33.42.2 Function Documentation

33.42.2.1 void jpatcher_deleteobj (t_object * p, t_jbox * b)

Delete an object that is in a patcher.

Parameters

- p* The patcher.
- b* The object box to delete.

33.42.2.2 t_max_err jpatcher_get_bgcolor (t_object *p, t_jrgba *prgba)

Retrieve a patcher's locked background color.

Parameters

- p* The patcher to be queried.
- prgba* The address of a valid `t_jrgba` struct that will be filled-in with the current patcher color values.

Returns

A Max error code.

33.42.2.3 char jpatcher_get_bghidden (t_object *p)

Determine whether a patcher's background layer is hidden.

Parameters

- p* The patcher to be queried.

Returns

True if the background layer is hidden, otherwise false.

33.42.2.4 char jpatcher_get_bglocked (t_object *p)

Determine whether a patcher's background layer is locked.

Parameters

- p* The patcher to be queried.

Returns

True if the background layer is locked, otherwise false.

33.42.2.5 t_object* jpatcher_get_box (t_object *p)

If a patcher is inside a box, return its box.

Parameters

- p* The patcher to be queried.

Returns

A pointer to the box containing the patcher, otherwise NULL.

33.42.2.6 long jpatcher_get_count (t_object *p)

Determine the number of boxes in a patcher.

Parameters

p The patcher to be queried.

Returns

The number of boxes in the patcher.

33.42.2.7 long jpatcher_get_currentfileversion (void)

Return the file version for any new patchers, e.g.
the current version created by Max.

Returns

The file version number.

33.42.2.8 long jpatcher_get_default_fontface (t_object *p)

Return the index of the default font face used for new objects in a patcher.

Parameters

p A pointer to a patcher instance.

Returns

The index of the default font face used for new objects in a patcher.

33.42.2.9 t_symbol* jpatcher_get_default_fontname (t_object *p)

Return the name of the default font used for new objects in a patcher.

Parameters

p A pointer to a patcher instance.

Returns

The name of the default font used for new objects in a patcher.

33.42.2.10 float jpatcher_get_default_fontsize (t_object *p)

Return the size of the default font used for new objects in a patcher.

Parameters

p A pointer to a patcher instance.

Returns

The size of the default font used for new objects in a patcher.

33.42.2.11 t_max_err jpatcher_get_defrect (t_object *p, t_rect *pr)

Query a patcher to determine the location and dimensions of its window when initially opened.

Parameters

p A pointer to a patcher instance.

pr The address of valid [t_rect](#) whose values will be filled-in upon return.

Returns

A Max error code.

33.42.2.12 char jpatcher_get_dirty (t_object *p)

Determine whether a patcher's dirty bit has been set.

Parameters

p The patcher to be queried.

Returns

True if the patcher is dirty, otherwise false.

33.42.2.13 t_max_err jpatcher_get_editing_bgcolor (t_object *p, t_jrgba *prgba)

Retrieve a patcher's editing background color.

Parameters

p The patcher to be queried.

prgba The address of a valid [t_jrgba](#) struct that will be filled-in with the current patcher color values.

Returns

A Max error code.

33.42.2.14 char jpatcher_get_fghidden (t_object *p)

Determine whether a patcher's foreground layer is hidden.

Parameters

p The patcher to be queried.

Returns

True if the foreground layer is hidden, otherwise false.

33.42.2.15 t_symbol* jpatcher_get_filename (t_object *p)

Retrieve a patcher's file name.

Parameters

p The patcher to be queried.

Returns

The patcher's file name.

33.42.2.16 t_symbol* jpatcher_get_filepath (t_object *p)

Retrieve a patcher's file path.

Parameters

p The patcher to be queried.

Returns

The patcher's file path.

33.42.2.17 long jpatcher_get_fileversion (t_object *p)

Return the file version of the patcher.

Parameters

p A pointer to the patcher whose version number is desired.

Returns

The file version number.

33.42.2.18 t_object* jpatcher_get_firstline (t_object *p)

Get the first line (patch-cord) in a patcher.

All lines in a patcher are maintained internally in a [t_linklist](#). Use this function to begin traversing a patcher's lines.

Parameters

p The patcher to be queried.

Returns

The first jpatchline in a patcher.

33.42.2.19 t_object* jpatcher_get_firstobject (t_object *p)

Get the first box in a patcher.

All boxes in a patcher are maintained internally in a [t_linklist](#). Use this function together with [jbox_get_nextobject\(\)](#) to traverse a patcher.

Parameters

p The patcher to be queried.

Returns

The first box in a patcher.

See also

[jbox_get_prevobject\(\)](#) [jbox_get_nextobject\(\)](#) [jpatcher_get_lastobject\(\)](#)

33.42.2.20 t_object* jpatcher_get_firstview (t_object *p)

Get the first view (jpatcherview) for a given patcher.

All views of a patcher are maintained internally as a [t_linklist](#). Use this function to begin traversing a patcher's views.

Parameters

p The patcher to be queried.

Returns

The first view of a patcher.

33.42.2.21 t_max_err jpatcher_get_gridsize (t_object *p, double *gridsizeX, double *gridsizeY)

Retrieve a patcher's grid size.

Parameters

p The patcher to be queried.

gridsizeX The address of a double that will be set to the current horizontal grid spacing for the patcher.

gridsizeY The address of a double that will be set to the current vertical grid spacing for the patcher.

Returns

A Max error code.

33.42.2.22 t_object* jpatcher_get_lastobject (t_object *p)

Get the last box in a patcher.

All boxes in a patcher are maintained internally in a [t_linklist](#). Use this function together with [jbox_get_prevobject\(\)](#) to traverse a patcher.

Parameters

p The patcher to be queried.

Returns

The last box in a patcher.

See also

[jbox_get_prevobject\(\)](#) [jbox_get_nextobject\(\)](#) [jpatcher_get_firstobject\(\)](#)

33.42.2.23 t_symbol* jpatcher_get_name (t_object *p)

Retrieve a patcher's name.

Parameters

p The patcher to be queried.

Returns

The patcher's name.

33.42.2.24 t_object* jpatcher_get_parentpatcher (t_object *p)

Given a patcher, return its parent patcher.

Parameters

p The patcher to be queried.

Returns

The patcher's parent patcher, if there is one. If there is no parent patcher (this is a top-level patcher) then NULL is returned.

33.42.2.25 char jpatcher_get_presentation (t_object *p)

Determine whether a patcher is currently in presentation mode.

Parameters

p The patcher to be queried.

Returns

True if the patcher is in presentation mode, otherwise false.

33.42.2.26 t_max_err jpatcher_get_rect (t_object *p, t_rect *pr)

Query a patcher to determine its location and size.

Parameters

p A pointer to a patcher instance.

pr The address of valid [t_rect](#) whose values will be filled-in upon return.

Returns

A Max error code.

33.42.2.27 t_symbol* jpatcher_get_title (t_object *p)

Retrieve a patcher's title.

Parameters

p The patcher to be queried.

Returns

The patcher's title.

33.42.2.28 t_object* jpatcher_get_toppatcher (t_object *p)

Given a patcher, return the top-level patcher for the tree in which it exists.

Parameters

p The patcher to be queried.

Returns

The patcher's top-level parent patcher.

33.42.2.29 int jpatcher_is_patcher (t_object *p)

Determine if a [t_object](#)* is a patcher object.

Parameters

p The object pointer to test.

Returns

Returns true if the object is a patcher, otherwise returns non-zero.

33.42.2.30 t_max_err jpatcher_set_bgcolor (t_object *p, t_jrgba *prgba)

Set a patcher's locked background color.

Parameters

p The patcher to be queried.

prgba The address of a [t_jrgba](#) struct containing the new color to use.

Returns

A Max error code.

33.42.2.31 t_max_err jpatcher_set_bghidden (t_object *p, char c)

Set whether a patcher's background layer is hidden.

Parameters

p The patcher whose dirty bit will be set.

c Pass true to hide the patcher's background layer, otherwise pass false.

Returns

A Max error code.

33.42.2.32 t_max_err jpatcher_set_bglocked (t_object *p, char c)

Set whether a patcher's background layer is locked.

Parameters

p The patcher whose dirty bit will be set.

c Pass true to lock the patcher's background layer, otherwise pass false.

Returns

A Max error code.

33.42.2.33 t_max_err jpatcher_set_defrect (t_object *p, t_rect *pr)

Set a patcher's default location and size.

Parameters

- p* A pointer to a patcher instance.
- pr* The address of a [t_rect](#) with the new position and size.

Returns

A Max error code.

33.42.2.34 t_max_err jpatcher_set_dirty (t_object *p, char c)

Set a patcher's dirty bit.

Parameters

- p* The patcher whose dirty bit will be set.
- c* The new value for the patcher's dirty bit (pass true or false).

Returns

A Max error code.

33.42.2.35 t_max_err jpatcher_set_editing_bgcolor (t_object *p, t_jrgba *prgba)

Set a patcher's editing background color.

Parameters

- p* The patcher to be queried.
- prgba* The address of a [t_jrgba](#) struct containing the new color to use.

Returns

A Max error code.

33.42.2.36 t_max_err jpatcher_set_fghidden (t_object *p, char c)

Set whether a patcher's foreground layer is hidden.

Parameters

- p* The patcher whose dirty bit will be set.
- c* Pass true to hide the patcher's foreground layer, otherwise pass false.

Returns

A Max error code.

33.42.2.37 t_max_err jpatcher_set_gridsize (t_object *p, double gridsizeX, double gridsizeY)

Set a patcher's grid size.

Parameters

- p* The patcher to be queried.
- gridsizeX* The new horizontal grid spacing for the patcher.
- gridsizeY* The new vertical grid spacing for the patcher.

Returns

A Max error code.

33.42.2.38 t_max_err jpatcher_set_locked (t_object *p, char c)

Lock or unlock a patcher.

Parameters

- p* The patcher whose locked state will be changed.
- c* Pass true to lock a patcher, otherwise pass false.

Returns

A Max error code.

33.42.2.39 t_max_err jpatcher_set_presentation (t_object *p, char c)

Set a patcher to presentation mode.

Parameters

- p* The patcher whose locked state will be changed.
- c* Pass true to switch the patcher to presentation mode, otherwise pass false.

Returns

A Max error code.

33.42.2.40 t_max_err jpatcher_set_rect (t_object *p, t_rect *pr)

Set a patcher's location and size.

Parameters

- p* A pointer to a patcher instance.
- pr* The address of a [t_rect](#) with the new position and size.

Returns

A Max error code.

33.42.2.41 t_max_err jpatcher_set_title (t_object **p*, t_symbol **ps*)

Set a patcher's title.

Parameters

- p* The patcher whose locked state will be changed.
- ps* The new title for the patcher.

Returns

A Max error code.

33.42.2.42 t_symbol* jpatcher_uniqueboxname (t_object **p*, t_symbol **classname*)

Generate a unique name for a box in patcher.

Parameters

- p* A pointer to a patcher instance.
- classname* The name of an object's class.

Returns

The newly-generated unique name.

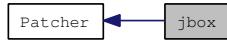
Remarks

This is the function used by pattr to assign names to objects in a patcher.

33.43 jbox

A box in the patcher.

Collaboration diagram for jbox:



Data Structures

- struct [t_jboxdrawparams](#)
The [t_jboxdrawparams](#) structure.

Defines

- #define [JBOX_DRAWFIRSTIN](#) (1<<0)
draw first inlet
- #define [JBOX_NODRAWBOX](#) (1<<1)
don't draw the frame
- #define [JBOX_DRAWINLAST](#) (1<<2)
draw inlets after update method
- #define [JBOX_TRANSPARENT](#) (1<<3)
don't make transparent unless you need it (for efficiency)
- #define [JBOX_NOGROW](#) (1<<4)
don't even draw grow thingie
- #define [JBOX_GROWY](#) (1<<5)
can grow in y direction by dragging
- #define [JBOX_GROWBOTH](#) (1<<6)
can grow independently in both x and y
- #define [JBOX_IGNORELOCKCLICK](#) (1<<7)
box should ignore a click if patcher is locked
- #define [JBOX_HILITE](#) (1<<8)
flag passed to [jbox_new\(\)](#) to tell max that the UI object can receive the focus when clicked on -- may be replaced by JBOX_FOCUS in the future
- #define [JBOX_BACKGROUND](#) (1<<9)
immediately set box into the background
- #define [JBOX_NOFLOATINSPECTOR](#) (1<<10)

no floating inspector window

- #define **JBOX_TEXTFIELD** (1<<11)
save/load text from textfield, unless JBOX_BINBUF flag is set
- #define **JBOX_FIXWIDTH** (1<<19)
give the box a textfield based fix-width (bfixwidth) method
- #define **JBOX_FONTATTR** (1<<18)
if you want font related attribute you must add this to jbox_initclass()
- #define **JBOX_BINBUF** (1<<14)
save/load text from b_binbuf
- #define **JBOX_MOUSEDRAGDELTA** (1<<12)
hides mouse cursor in drag and sends mousedragdelta instead of mousdrag (for infinite scrolling like number)
- #define **JBOX_COLOR** (1<<13)
support the "color" method for color customization
- #define **JBOX_DRAWIOLOCKED** (1<<15)
draw inlets and outlets when locked (default is not to draw them)
- #define **JBOX_DRAWBACKGROUND** (1<<16)
set to have box bg filled in for you based on getdrawparams method or brgba attribute
- #define **JBOX_NOINSPECTFIRSTIN** (1<<17)
flag for objects such as bpatcher that have a different b_firstin,
- #define **JBOX_DEFAULTNAMES** (1<<18)
flag instructing jbox_new to attach object to the defaults object for live defaults updating
- #define **JBOX_FOCUS** (1<<20)
more advanced focus support (passed to jbox_initclass() to add "nextfocus" and "prevfocus" attributes to the UI object). Not implemented as of 2009-05-11

Enumerations

- enum {
JBOX_FONTFACE_REGULAR = 0,
JBOX_FONTFACE_BOLD = 1,
JBOX_FONTFACE_ITALIC = 2,
JBOX_FONTFACE_BOLDITALIC = 3 }
actual numerical values of the b_fontface attribute; use jbox_fontface() to weight

- enum HitTestResult {

 HitNothing = 0,

 HitBox = 1,

 HitInlet = 2,

 HitOutlet = 3,

 HitGrowBox = 4,

 HitLine = 5 }

enumerations used for box decorators

Functions

- **t_max_err jbox_get_rect_for_view** (**t_object** *box, **t_object** *patcherview, **t_rect** *rect)
Find the rect for a box in a given patcherview.
- **t_max_err jbox_set_rect_for_view** (**t_object** *box, **t_object** *patcherview, **t_rect** *rect)
Change the rect for a box in a given patcherview.
- **t_max_err jbox_get_rect_for_sym** (**t_object** *box, **t_symbol** *which, **t_rect** *pr)
Find the rect for a box with a given attribute name.
- **t_max_err jbox_set_rect_for_sym** (**t_object** *box, **t_symbol** *which, **t_rect** *pr)
Change the rect for a box with a given attribute name.
- **t_max_err jbox_set_rect** (**t_object** *box, **t_rect** *pr)
Set both the presentation rect and the patching rect.
- **t_max_err jbox_get_patching_rect** (**t_object** *box, **t_rect** *pr)
Retrieve the patching rect of a box.
- **t_max_err jbox_set_patching_rect** (**t_object** *box, **t_rect** *pr)
Change the patching rect of a box.
- **t_max_err jbox_get_presentation_rect** (**t_object** *box, **t_rect** *pr)
Retrieve the presentation rect of a box.
- **t_max_err jbox_set_presentation_rect** (**t_object** *box, **t_rect** *pr)
Change the presentation rect of a box.
- **t_max_err jbox_set_position** (**t_object** *box, **t_pt** *pos)
Set the position of a box for both the presentation and patching views.
- **t_max_err jbox_get_patching_position** (**t_object** *box, **t_pt** *pos)
Fetch the position of a box for the patching view.
- **t_max_err jbox_set_patching_position** (**t_object** *box, **t_pt** *pos)
Set the position of a box for the patching view.

- `t_max_err jbox_get_presentation_position (t_object *box, t_pt *pos)`
Fetch the position of a box for the presentation view.
- `t_max_err jbox_set_presentation_position (t_object *box, t_pt *pos)`
Set the position of a box for the presentation view.
- `t_max_err jbox_set_size (t_object *box, t_size *size)`
Set the size of a box for both the presentation and patching views.
- `t_max_err jbox_get_patching_size (t_object *box, t_size *size)`
Fetch the size of a box for the patching view.
- `t_max_err jbox_set_patching_size (t_object *box, t_size *size)`
Set the size of a box for the patching view.
- `t_max_err jbox_get_presentation_size (t_object *box, t_size *size)`
Fetch the size of a box for the presentation view.
- `t_max_err jbox_set_presentation_size (t_object *box, t_size *size)`
Set the size of a box for the presentation view.
- `t_symbol * jbox_get_maxclass (t_object *b)`
Retrieve the name of the class of the box's object.
- `t_object * jbox_get_object (t_object *b)`
Retrieve a pointer to the box's object.
- `t_object * jbox_get_patcher (t_object *b)`
Retrieve a box's patcher.
- `char jbox_get_hidden (t_object *b)`
Retrieve a box's 'hidden' attribute.
- `t_max_err jbox_set_hidden (t_object *b, char c)`
Set a box's 'hidden' attribute.
- `t_symbol * jbox_get_fontname (t_object *b)`
Retrieve a box's 'fontname' attribute.
- `t_max_err jbox_set_fontname (t_object *b, t_symbol *ps)`
Set a box's 'fontname' attribute.
- `double jbox_get_fontsize (t_object *b)`
Retrieve a box's 'fontsize' attribute.
- `t_max_err jbox_set_fontsize (t_object *b, double d)`
Set a box's 'fontsize' attribute.
- `t_max_err jbox_get_color (t_object *b, t_jrgba *prgba)`
Retrieve a box's 'color' attribute.

- **t_max_err jbox_set_color (t_object *b, t_jrgba *prgba)**
Set a box's 'color' attribute.
- **t_symbol * jbox_get_hint (t_object *b)**
Retrieve a box's hint text as a symbol.
- **t_max_err jbox_set_hint (t_object *b, t_symbol *s)**
Set a box's hint text using a symbol.
- **char * jbox_get_hintstring (t_object *bb)**
Retrieve a box's hint text as a C-string.
- **void jbox_set_hintstring (t_object *bb, char *s)**
Set a box's hint text using a C-string.
- **char * jbox_get_annotation (t_object *bb)**
Retrieve a box's annotation string, if the user has given it an annotation.
- **void jbox_set_annotation (t_object *bb, char *s)**
Set a box's annotation string.
- **t_object * jbox_get_nextobject (t_object *b)**
The next box in the patcher's (linked) list of boxes.
- **t_object * jbox_get_prevobject (t_object *b)**
The previous box in the patcher's (linked) list of boxes.
- **t_symbol * jbox_get_varname (t_object *b)**
Retrieve a box's scripting name.
- **t_max_err jbox_set_varname (t_object *b, t_symbol *ps)**
Set a box's scripting name.
- **t_symbol * jbox_get_id (t_object *b)**
Retrieve a box's unique id.
- **char jbox_get_canhilite (t_object *b)**
Retrieve a box flag value from a box.
- **char jbox_get_background (t_object *b)**
Determine whether a box is located in the patcher's background layer.
- **t_max_err jbox_set_background (t_object *b, char c)**
Set whether a box should be in the background or foreground layer of a patcher.
- **char jbox_get_ignoreclick (t_object *b)**
Determine whether a box ignores clicks.
- **t_max_err jbox_set_ignoreclick (t_object *b, char c)**

Set whether a box ignores clicks.

- `char jbox_get_drawfirstin (t_object *b)`
Determine whether a box draws its first inlet.
- `char jbox_get_outline (t_object *b)`
Determine whether a box draws an outline.
- `t_max_err jbox_set_outline (t_object *b, char c)`
Set whether a box draws an outline.
- `char jbox_get_growy (t_object *b)`
Retrieve a box flag value from a box.
- `char jbox_get_growboth (t_object *b)`
Retrieve a box flag value from a box.
- `char jbox_get_nogrow (t_object *b)`
Retrieve a box flag value from a box.
- `char jbox_get_drawinlast (t_object *b)`
Retrieve a box flag value from a box.
- `t_object * jbox_get_textfield (t_object *b)`
Retrieve a pointer to a box's textfield.
- `char jbox_get_presentation (t_object *b)`
Determine if a box is included in the presentation view.
- `t_max_err jbox_set_presentation (t_object *b, char c)`
Determine if a box is included in the presentation view.
- `t_max_err jbox_new (t_jbox *b, long flags, long argc, t_atom *argv)`
Set up your UI object's `t_jbox` member.
- `void jbox_free (t_jbox *b)`
Tear down your UI object's `t_jbox` member.
- `void jbox_ready (t_jbox *b)`
Mark the box ready to be accessed and drawn by Max.
- `void jbox_redraw (t_jbox *b)`
Request that your object/box be re-drawn by Max.
- `t_max_err jbox_notify (t_jbox *b, t_symbol *s, t_symbol *msg, void *sender, void *data)`
Send a notification to a box.

33.43.1 Detailed Description

A box in the patcher.

33.43.2 Define Documentation

33.43.2.1 `#define JBOX_NOINSPECTFIRSTIN (1<<17)`

flag for objects such as bpatcher that have a different b_firstin,
but the attrs of the b_firstin should not be shown in the inspector

33.43.3 Enumeration Type Documentation

33.43.3.1 anonymous enum

actual numerical values of the b_fontface attribute; use jbox_fontface() to weight

Enumerator:

`JBOX_FONTFACE_REGULAR` normal

`JBOX_FONTFACE_BOLD` bold

`JBOX_FONTFACE_ITALIC` italic

`JBOX_FONTFACE_BOLDITALIC` bold and italic

33.43.3.2 enum HitTestResult

enumerations used for box decorators

Enumerator:

`HitNothing` a hole in the box

`HitBox` the body of the box

`HitInlet` an inlet

`HitOutlet` an outlet

`HitGrowBox` the grow handle

`HitLine` a line

33.43.4 Function Documentation

33.43.4.1 `void jbox_free (t_jbox * b)`

Tear down your UI object's `t_jbox` member.

This should be called from your UI object's free method.

Parameters

`b` The address of your object's `t_jbox` member (which should be the first member of the object's struct).

33.43.4.2 char* jbox_get_annotation (*t_object* * *bb*)

Retrieve a box's annotation string, if the user has given it an annotation.

Parameters

bb The box to query.

Returns

The user-created annotation string for a box, or NULL if no string exists.

33.43.4.3 char jbox_get_background (*t_object* * *b*)

Determine whether a box is located in the patcher's background layer.

Parameters

b The box to query.

Returns

Zero if the object is in the foreground, otherwise non-zero.

33.43.4.4 char jbox_get_canhilite (*t_object* * *b*)

Retrieve a box flag value from a box.

Parameters

b The box to query.

Returns

The value of the canhilite bit in the box's flags.

33.43.4.5 t_max_err jbox_get_color (*t_object* * *b*, *t_jrgba* * *prgba*)

Retrieve a box's 'color' attribute.

Parameters

b The box to query.

prgba The address of a valid *t_rect* whose values will be filled-in upon return.

Returns

A Max error code.

33.43.4.6 char jbox_get_drawfirstin (t_object * *b*)

Determine whether a box draws its first inlet.

Parameters

b The box to query.

Returns

Zero if the inlet is not drawn, otherwise non-zero.

33.43.4.7 char jbox_get_drawinlast (t_object * *b*)

Retrieve a box flag value from a box.

Parameters

b The box to query.

Returns

The value of the drawinlast bit in the box's flags.

33.43.4.8 t_symbol* jbox_get_fontname (t_object * *b*)

Retrieve a box's 'fontname' attribute.

Parameters

b The box to query.

Returns

The font name.

33.43.4.9 double jbox_get_fontsize (t_object * *b*)

Retrieve a box's 'fontsize' attribute.

Parameters

b The box to query.

Returns

The font size in points.

33.43.4.10 char jbox_get_growboth (t_object * *b*)

Retrieve a box flag value from a box.

Parameters

b The box to query.

Returns

The value of the growboth bit in the box's flags.

33.43.4.11 char jbox_get_growy (t_object * *b*)

Retrieve a box flag value from a box.

Parameters

b The box to query.

Returns

The value of the growy bit in the box's flags.

33.43.4.12 char jbox_get_hidden (t_object * *b*)

Retrieve a box's 'hidden' attribute.

Parameters

b The box to query.

Returns

True if the box is hidden, otherwise false.

33.43.4.13 t_symbol* jbox_get_hint (t_object * *b*)

Retrieve a box's hint text as a symbol.

Parameters

b The box to query.

Returns

The box's hint text.

33.43.4.14 `char* jbox_get_hintstring (t_object * bb)`

Retrieve a box's hint text as a C-string.

Parameters

bb The box to query.

Returns

The box's hint text.

33.43.4.15 `t_symbol* jbox_get_id (t_object * b)`

Retrieve a boxes unique id.

Parameters

b The box to query.

Returns

The unique id of the object. This is a symbol that is referenced, for example, by patchlines.

33.43.4.16 `char jbox_get_ignoreclick (t_object * b)`

Determine whether a box ignores clicks.

Parameters

b The box to query.

Returns

Zero if the object responds to clicks, otherwise non-zero.

33.43.4.17 `t_symbol* jbox_get_maxclass (t_object * b)`

Retrieve the name of the class of the box's object.

Parameters

b The box to query.

Returns

The name of the class of the box's object.

33.43.4.18 t_object* jbox_get_nextobject (t_object * *b*)

The next box in the patcher's (linked) list of boxes.

Parameters

b The box to query.

Returns

The next box in the list.

33.43.4.19 char jbox_get_nogrow (t_object * *b*)

Retrieve a box flag value from a box.

Parameters

b The box to query.

Returns

The value of the nogrow bit in the box's flags.

33.43.4.20 t_object* jbox_get_object (t_object * *b*)

Retrieve a pointer to the box's object.

Parameters

b The box to query.

Returns

A pointer to the box's object.

33.43.4.21 char jbox_get_outline (t_object * *b*)

Determine whether a box draws an outline.

Parameters

b The box to query.

Returns

Zero if the outline is not drawn, otherwise non-zero.

33.43.4.22 `t_object* jbox_get_patcher (t_object *b)`

Retrieve a box's patcher.

Parameters

b The box to query.

Returns

If the box has a patcher, the patcher's pointer is returned. Otherwise NULL is returned.

33.43.4.23 `t_max_err jbox_get_patching_position (t_object *box, t_pt *pos)`

Fetch the position of a box for the patching view.

Parameters

box The box whose position will be retrieved.

pos The address of a valid `t_pt` whose x and y values will be filled in.

Returns

A Max error code.

33.43.4.24 `t_max_err jbox_get_patching_rect (t_object *box, t_rect *pr)`

Retrieve the patching rect of a box.

Parameters

box The box whose rect values will be retrieved.

pr The address of a valid `t_rect` whose values will be filled in.

Returns

A Max error code.

33.43.4.25 `t_max_err jbox_get_patching_size (t_object *box, t_size *size)`

Fetch the size of a box for the patching view.

Parameters

box The box whose size will be retrieved.

size The address of a valid `t_size` whose width and height values will be filled in.

Returns

A Max error code.

33.43.4.26 char jbox_get_presentation (t_object * *b*)

Determine if a box is included in the presentation view.

Parameters

b The box to query.

Returns

Non-zero if in presentation mode, otherwise zero.

33.43.4.27 t_max_err jbox_get_presentation_position (t_object * *box*, t_pt * *pos*)

Fetch the position of a box for the presentation view.

Parameters

box The box whose position will be retrieved.

pos The address of a valid [t_pt](#) whose x and y values will be filled in.

Returns

A Max error code.

33.43.4.28 t_max_err jbox_get_presentation_rect (t_object * *box*, t_rect * *pr*)

Retrieve the presentation rect of a box.

Parameters

box The box whose rect values will be retrieved.

pr The address of a valid [t_rect](#) whose values will be filled in.

Returns

A Max error code.

33.43.4.29 t_max_err jbox_get_presentation_size (t_object * *box*, t_size * *size*)

Fetch the size of a box for the presentation view.

Parameters

box The box whose size will be retrieved.

size The address of a valid [t_size](#) whose width and height values will be filled in.

Returns

A Max error code.

33.43.4.30 t_object* jbox_get_prevobject (t_object * *b*)

The previous box in the patcher's (linked) list of boxes.

Parameters

b The box to query.

Returns

The next box in the list.

33.43.4.31 t_max_err jbox_get_rect_for_sym (t_object * *box*, t_symbol * *which*, t_rect * *pr*)

Find the rect for a box with a given attribute name.

Parameters

box The box whose rect will be fetched.

which The name of the rect attribute to be fetched, for example `_sym_presentation_rect` or `_sym_patch_rect`.

pr The address of a valid `t_rect` whose members will be filled in by this function.

Returns

A Max error code.

33.43.4.32 t_max_err jbox_get_rect_for_view (t_object * *box*, t_object * *patcherview*, t_rect * *rect*)

Find the rect for a box in a given patcherview.

Parameters

box The box whose rect will be fetched.

patcherview A patcherview in which the box exists.

rect The address of a valid `t_rect` whose members will be filled in by this function.

Returns

A Max error code.

33.43.4.33 t_object* jbox_get_textfield (t_object * *b*)

Retrieve a pointer to a box's textfield.

Parameters

b The box to query.

Returns

The textfield for the box, assuming it has one. If the box does not own a textfield then NULL is returned.

33.43.4.34 t_symbol* jbox_get_varname (t_object * b)

Retrieve a box's scripting name.

Parameters

b The box to query.

Returns

The box's scripting name.

33.43.4.35 t_max_err jbox_new (t_jbox * b, long flags, long argc, t_atom * argv)

Set up your UI object's [t_jbox](#) member.

This should be called from your UI object's free method.

Parameters

b The address of your UI object's [t_jbox](#) member (which should be the first member of the object's struct).

flags Flags to set the box's behavior, such as [JBOX_NODRAWBOX](#).

argc The count of atoms in the argv parameter.

argv The address of the first in an array of atoms to be passed to the box constructor. Typically these are simply the argument passed to your object when it is created.

Returns

A Max error code.

33.43.4.36 t_max_err jbox_notify (t_jbox * b, t_symbol * s, t_symbol * msg, void * sender, void * data)

Send a notification to a box.

This is the same as calling [object_notify\(\)](#) for a box.

Parameters

b The address of your object's [t_jbox](#) member.

s The name of the send object.

msg The notification name.

sender The sending object's address.

data A pointer to some data passed to the box's notify method.

Returns

A Max error code.

33.43.4.37 void jbox_ready (*t_jbox* **b*)

Mark the box ready to be accessed and drawn by Max.

This should typically be called at the end of your UI object's new method.

Parameters

b The address of your object's *t_jbox* member.

33.43.4.38 void jbox_redraw (*t_jbox* **b*)

Request that your object/box be re-drawn by Max.

Parameters

b The address of your object's *t_jbox* member.

33.43.4.39 void jbox_set_annotation (*t_object* **bb*, *char* **s*)

Set a box's annotation string.

Parameters

bb The box to query.

s The annotation string for the box.

Returns

A Max error code.

33.43.4.40 *t_max_err* jbox_set_background (*t_object* **b*, *char* *c*)

Set whether a box should be in the background or foreground layer of a patcher.

Parameters

b The box to query.

c Pass zero to tell the box to appear in the foreground, or non-zero to indicate that the box should be in the background layer.

Returns

A Max error code.

33.43.4.41 *t_max_err* jbox_set_color (*t_object* **b*, *t_jrgba* **prgba*)

Set a box's 'color' attribute.

Parameters

b The box to query.

prgba The address of a `t_rect` containing the desired color for the box/object.

Returns

A Max error code.

33.43.4.42 `t_max_err jbox_set_fontname (t_object *b, t_symbol *ps)`

Set a box's 'fontname' attribute.

Parameters

b The box to query.

ps The font name. Note that the font name may be case-sensitive.

Returns

A Max error code.

33.43.4.43 `t_max_err jbox_set_fontsize (t_object *b, double d)`

Set a box's 'fontsize' attribute.

Parameters

b The box to query.

d The fontsize in points.

Returns

A Max error code.

33.43.4.44 `t_max_err jbox_set_hidden (t_object *b, char c)`

Set a box's 'hidden' attribute.

Parameters

b The box to query.

c Set to true to hide the box, otherwise false.

Returns

A Max error code.

33.43.4.45 `t_max_err jbox_set_hint (t_object *b, t_symbol *s)`

Set a box's hint text using a symbol.

Parameters

b The box to query.

s The new text to use for the box's hint.

Returns

A Max error code.

33.43.4.46 void jbox_set_hintstring (*t_object* * *bb*, *char* * *s*)

Set a box's hint text using a C-string.

Parameters

bb The box to query.

s The new text to use for the box's hint.

Returns

A Max error code.

33.43.4.47 *t_max_err* jbox_set_ignoreclick (*t_object* * *b*, *char* *c*)

Set whether a box ignores clicks.

Parameters

b The box to query.

c Pass zero to tell the box to respond to clicks, or non-zero to indicate that the box should ignore clicks.

Returns

A Max error code.

33.43.4.48 *t_max_err* jbox_set_outline (*t_object* * *b*, *char* *c*)

Set whether a box draws an outline.

Parameters

b The box to query.

c Pass zero to hide the outline, or non-zero to indicate that the box should draw the outline.

Returns

A Max error code.

33.43.4.49 t_max_err jbox_set_patching_position (t_object * *box*, t_pt * *pos*)

Set the position of a box for the patching view.

Parameters

box The box whose positon will be changed.

pos The address of a [t_pt](#) with the new x and y values.

Returns

A Max error code.

33.43.4.50 t_max_err jbox_set_patching_rect (t_object * *box*, t_rect * *pr*)

Change the patching rect of a box.

Parameters

box The box whose rect will be changed.

pr The address of a [t_rect](#) with the new rect values.

Returns

A Max error code.

33.43.4.51 t_max_err jbox_set_patching_size (t_object * *box*, t_size * *size*)

Set the size of a box for the patching view.

Parameters

box The box whose size will be changed.

size The address of a [t_size](#) with the new width and height values.

Returns

A Max error code.

33.43.4.52 t_max_err jbox_set_position (t_object * *box*, t_pt * *pos*)

Set the position of a box for both the presentation and patching views.

Parameters

box The box whose position will be changed.

pos The address of a [t_pt](#) with the new x and y values.

Returns

A Max error code.

33.43.4.53 t_max_err jbox_set_presentation (t_object * *b*, char *c*)

Determine if a box is included in the presentation view.

Parameters

- b* The box to query.
- c* Pass zero to remove a box from the presentation view, or non-zero to add it to the presentation view.

Returns

Non-zero if in presentation mode, otherwise zero.

33.43.4.54 t_max_err jbox_set_presentation_position (t_object * *box*, t_pt * *pos*)

Set the position of a box for the presentation view.

Parameters

- box* The box whose rect will be changed.
- pos* The address of a [t_pt](#) with the new x and y values.

Returns

A Max error code.

33.43.4.55 t_max_err jbox_set_presentation_rect (t_object * *box*, t_rect * *pr*)

Change the presentation rect of a box.

Parameters

- box* The box whose rect will be changed.
- pr* The address of a [t_rect](#) with the new rect values.

Returns

A Max error code.

33.43.4.56 t_max_err jbox_set_presentation_size (t_object * *box*, t_size * *size*)

Set the size of a box for the presentation view.

Parameters

- box* The box whose size will be changed.
- size* The address of a [t_size](#) with the new width and height values.

Returns

A Max error code.

33.43.4.57 `t_max_err jbox_set_rect (t_object * box, t_rect * pr)`

Set both the presentation rect and the patching rect.

Parameters

box The box whose rect will be changed.

pr The address of a `t_rect` with the new rect values.

Returns

A Max error code.

33.43.4.58 `t_max_err jbox_set_rect_for_sym (t_object * box, t_symbol * which, t_rect * pr)`

Change the rect for a box with a given attribute name.

Parameters

box The box whose rect will be changed.

which The name of the rect attribute to be changed, for example `_sym_presentation_rect` or `_sym_patchrect`.

pr The address of a valid `t_rect` that will replace the current values used by the box.

Returns

A Max error code.

33.43.4.59 `t_max_err jbox_set_rect_for_view (t_object * box, t_object * patcherview, t_rect * rect)`

Change the rect for a box in a given patcherview.

Parameters

box The box whose rect will be changed.

patcherview A patcherview in which the box exists.

rect The address of a valid `t_rect` that will replace the current values used by the box in the given view.

Returns

A Max error code.

33.43.4.60 `t_max_err jbox_set_size (t_object * box, t_size * size)`

Set the size of a box for both the presentation and patching views.

Parameters

box The box whose size will be changed.

size The address of a `t_size` with the new size values.

Returns

A Max error code.

33.43.4.61 t_max_err jbox_set_varname (t_object * *b*, t_symbol * *ps*)

Set a box's scripting name.

Parameters

b The box to query.

ps The new scripting name for the box.

Returns

A Max error code.

33.44 jpatchline

A patch cord.

Collaboration diagram for jpatchline:



Functions

- `t_max_err jpatchline_get_startpoint (t_object *l, double *x, double *y)`
Retrieve a patchline's starting point.
- `t_max_err jpatchline_get_endpoint (t_object *l, double *x, double *y)`
Retrieve a patchline's ending point.
- `long jpatchline_get_nummidpoints (t_object *l)`
Determine the number of midpoints (segments) in a patchline.
- `t_object * jpatchline_get_box1 (t_object *l)`
Return the object box from which a patchline originates.
- `long jpatchline_get_outletnum (t_object *l)`
Return the outlet number of the originating object box from which a patchline begins.
- `t_object * jpatchline_get_box2 (t_object *l)`
Return the destination object box for a patchline.
- `long jpatchline_get_inletnum (t_object *l)`
Return the inlet number of the destination object box to which a patchline is connected.
- `t_object * jpatchline_get_nextline (t_object *b)`
Given a patchline, traverse to the next patchline in the (linked) list.
- `char jpatchline_get_hidden (t_object *l)`
Determine if a patch line is hidden.
- `t_max_err jpatchline_set_hidden (t_object *l, char c)`
Set a patchline's visibility.
- `t_max_err jpatchline_get_color (t_object *l, t_jrgba *prgba)`
Get the color of a patch line.
- `t_max_err jpatchline_set_color (t_object *l, t_jrgba *prgba)`
Set the color of a patch line.

33.44.1 Detailed Description

A patch cord.

33.44.2 Function Documentation

33.44.2.1 `t_object* jpatchline_get_box1 (t_object *l)`

Return the object box from which a patchline originates.

Parameters

l A pointer to the patchline's instance.

Returns

The object box from which the patchline originates.

33.44.2.2 `t_object* jpatchline_get_box2 (t_object *l)`

Return the destination object box for a patchline.

Parameters

l A pointer to the patchline's instance.

Returns

The destination object box for a patchline.

33.44.2.3 `t_max_err jpatchline_get_color (t_object *l, t_jrgba *prgba)`

Get the color of a patch line.

Parameters

l A patchline instance.

prgba The address of a valid `t_jrgba` struct that will be filled with the color values of the patch line.

Returns

An error code.

33.44.2.4 `t_max_err jpatchline_get_endpoint (t_object *l, double *x, double *y)`

Retrieve a patchline's ending point.

Parameters

l A pointer to the patchline's instance.

- x* The address of a variable to hold the x-coordinate of the ending point's position upon return.
- y* The address of a variable to hold the y-coordinate of the ending point's position upon return.

Returns

A Max error code.

33.44.2.5 char jpatchline_get_hidden (*t_object* **l*)

Determine if a patchline is hidden.

Parameters

l A patchline instance.

Returns

Zero if the patchline is visible, non-zero if it is hidden.

33.44.2.6 long jpatchline_get_inletnum (*t_object* **l*)

Return the inlet number of the destination object box to which a patchline is connected.

Parameters

l A pointer to the patchline's instance.

Returns

The inlet number.

33.44.2.7 *t_object jpatchline_get_nextline (*t_object* **b*)**

Given a patchline, traverse to the next patchline in the (linked) list.

Parameters

b A patchline instance.

Returns

The next patchline. If the current patchline is at the end (tail) of the list, then NULL is returned.

33.44.2.8 long jpatchline_get_nummidpoints (*t_object* **l*)

Determine the number of midpoints (segments) in a patchline.

Parameters

l A pointer to the patchline's instance.

Returns

The number of midpoints in the patchline.

33.44.2.9 long jpatchline_get_outletnum (*t_object* **l*)

Return the outlet number of the originating object box from which a patchline begins.

Parameters

l A pointer to the patchline's instance.

Returns

The outlet number.

33.44.2.10 t_max_err jpatchline_get_startpoint (*t_object* **l*, *double* **x*, *double* **y*)

Retrieve a patchline's starting point.

Parameters

l A pointer to the patchline's instance.

x The address of a variable to hold the x-coordinate of the starting point's position upon return.

y The address of a variable to hold the y-coordinate of the starting point's position upon return.

Returns

A Max error code.

33.44.2.11 t_max_err jpatchline_set_color (*t_object* **l*, *t_jrgba* **prgba*)

Set the color of a patch line.

Parameters

l A patchline instance.

prgba The address of a valid *t_jrgba* struct containing the color to use.

Returns

An error code.

33.44.2.12 t_max_err jpatchline_set_hidden (*t_object* **l*, *char* *c*)

Set a patchline's visibility.

Parameters

l A patchline instance.

c Pass 0 to make a patchline visible, or non-zero to hide it.

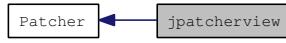
Returns

An error code.

33.45 jpatcherview

A view of a patcher.

Collaboration diagram for jpatcherview:



Functions

- `t_object * patcherview_findpatcherview (int x, int y)`
Find a patcherview at the given screen coords.
- `char patcherview_get_visible (t_object *pv)`
Query a patcherview to determine whether it is visible.
- `t_max_err patcherview_set_visible (t_object *pv, char c)`
Set the 'visible' attribute of a patcherview.
- `t_max_err patcherview_get_rect (t_object *pv, t_rect *pr)`
Get the value of the rect attribute for a patcherview.
- `t_max_err patcherview_set_rect (t_object *pv, t_rect *pr)`
Set the value of the rect attribute for a patcherview.
- `char patcherview_get_locked (t_object *p)`
Find out if a patcherview is locked.
- `t_max_err patcherview_set_locked (t_object *p, char c)`
Lock or unlock a patcherview.
- `char patcherview_get_presentation (t_object *pv)`
Find out if a patcherview is a presentation view.
- `t_max_err patcherview_set_presentation (t_object *p, char c)`
Set whether or not a patcherview is a presentation view.
- `double patcherview_get_zoomfactor (t_object *pv)`
Fetch the zoom-factor of a patcherview.
- `t_max_err patcherview_set_zoomfactor (t_object *pv, double d)`
Set the zoom-factor of a patcherview.
- `t_object * patcherview_get_nextview (t_object *pv)`
Given a patcherview, find the next patcherview.
- `t_object * patcherview_get_jgraphics (t_object *pv)`
Given a patcherview, return the `t_jgraphics` context for that view.

- `t_object * patcherview_get_patcher (t_object *pv)`

Given a patcherview, return its patcher.

- `t_object * patcherview_get_topview (t_object *pv)`

Given a patcherview, return the top patcherview (possibly itself).

33.45.1 Detailed Description

A view of a patcher.

33.45.2 Function Documentation

33.45.2.1 `t_object* patcherview_findpatcherview (int x, int y)`

Find a patcherview at the given screen coords.

Parameters

- x* The horizontal coordinate at which to find a patcherview.
- y* The vertical coordinate at which to find a patcherview.

Returns

A pointer to the patcherview at the specified location, or NULL if no patcherview exists at that location.

33.45.2.2 `t_object* patcherview_get_jgraphics (t_object *pv)`

Given a patcherview, return the `t_jgraphics` context for that view.

Parameters

- pv* The patcherview instance.

Returns

The `t_jgraphics` context for the view.

33.45.2.3 `char patcherview_get_locked (t_object *p)`

Find out if a patcherview is locked.

Parameters

- p* The patcherview instance whose attribute value will be fetched.

Returns

Returns 0 if unlocked, otherwise returns non-zero.

33.45.2.4 `t_object* patcherview_get_nextview (t_object *pv)`

Given a patcherview, find the next patcherview.

The views of a patcher are maintained internally as a [t_linklist](#), and so the views can be traversed should you need to perform operations on all of a patcher's patcherviews.

Parameters

pv The patcherview instance from which to find the next patcherview.

Returns

The next patcherview in the list, or NULL if the patcherview passed in *pv* is the tail.

33.45.2.5 `t_object* patcherview_get_patcher (t_object *pv)`

Given a patcherview, return its patcher.

Parameters

pv The patcherview instance for which to fetch the patcher.

Returns

The patcher.

33.45.2.6 `char patcherview_get_presentation (t_object *pv)`

Find out if a patcherview is a presentation view.

Parameters

pv The patcherview instance whose attribute value will be fetched.

Returns

Returns 0 if the view is not a presentation view, otherwise returns non-zero.

33.45.2.7 `t_max_err patcherview_get_rect (t_object *pv, t_rect *pr)`

Get the value of the rect attribute for a patcherview.

Parameters

pv The patcherview instance whose attribute value will be fetched.

pr The address of a valid [t_rect](#) struct, whose contents will be filled upon return.

Returns

An error code.

33.45.2.8 t_object* patcherview_get_topview (t_object *pv)

Given a patcherview, return the top patcherview (possibly itself).

If the patcherview is inside a bpatcher which is in a patcher then this will give you the view the bpatcher view is inside of.

Parameters

pv The patcherview instance whose top view you want to get.

Returns

The top patcherview.

33.45.2.9 char patcherview_get_visible (t_object *pv)

Query a patcherview to determine whether it is visible.

Parameters

pv The patcherview instance to query.

Returns

Returns zero if the patcherview is invisible, otherwise returns non-zero.

33.45.2.10 double patcherview_get_zoomfactor (t_object *pv)

Fetch the zoom-factor of a patcherview.

Parameters

pv The patcherview instance whose attribute value will be fetched.

Returns

The factor by which the view is zoomed.

33.45.2.11 t_max_err patcherview_set_locked (t_object *p, char c)

Lock or unlock a patcherview.

Parameters

p The patcherview instance whose attribute value will be set.

c Set this value to zero to unlock the patcherview, otherwise pass a non-zero value.

Returns

An error code.

33.45.2.12 t_max_err patcherview_set_presentation (t_object **p*, char *c*)

Set whether or not a patcherview is a presentation view.

Parameters

- p* The patcherview instance whose attribute value will be set.
- c* Set this value to non-zero to make the patcherview a presentation view, otherwise pass zero.

Returns

An error code.

33.45.2.13 t_max_err patcherview_set_rect (t_object **pv*, t_rect **pr*)

Set the value of the rect attribute for a patcherview.

Parameters

- pv* The patcherview instance whose attribute value will be set.
- pr* The address of a valid [t_rect](#) struct.

Returns

An error code.

33.45.2.14 t_max_err patcherview_set_visible (t_object **pv*, char *c*)

Set the 'visible' attribute of a patcherview.

Parameters

- pv* The patcherview instance whose attribute value will be set.
- c* Whether or not the patcherview should be made visible.

Returns

An error code.

33.45.2.15 t_max_err patcherview_set_zoomfactor (t_object **pv*, double *d*)

Set the zoom-factor of a patcherview.

Parameters

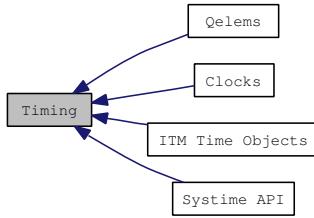
- pv* The patcherview instance whose attribute value will be set.
- d* The zoom-factor at which the patcherview should display the patcher.

Returns

An error code.

33.46 Timing

Collaboration diagram for Timing:



Modules

- [Clocks](#)

Clock objects are your interface to Max's scheduler.

- [Qelems](#)

Your object's methods may be called at interrupt level.

- [Systime API](#)

The Systime API provides the means of getting the system time, instead of the scheduler time as you would with functions like `gettime()`.

- [ITM Time Objects](#)

ITM Time Objects are a high-level interface to ITM, a tempo-based scheduler API.

33.47 Clocks

Clock objects are your interface to Max's scheduler.

Collaboration diagram for Clocks:



Typedefs

- `typedef t_object t_clock`

A clock.

Functions

- `void *clock_new (void *obj, method fn)`

Create a new Clock object.

- `void clock_delay (void *x, long n)`

Schedule the execution of a Clock.

- `void clock_unset (void *x)`

Cancel the scheduled execution of a Clock.

- `void clock_fdelay (void *c, double time)`

Schedule the execution of a Clock using a floating-point argument.

- `void clock_gettime (double *time)`

Find out the current logical time of the scheduler in milliseconds as a floating-point number.

- `void setclock_delay (t_object *x, void *c, long time)`

Schedule a Clock on a scheduler.

- `void setclock_unset (t_object *x, void *c)`

Remove a Clock from a scheduler.

- `long setclock_gettime (t_object *x)`

Find out the current time value of a setclock object.

- `void setclock_fdelay (t_object *s, void *c, double time)`

Schedule a Clock on a scheduler, using a floating-point time argument.

- `void setclock_gettime (t_object *s, double *time)`

Find out the current time value of a setclock object in floating-point milliseconds.

- `double systimer_gettime (void)`

While most Max/MSP timing references "logical" time derived from Max's millisecond scheduler, time values produced by the [systimer_gettime\(\)](#) are referenced from the CPU clock and can be used to time real world events with microsecond precision.

- long [gettime](#) (void)

Find out the current logical time of the scheduler in milliseconds.

- void * [scheduler_new](#) (void)

Create a new local scheduler.

- void * [scheduler_set](#) (void *x)

Make a scheduler current, so that future related calls (such as [clock_delay\(\)](#)) will affect the appropriate scheduler.

- void * [scheduler_get](#) ()

Get the currently set scheduler.

- void * [scheduler_fromobject](#) (t_object *o)

Get the scheduler associated with a given object, if any.

- void [scheduler_run](#) (void *x, double until)

Run scheduler events to a selected time.

- void [scheduler_settime](#) (void *x, double time)

Set the current time of the scheduler.

- void [scheduler_gettime](#) (void *x, double *time)

Retrieve the current time of the selected scheduler.

- void [scheduler_shift](#) (void *x, double amount)

Shift scheduler's current time and run time for all pending clock.

33.47.1 Detailed Description

Clock objects are your interface to Max's scheduler. To use the scheduler, you create a new Clock object using [clock_new](#) in your instance creation function. You also have to write a clock function that will be executed when the clock goes off, declared as follows:

```
void myobject_tick (myobject *x);
```

The argument x is determined by the arg argument to [clock_new\(\)](#). Almost always it will be pointer to your object. Then, in one of your methods, use [clock_delay\(\)](#) or [clock_fdelay\(\)](#) to schedule yourself. If you want unschedule yourself, call [clock_unset\(\)](#). To find out what time it is now, use [gettime\(\)](#) or [clock_gettime\(\)](#). More advanced clock operations are possible with the setclock object interface described in Chapter 9. We suggest you take advantage of the higher timing precision of the floating-point clock routines—all standard Max 4 timing objects such as metro use them.

When the user has Overdrive mode enabled, your clock function will execute at interrupt level.

33.47.2 Using Clocks

Under normal circumstances, `gettime` or `clock_gettime` will not be necessary for scheduling purposes if you use `clock_delay` or `clock_fdelay`, but it may be useful for recording the timing of messages or events.

As an example, here's a fragment of how one might go about writing a metronome using the Max scheduler. First, here's the data structure we'll use.

```
typedef struct mymetro {
    t_object *m_obj;
    void *m_clock;
    double m_interval;
    void *m_outlet;
} t_mymetro;
```

We'll assume that the class has been initialized already. Here's the instance creation function that will allocate a new Clock.

```
void *mymetro_create (double defaultInterval)
{
    t_mymetro *x;
    x = (t_mymetro *)newobject(mymetro_class); // allocate space
    x->m_clock = clock_new(x, (method)mymetro_tick); // make a clock
    x->m_interval = defaultInterval; // store the interval
    x->m_outlet = bangout(x); // outlet for ticks
    return x; // return the new object
}
```

Here's the method written to respond to the bang message that starts the metronome.

```
void mymetro_bang (t_mymetro *x)
{
    clock_fdelay(x->m_clock, 0.);
}
```

Here's the Clock function.

```
void mymetro_tick(t_mymetro *x)
{
    clock_fdelay(x->m_clock, x->m_interval);
    // schedule another metronome tick
    outlet_bang(x->m_outlet); // send out a bang
}
```

You may also want to stop the metronome at some point. Here's a method written to respond to the message `stop`. It uses `clock_unset`.

```
void mymetro_stop (t_mymetro *x)
{
    clock_unset(x->m_clock);
}
```

In your object's free function, you should call `freeobject` on any Clocks you've created.

```
void mymetro_free (MyMetro *x)
{
    freeobject((t_object *)x->m_clock);
}
```

33.47.3 Scheduling with setclock Objects

The setclock object allows a more general way of scheduling Clocks by generalizing the advancement of the time associated with a scheduler. Each setclock object's "time" can be changed by a process other than the internal millisecond clock. In addition, the object implements routines that modify the mapping of the internal millisecond clock onto the current value of time in an object. Your object can call a set of routines that use either setclock or the normal millisecond clock transparently. Many Max objects accept the message clock followed by an optional symbol to set their internal scheduling to a named setclock object. The typical implementation passes the binding of a Symbol (the s_thing field) to the Setclock functions. By default, the empty symbol is passed. If the binding has been linked to a setclock object, it will be used to schedule the Clock. Otherwise, the Clock is scheduled using the main internal millisecond scheduler. The Setclock data structure is a replacement for void * since there will be no reason for external objects to access it directly.

33.47.3.1 Using the setclock Object Routines

Here's an example implementation of the relevant methods of a metronome object using the Setclock routines.

```
typedef struct metro
{
    t_object m_ob;
    long m_interval;
    long m_running;
    void *m_clock;
    t_symbol *m_setclock;
} t_metro;
```

Here's the implementation of the routines for turning the metronome on and off. Assume that in the instance creation function, the `t_symbol` `m_setclock` has been set to the empty symbol (`gensym("")`) and `m_clock` has been created; the clock function `metro_tick()` is defined further on.

```
void metro_bang(Metro **x) // turn metronome on
{
    x->m_running = 1;
    setclock_delay(x->m_setclock->s_thing, x->m_clock, 0);
}

void metro_stop(Metro **x)
{
    x->m_running = 0;
    setclock_unset(x->m_setclock->s_thing, x->m_clock);
}
```

Here is the implementation of the clock function `metro_tick()` that runs periodically.

```
void metro_tick(Metro **x)
{
    outlet_bang(x->m_ob.o_outlet);
    if (x->m_running)
        setclock_delay(x->m_setclock->s_thing, x->m_clock, x->m_interval);
}
```

Finally, here is an implementation of the method to respond to the clock message. Note that the function tries to verify that a non-zero value bound to the `t_symbol` passed as an argument is in fact an instance of setclock by checking to see if it responds to the unset message. If not, the metronome refuses to assign the `t_symbol` to its internal `m_setclock` field.

```

void metro_clock(Metro *x, t_symbol *s)
{
    void *old = x->m_setclock->s_thing;
    void *c = 0;

    // the line below can be restated as:
    // if s is the empty symbol
    // or s->s_thing is zero
    // or s->s_thing is non-zero and a setclock object
    if ((s == gensym("")) || ((c = s->s_thing) && zgetfn(c,&s_unset)))
    {
        if (old)
            setclock_unset(old,x->m_clock);
        x->m_setclock = s;
        if (x->m_running)
            setclock_delay(c,x->m_clock,0L);
    }
}

```

33.47.4 Creating Schedulers

If you want to schedule events independently of the time of the global Max scheduler, you can create your own scheduler with [scheduler_new\(\)](#). By calling [scheduler_set\(\)](#) with the newly created scheduler, calls to [clock_new\(\)](#) will create Clocks tied to your scheduler instead of Max's global one. You can then control the time of the scheduler (using [scheduler_settime\(\)](#)) as well as when it executes clock functions (using [scheduler_run\(\)](#)). This is a more general facility than the setclock object routines, but unlike using the time from a setclock object to determine when a Clock function runs, once a Clock is tied to a scheduler.

33.47.5 Function Documentation

33.47.5.1 void [clock_delay](#) (void * *x*, long *n*)

Schedule the execution of a Clock.

[clock_delay\(\)](#) sets a clock to go off at a certain number of milliseconds from the current logical time.

Parameters

- x* Clock to schedule.
- n* Delay, in milliseconds, before the Clock will execute.

See also

[clock_fdelay\(\)](#)

33.47.5.2 void [clock_fdelay](#) (void * *c*, double *time*)

Schedule the execution of a Clock using a floating-point argument.

[clock_fdelay\(\)](#) sets a clock to go off at a certain number of milliseconds from the current logical time.

Parameters

- c* Clock to schedule.
- time* Delay, in milliseconds, before the Clock will execute.

See also[clock_delay\(\)](#)**33.47.5.3 void clock_gettime (double * time)**

Find out the current logical time of the scheduler in milliseconds as a floating-point number.

Parameters

time Returns the current time.

See also[gettime\(\)](#)
[setclock_gettime\(\)](#)
[setclock_gettime\(\)](#)**33.47.5.4 void* clock_new (void * obj, method fn)**

Create a new Clock object.

Normally, [clock_new\(\)](#) is called in your instance creation function—and it cannot be called from a thread other than the main thread. To get rid of a clock object you created, use [freeobject\(\)](#).

Parameters

obj Argument that will be passed to clock function *fn* when it is called. This will almost always be a pointer to your object.

fn Function to be called when the clock goes off, declared to take a single argument as shown in [Using Clocks](#).

Returns

A pointer to a newly created Clock object.

33.47.5.5 void clock_unset (void * x)

Cancel the scheduled execution of a Clock.

[clock_unset\(\)](#) will do nothing (and not complain) if the Clock passed to it has not been set.

Parameters

x Clock to cancel.

33.47.5.6 long gettime (void)

Find out the current logical time of the scheduler in milliseconds.

Returns

Returns the current time.

See also[clock_gettime\(\)](#)**33.47.5.7 void* scheduler_fromobject (t_object * o)**

Get the scheduler associated with a given object, if any.

Parameters

o The object who's scheduler is to be returned.

Returns

This routine returns a pointer to the scheduler or the passed in object,

See also[Creating Schedulers](#)**33.47.5.8 void* scheduler_get ()**

Get the currently set scheduler.

Returns

This routine returns a pointer to the current scheduler,

See also[Creating Schedulers](#)**33.47.5.9 void scheduler_gettime (void * x, double * time)**

Retrieve the current time of the selected scheduler.

Parameters

x The scheduler to query.

time The current time of the selected scheduler.

See also[Creating Schedulers](#)**33.47.5.10 void* scheduler_new (void)**

Create a new local scheduler.

Returns

A pointer to the newly created scheduler.

See also[Creating Schedulers](#)

33.47.5.11 void scheduler_run (void **x*, double *until*)

Run scheduler events to a selected time.

Parameters

- x* The scheduler to advance.
- until* The ending time for this run (in milliseconds).

See also

[Creating Schedulers](#)

33.47.5.12 void* scheduler_set (void **x*)

Make a scheduler current, so that future related calls (such as [clock_delay\(\)](#)) will affect the appropriate scheduler.

Parameters

- x* The scheduler to make current.

Returns

This routine returns a pointer to the previously current scheduler, wsaved and restored when local scheduling is complete.

See also

[Creating Schedulers](#)

33.47.5.13 void scheduler_settime (void **x*, double *time*)

Set the current time of the scheduler.

Parameters

- x* The scheduler to set.
- time* The new current time for the selected scheduler (in milliseconds).

See also

[Creating Schedulers](#)

33.47.5.14 void scheduler_shift (void **x*, double *amount*)

Shift scheduler's current time and run time for all pending clock.

Could be used to change scheduler's time reference without impacting current clocks.

Parameters

- x* The scheduler to affect.

amount Number of milliseconds to shift by.

See also

[Creating Schedulers](#)

33.47.5.15 void setclock_delay (*t_object *x, void *c, long time*)

Schedule a Clock on a scheduler.

Schedules the Clock c to execute at time units after the current time. If scheduler x is 0 or does not point to a setclock object, the internal millisecond scheduler is used. Otherwise c is scheduled on the setclock object's list of Clocks. The Clock should be created with [clock_new\(\)](#), the same as for a Clock passed to [clock_delay\(\)](#).

Parameters

x A setclock object to be used for scheduling this clock.

c Clock object containing the function to be executed.

time Time delay (in the units of the Setclock) from the current time when the Clock will be executed.

See also

[Scheduling with setclock Objects](#)
[setclock_fdelay\(\)](#)

33.47.5.16 void setclock_fdelay (*t_object *s, void *c, double time*)

Schedule a Clock on a scheduler, using a floating-point time argument.

Parameters

s A setclock object to be used for scheduling this clock.

c Clock object containing the function to be executed.

time Time delay (in the units of the Setclock) from the current time when the Clock will be executed.

See also

[Scheduling with setclock Objects](#)
[setclock_delay\(\)](#)

33.47.5.17 void setclock_gettime (*t_object *s, double *time*)

Find out the current time value of a setclock object in floating-point milliseconds.

Parameters

s A setclock object.

time The current time in milliseconds.

See also

[Scheduling with setclock Objects](#)
[setclock_gettime\(\)](#)

33.47.5.18 long setclock_gettime (t_object *x)

Find out the current time value of a setclock object.

Parameters

x A setclock object.

Returns

Returns the current time value of the setclock object scheduler. If scheduler is 0, setclock_gettime is equivalent to the function gettimeofday that returns the current value of the internal millisecond clock.

See also

[Scheduling with setclock Objects](#)
[setclock_gettime\(\)](#)

33.47.5.19 void setclock_unset (t_object *x, void *c)

Remove a Clock from a scheduler.

This function unschedules the Clock *c* in the list of Clocks in the setclock object *x*, or the internal millisecond scheduler if scheduler is 0.

Parameters

x The setclock object that was used to schedule this clock. If 0, the clock is unscheduled from the internal millisecond scheduler.
c Clock object to be removed from the scheduler.

See also

[Scheduling with setclock Objects](#)

33.47.5.20 double systimer_gettime (void)

While most Max/MSP timing references "logical" time derived from Max's millisecond scheduler, time values produced by the [systimer_gettime\(\)](#) are referenced from the CPU clock and can be used to time real world events with microsecond precision.

The standard 'cpuclock' external in Max is a simple wrapper around this function.

Returns

Returns the current real-world time.

33.48 Qelems

Your object's methods may be called at interrupt level.

Collaboration diagram for Qelems:



Typedefs

- `typedef void * t_qelem`

A qelem.

Functions

- `void * qelem_new (void *obj, method fn)`

Create a new Qelem.

- `void qelem_set (void *q)`

Cause a Qelem to execute.

- `void qelem_unset (void *q)`

Cancel a Qelem's execution.

- `void qelem_free (void *x)`

Free a Qelem object created with `qelem_new()`.

- `void qelem_front (void *x)`

Cause a Qelem to execute with a higher priority.

33.48.1 Detailed Description

Your object's methods may be called at interrupt level. This happens when the user has Overdrive mode enabled and one of your methods is called, directly or indirectly, from a scheduler Clock function. This means that you cannot count on doing certain things—like drawing, asking the user what file they would like opened, or calling any Macintosh toolbox trap that allocates or purges memory—from within any method that responds to any message that could be sent directly from another Max object. The mechanism you'll use to get around this limitation is the Qelem (queue element) structure. Qelems also allow processor-intensive tasks to be done at a lower priority than in an interrupt. As an example, drawing on the screen, especially in color, takes a long time in comparison with a task like sending MIDI data.

A Qelem works very much like a Clock. You create a new Qelem in your creation function with `qelem_new` and store a pointer to it in your object. Then you write a queue function, very much like the clock function (it takes the same single argument that will usually be a pointer to your object) that will be called when the Qelem has been set. You set the Qelem to run its function by calling `qelem_set()`.

Often you'll want to use Qelems and Clocks together. For example, suppose you want to update the display for a counter that changes 20 times a second. This can be accomplished by writing a Clock function that

calls `qelem_set()` and then reschedules itself for 50 milliseconds later using the technique shown in the metronome example above. This scheme works even if you call `qelem_set()` faster than the computer can draw the counter, because if a Qelem is already set, `qelem_set()` will not set it again. However, when drawing the counter, you'll display its current value, not a specific value generated in the Clock function.

Note that the Qelem-based defer mechanism discussed later in this chapter may be easier for lowering the priority of one-time events, such as opening a standard file dialog box in response to a read message.

If your Qelem routine sends messages using `outlet_int()` or any other of the outlet functions, it needs to use the lockout mechanism described in the Interrupt Level Considerations section.

33.48.2 Function Documentation

33.48.2.1 void qelem_free (void * *x*)

Free a Qelem object created with `qelem_new()`.

Typically this will be in your object's free funtion.

Parameters

x The Qelem to destroy.

33.48.2.2 void qelem_front (void * *x*)

Cause a Qelem to execute with a higher priority.

This function is identical to `qelem_set()`, except that the Qelem's function is placed at the front of the list of routines to execute in the main thread instead of the back. Be polite and only use `qelem_front()` only for special time-critical applications.

Parameters

x The Qelem whose function will be executed in the main thread.

33.48.2.3 void* qelem_new (void * *obj*, method *fn*)

Create a new Qelem.

The created Qelem will need to be freed using `qelem_free()`, do not use `freeobject()`.

Parameters

obj Argument to be passed to function *fun* when the Qelem executes. Normally a pointer to your object.

fn Function to execute.

Returns

A pointer to a Qelem instance. You need to store this value to pass to `qelem_set()`.

Remarks

Any kind of drawing or calling of Macintosh Toolbox routines that allocate or purge memory should be done in a Qelem function.

33.48.2.4 void qelem_set (void * *q*)

Cause a Qelem to execute.

Parameters

q The Qelem whose function will be executed in the main thread.

Remarks

The key behavior of [qelem_set\(\)](#) is this: if the Qelem object has already been set, it will not be set again. (If this is not what you want, see [defer\(\)](#).) This is useful if you want to redraw the state of some data when it changes, but not in response to changes that occur faster than can be drawn. A Qelem object is unset after its queue function has been called.

33.48.2.5 void qelem_unset (void * *q*)

Cancel a Qelem's execution.

If the Qelem's function is set to be called, [qelem_unset\(\)](#) will stop it from being called. Otherwise, [qelem_unset\(\)](#) does nothing.

Parameters

q The Qelem whose execution you wish to cancel.

33.49 Systime API

The Systime API provides the means of getting the system time, instead of the scheduler time as you would with functions like [gettime\(\)](#).

Collaboration diagram for Systime API:



Data Structures

- struct [t_datetime](#)

The Systime data structure.

Enumerations

- enum [e_max_dateflags](#) {

 SYSDATEFORMAT_FLAGS_SHORT = 1,

 SYSDATEFORMAT_FLAGS_MEDIUM = 2,

 SYSDATEFORMAT_FLAGS_LONG = 3
 }

Flags for the [sysdateformat_formatdatetime\(\)](#) function.

Functions

- unsigned long [systime_ticks](#) (void)

Find out the operating system's time in ticks.
- unsigned long [systime_ms](#) (void)

Find out the operating system's time in milliseconds.
- void [systime_datetime](#) ([t_datetime](#) *d)

Find out the operating system's date and time.
- unsigned long [systime_seconds](#) (void)

Find out the operating system's time in seconds.
- void [systime_secondstodate](#) (unsigned long secs, [t_datetime](#) *d)

Convert a time in seconds into a [t_datetime](#) representation.
- unsigned long [systime_datetoseconds](#) ([t_datetime](#) *d)

Convert a [t_datetime](#) representation of time into seconds.
- void [sysdateformat_strftimetodatetime](#) (char *strf, [t_datetime](#) *d)

Fill a [t_datetime](#) struct with a datetime formatted string.

- void [sysdateformat_formatdatetime](#) ([t_datetime](#) **d*, long *dateflags*, long *timeflags*, char **s*, long *buflen*)

Get a human friendly string representation of a [t_datetime](#).

33.49.1 Detailed Description

The Systime API provides the means of getting the system time, instead of the scheduler time as you would with functions like [getttime\(\)](#).

33.49.2 Enumeration Type Documentation

33.49.2.1 enum e_max_dateflags

Flags for the [sysdateformat_formatdatetime\(\)](#) function.

Enumerator:

`SYSDATEFORMAT_FLAGS_SHORT` short
`SYSDATEFORMAT_FLAGS_MEDIUM` medium
`SYSDATEFORMAT_FLAGS_LONG` long

33.49.3 Function Documentation

33.49.3.1 void sysdateformat_formatdatetime ([t_datetime](#) **d*, long *dateflags*, long *timeflags*, char **s*, long *buflen*)

Get a human friendly string representation of a [t_datetime](#).

For example: "Today", "Yesterday", etc.

Parameters

- d* The address of a [t_datetime](#) to fill.
- dateflags* One of the values defined in [e_max_dateflags](#).
- timeflags* Currently unused. Pass 0.
- s* An already allocated string to hold the human friendly result.
- buflen* The number of characters allocated to the string *s*.

33.49.3.2 void sysdateformat_strftimetodatetime (char **strf*, [t_datetime](#) **d*)

Fill a [t_datetime](#) struct with a datetime formatted string.

For example, the string "2007-12-24 12:21:00".

Parameters

- strf* A string containing the datetime.
- d* The address of a [t_datetime](#) to fill.

33.49.3.3 void systime_datetime (*t_datetime* * *d*)

Find out the operating system's date and time.

Parameters

d Returns the system's date and time in a [t_datetime](#) data structure.

33.49.3.4 unsigned long systime_datetoseconds (*t_datetime* * *d*)

Convert a [t_datetime](#) representation of time into seconds.

Parameters

d The address of a [t_datetime](#) that contains a valid period of time.

Returns

The number of seconds represented by *d*.

33.49.3.5 unsigned long systime_ms (void)

Find out the operating system's time in milliseconds.

Returns

the system time in milliseconds.

33.49.3.6 unsigned long systime_seconds (void)

Find out the operating system's time in seconds.

Returns

the system time in seconds.

33.49.3.7 void systime_secondstodate (unsigned long *secs*, *t_datetime* * *d*)

Convert a time in seconds into a [t_datetime](#) representation.

Parameters

secs A number of seconds to be represented as a [t_datetime](#).

d The address of a [t_datetime](#) that will be filled with the converted value.

33.49.3.8 unsigned long systime_ticks (void)

Find out the operating system's time in ticks.

Returns

the system time in ticks.

33.50 ITM Time Objects

ITM Time Objects are a high-level interface to ITM, a tempo-based scheduler API.

Collaboration diagram for ITM Time Objects:



Enumerations

- enum {

 TIME_FLAGS_LOCATION = 1,

 TIME_FLAGS_TICKSONLY = 2,

 TIME_FLAGS_FIXEDONLY = 4,

 TIME_FLAGS_LOOKAHEAD = 8,

 TIME_FLAGS_USECLOCK = 16,

 TIME_FLAGS_USEQELEM = 32,

 TIME_FLAGS_FIXED = 64,

 TIME_FLAGS_PERMANENT = 128,

 TIME_FLAGS_TRANSPORT = 256,

 TIME_FLAGS_EVENTLIST = 512,

 TIME_FLAGS_CHECKSCHEDULE = 1024,

 TIME_FLAGS_LISTENTICKS = 2048,

 TIME_FLAGS_NOUNITS = 4096,

 TIME_FLAGS_BBUSOURCE = 8192,

 TIME_FLAGS_POSITIVE = 16384
 }

Flags that determine attribute and time object behavior.

Functions

- void * **itm_getglobal** (void)

Return the global (default / unnamed) itm object.
- void * **itm_getnamed** (**t_symbol** *s, void *scheduler, **t_symbol** *defaultclocksourcename, long create)

Return a named itm object.
- void **itm_reference** (**t_itm** *x)

Reference an itm object.
- void **itm_dereference** (**t_itm** *x)

Stop referencing an itm object.
- double **itm_gettime** (**t_itm** *x)

Report the current internal time.

- double `itm_getticks (t_itm *x)`

Report the current time of the item in ticks.

- void `itm_dump (t_itm *x)`

Print diagnostic information about an item object to the Max window.

- void `itm_settimesignature (t_itm *x, long num, long denom, long flags)`

Set an item object's current time signature.

- void `itm_gettimesignature (t_itm *x, long *num, long *denom)`

Query an item object for its current time signature.

- void `itm_pause (t_itm *x)`

Pause the passage of time for an item object.

- void `itm_resume (t_itm *x)`

Start the passage of time for an item object, from it's current location.

- long `itm_getstate (t_itm *x)`

Find out if time is currently progressing for a given item object.

- void `itm_setresolution (double res)`

Set the number of ticks-per-quarter-note globally for the item system.

- double `itm_getresolution (void)`

Get the number of ticks-per-quarter-note globally from the item system.

- `t_symbol * itm_getname (t_itm *x)`

Given an item object, return its name.

- double `itm_tickstoms (t_itm *x, double ticks)`

Convert a time value in ticks to the equivalent value in milliseconds, given the context of a specified item object.

- double `itm_mstoticks (t_itm *x, double ms)`

Convert a time value in milliseconds to the equivalent value in ticks, given the context of a specified item object.

- double `itm_mstosamps (t_itm *x, double ms)`

Convert a time value in milliseconds to the equivalent value in samples, given the context of a specified item object.

- double `itm_sampstoms (t_itm *x, double samps)`

Convert a time value in samples to the equivalent value in milliseconds, given the context of a specified item object.

- void `itm_barbeatunitstoticks (t_itm *x, long bars, long beats, double units, double *ticks, char position)`

Convert a time value in bbu to the equivalent value in ticks, given the context of a specified item object.

- void **itm_ticksstobarbeatunits** (**t_itm** *x, double ticks, long *bars, long *beats, double *units, char position)

Convert a time value in bbu to the equivalent value in ticks, given the context of a specified itm object.
- long **itm_isunitfixed** (**t_symbol** *u)

Given the name of a time unit (e.g.
- void **time_stop** (**t_timeobject** *x)

Stop a currently scheduled time object.
- void **time_tick** (**t_timeobject** *x)

Execute a time object's task, then if it was already set to execute, reschedule for the current interval value of the object.
- double **time_getms** (**t_timeobject** *x)

Convert the value of a time object to milliseconds.
- double **time_getticks** (**t_timeobject** *x)

Convert the value of a time object to ticks.
- void **time_getphase** (**t_timeobject** *tx, double *phase, double *slope, double *ticks)

Return the phase of the ITM object (transport) associated with a time object.
- void **time_listen** (**t_timeobject** *x, **t_symbol** *attr, long flags)

Specify that a millisecond-based attribute to be updated automatically when the converted milliseconds of the time object's value changes.
- void **time_setvalue** (**t_timeobject** *tx, **t_symbol** *s, long argc, **t_atom** *argv)

Set the current value of a time object (either an interval or a position) using a Max message.
- void **class_time_addattr** (**t_class** *c, char *attrname, char *attrlabel, long flags)

Create an attribute permitting a time object to be changed in a user-friendly way.
- void * **time_new** (**t_object** *owner, **t_symbol** *attrname, **method** tick, long flags)

Create a new time object.
- **t_object** * **time_getnamed** (**t_object** *owner, **t_symbol** *attrname)

Return a time object associated with an attribute of an owning object.
- long **time_isfixedunit** (**t_timeobject** *x)

Return whether this time object currently holds a fixed (millisecond-based) value.
- void **time_schedule** (**t_timeobject** *x, **t_timeobject** *quantize)

Schedule a task, with optional quantization.
- void **time_schedule_limit** (**t_timeobject** *x, **t_timeobject** *quantize)

Schedule a task, with optional minimum interval.,
- void **time_now** (**t_timeobject** *x, **t_timeobject** *quantize)

Schedule a task for right now, with optional quantization.

- `void * time_getitm (t_timeobject *ox)`
Return the ITM object associated with this time object.
- `double time_calcquantize (t_timeobject *ox, t_itm *vitm, t_timeobject *oq)`
Calculate the quantized interval (in ticks) if this time object were to be scheduled at the current time.
- `void time_setclock (t_timeobject *tx, t_symbol *sc)`
Associate a named setclock object with a time object (unsupported).

Variables

- `BEGIN_USING_C_LINKAGE typedef t_object t_itm`
A low-level object for tempo-based scheduling.
- `BEGIN_USING_C_LINKAGE typedef t_object t_timeobject`
A high-level time object for tempo-based scheduling.

33.50.1 Detailed Description

ITM Time Objects are a high-level interface to ITM, a tempo-based scheduler API. They provide an abstraction so your object can schedule events either in milliseconds (as traditional clock objects) or ticks (tempo-relative units).

33.50.2 Enumeration Type Documentation

33.50.2.1 anonymous enum

Flags that determine attribute and time object behavior.

Enumerator:

- `TIME_FLAGS_LOCATION` 1 1 0 location-based bar/beat/unit values (as opposed to interval values, which are 0 0 0 relative)
- `TIME_FLAGS_TICKSONLY` only ticks-based values (not ms) are acceptable
- `TIME_FLAGS_FIXEDONLY` only fixed values (ms, hz, samples) are acceptable
- `TIME_FLAGS_LOOKAHEAD` add lookahead attribute (unsupported)
- `TIME_FLAGS_USECLOCK` this time object will schedule events, not just hold a value
- `TIME_FLAGS_USEQELEM` this time object will defer execution of scheduled events to low priority thread
- `TIME_FLAGS_FIXED` will only use normal clock (i.e., will never execute out of ITM)
- `TIME_FLAGS_PERMANENT` event will be scheduled in the permanent list (tied to a specific time)
- `TIME_FLAGS_TRANSPORT` add a transport attribute
- `TIME_FLAGS_EVENTLIST` add an eventlist attribute (unsupported)

TIME_FLAGS_CHECKSCHEDULE internal use only
TIME_FLAGS_LISTENTICKS flag for time_listen: only get notifications if the time object holds tempo-relative values
TIME_FLAGS_NOUNITS internal use only
TIME_FLAGS_BBUSOURCE source time was in bar/beat/unit values, need to recalculate when time sig changes
TIME_FLAGS_POSITIVE constrain any values < 0 to 0

33.50.3 Function Documentation

33.50.3.1 void class_time_addattr (t_class * *c*, char * *attrname*, char * *attrlabel*, long *flags*)

Create an attribute permitting a time object to be changed in a user-friendly way.

Parameters

c Class being initialized.
attrname Name of the attribute associated with the time object.
attrlabel Descriptive label for the attribute (appears in the inspector)
flags Options, see "Flags that determine time object behavior" above

33.50.3.2 void itm_barbeatunitstoticks (t_itm * *x*, long *bars*, long *beats*, double *units*, double * *ticks*, char *position*)

Convert a time value in bbu to the equivalent value in ticks, given the context of a specified itm object.

Parameters

x An itm object.
bars The measure number of the location/position.
beats The beat number of the location/position.
units The number of ticks past the beat of the location/position.
ticks The address of a variable to hold the number of ticks upon return.
position Set this parameter to **TIME_FLAGS_LOCATION** or to zero (for position mode).

33.50.3.3 void itm_dereference (t_itm * *x*)

Stop referencing an itm object.

When you are done using an itm object, you must call this function to decrement its reference count.

Parameters

x The itm object.

33.50.3.4 void itm_dump (*t_itm* **x*)

Print diagnostic information about an itm object to the Max window.

Parameters

x The itm object.

33.50.3.5 void* itm_getglobal (void)

Return the global (default / unnamed) itm object.

Returns

The global [t_itm](#) object.

33.50.3.6 t_symbol* itm_getname (*t_itm* **x*)

Given an itm object, return its name.

Parameters

x The itm object.

Returns

The name of the itm.

33.50.3.7 void* itm_getnamed (*t_symbol* **s*, void **scheduler*, *t_symbol* **defaultclocksourcename*, long *create*)

Return a named itm object.

Parameters

s The name of the itm to return.

scheduler

defaultclocksourcename

create If non-zero, then create this named itm should it not already exist.

Returns

The global [t_itm](#) object.

33.50.3.8 double itm_getresolution (void)

Get the number of ticks-per-quarter-note globally from the itm system.

Returns

The number of ticks-per-quarter-note.

See also

[itm_setresolution\(\)](#)

33.50.3.9 long itm_getstate (t_itm * x)

Find out if time is currently progressing for a given itm object.

Parameters

x The itm object.

Returns

Returns non-zero if the time is running, or zero if it is paused.

See also

[itm_pause\(\)](#)
[itm_resume\(\)](#)

33.50.3.10 double itm_getticks (t_itm * x)

Report the current time of the itm in ticks.

You can use functions such as [itm_ticksbarbeatunits\(\)](#) or [itm_tickstoms\(\)](#) to convert to a different representation of the time.

Parameters

x The itm object.

Returns

The current time in ticks.

33.50.3.11 double itm_gettime (t_itm * x)

Report the current internal time.

This is the same as calling [clock_gettime\(\)](#);

Parameters

x The itm object.

Returns

The current internal time.

33.50.3.12 void item_gettimesignature (t_item **x*, long **num*, long **denom*)

Query an item object for its current time signature.

Parameters

x The item object.

num The address of a variable to hold the top number of the time signature upon return.

denom The address of a variable to hold the bottom number of the time signature upon return.

33.50.3.13 long item_isunitfixed (t_symbol **u*)

Given the name of a time unit (e.g.

'ms', 'ticks', 'bbu', 'samples', etc.), determine whether the unit is fixed (doesn't change with tempo, time-signature, etc.) or whether it is flexible.

Parameters

u The name of the time unit.

Returns

Zero if the unit is fixed (milliseconds, for example) or non-zero if it is flexible (ticks, for example).

33.50.3.14 double item_mstosamps (t_item **x*, double *ms*)

Convert a time value in milliseconds to the equivalent value in samples, given the context of a specified item object.

Parameters

x An item object.

ms A time specified in ms.

Returns

The time specified in samples.

33.50.3.15 double item_mstoticks (t_item **x*, double *ms*)

Convert a time value in milliseconds to the equivalent value in ticks, given the context of a specified item object.

Parameters

x An item object.

ms A time specified in ms.

Returns

The time specified in ticks.

33.50.3.16 void itm_pause (*t_itm* **x*)

Pause the passage of time for an itm object.

This is the equivalent to setting the state of a transport object to 0 with a toggle.

Parameters

x The itm object.

33.50.3.17 void itm_reference (*t_itm* **x*)

Reference an itm object.

When you are using an itm object, you must call this function to increment its reference count.

Parameters

x The itm object.

33.50.3.18 void itm_resume (*t_itm* **x*)

Start the passage of time for an itm object, from it's current location.

This is the equivalent to setting the state of a transport object to 0 with a toggle.

Parameters

x The itm object.

33.50.3.19 double itm_sampstoms (*t_itm* **x*, double *samps*)

Convert a time value in samples to the equivalent value in milliseconds, given the context of a specified itm object.

Parameters

x An itm object.

samps A time specified in samples.

Returns

The time specified in ms.

33.50.3.20 void itm_setresolution (double *res*)

Set the number of ticks-per-quarter-note globally for the itm system.

The default is 480.

Parameters

res The number of ticks-per-quarter-note.

See also[itm_getresolution\(\)](#)**33.50.3.21 void itm_settimesignature (t_itm **x*, long *num*, long *denom*, long *flags*)**

Set an item object's current time signature.

Parameters

- x* The item object.
- num* The top number of the time signature.
- denom* The bottom number of the time signature.
- flags* Currently unused -- pass zero.

33.50.3.22 void itm_ticksbarbeatunits (t_itm **x*, double *ticks*, long **bars*, long **beats*, double **units*, char *position*)

Convert a time value in bbu to the equivalent value in ticks, given the context of a specified item object.

Parameters

- x* An item object.
- ticks* The number of ticks to translate into a time represented as bars, beats, and ticks.
- bars* The address of a variable to hold the measure number of the location/position upon return.
- beats* The address of a variable to hold the beat number of the location/position upon return.
- units* The address of a variable to hold the number of ticks past the beat of the location/position upon return.
- position* Set this parameter to [TIME_FLAGS_LOCATION](#) or to zero (for position mode).

33.50.3.23 double itm_tickstoms (t_itm **x*, double *ticks*)

Convert a time value in ticks to the equivalent value in milliseconds, given the context of a specified item object.

Parameters

- x* An item object.
- ticks* A time specified in ticks.

Returns

The time specified in ms.

33.50.3.24 double time_calcquantize (*t_timeobject* * *ox*, *t_itm* * *vitm*, *t_timeobject* * *oq*)

Calculate the quantized interval (in ticks) if this time object were to be scheduled at the current time.

Parameters

- ox* Time object.
- vitm* The associated ITM object (use [time_getitm\(\)](#) to determine it).
- oq* A time object that holds a quantization interval, can be NULL.

Returns

Interval (in ticks) for scheduling this object.

33.50.3.25 void* time_getitm (*t_timeobject* * *ox*)

Return the ITM object associated with this time object.

Parameters

- ox* Time object.

Returns

The associated [t_itm](#) object.

33.50.3.26 double time_getms (*t_timeobject* * *x*)

Convert the value of a time object to milliseconds.

Parameters

- x* The time object.

Returns

The time object's value, converted to milliseconds.

33.50.3.27 *t_object time_getnamed (*t_object* * *owner*, *t_symbol* * *attrname*)**

Return a time object associated with an attribute of an owning object.

Parameters

- owner* Object that owns this time object (task routine, if any, will pass owner as argument).
- attrname* Name of the attribute associated with the time object.

Returns

The [t_timeobject](#) associated with the named attribute.

33.50.3.28 void time_getphase (t_timeobject * *tx*, double * *phase*, double * *slope*, double * *ticks*)

Return the phase of the ITM object (transport) associated with a time object.

Parameters

tx The time object.

phase Pointer to a double to receive the progress within the specified time value of the associated ITM object.

slope Pointer to a double to receive the slope (phase difference) within the specified time value of the associated ITM object.

ticks

33.50.3.29 double time_getticks (t_timeobject * *x*)

Convert the value of a time object to ticks.

Parameters

x The time object.

Returns

The time object's value, converted to ticks.

33.50.3.30 long time_isfixedunit (t_timeobject * *x*)

Return whether this time object currently holds a fixed (millisecond-based) value.

Parameters

x Time object.

Returns

True if time object's current value is fixed, false if it is tempo-relative.

33.50.3.31 void time_listen (t_timeobject * *x*, t_symbol * *attr*, long *flags*)

Specify that a millisecond-based attribute to be updated automatically when the converted milliseconds of the time object's value changes.

Parameters

x The time object.

attr Name of the millisecond based attribute in the owning object that will be updated

flags If TIME_FLAGS_LISTENTICKS is passed here, updating will not happen if the time value is fixed (ms) based

33.50.3.32 void* time_new (t_object *owner, t_symbol *attrname, method tick, long flags)

Create a new time object.

Parameters

owner Object that will own this time object (task routine, if any, will pass owner as argument).

attrname Name of the attribute associated with the time object.

tick Task routine that will be executed (can be NULL)

flags Options, see "Flags that determine time object behavior" above

Returns

The newly created [t_timeobject](#).

33.50.3.33 void time_now (t_timeobject *x, t_timeobject *quantize)

Schedule a task for right now, with optional quantization.

Parameters

x The time object that schedules temporary events. The time interval is ignored and 0 ticks is used instead.

quantize A time object that holds a quantization interval, can be NULL for no quantization.

33.50.3.34 void time_schedule (t_timeobject *x, t_timeobject *quantize)

Schedule a task, with optional quantization.

Parameters

x The time object that schedules temporary events (must have been created with TIME_FLAGS_USECLOCK but not TIME_FLAGS_PERMANENT)

quantize A time object that holds a quantization interval, can be NULL for no quantization.

33.50.3.35 void time_schedule_limit (t_timeobject *x, t_timeobject *quantize)

Schedule a task, with optional minimum interval.,

Parameters

x The time object that schedules temporary events (must have been created with TIME_FLAGS_USECLOCK but not TIME_FLAGS_PERMANENT)

quantize The minimum interval into the future when the event can occur, can be NULL if there is no minimum interval.

33.50.3.36 void time_setclock (*t_timeobject* **tx*, *t_symbol* **sc*)

Associate a named setclock object with a time object (unsupported).

Parameters

- tx* Time object.
- sc* Name of an associated setclock object.

33.50.3.37 void time_setvalue (*t_timeobject* **tx*, *t_symbol* **s*, *long* *argc*, *t_atom* **argv*)

Set the current value of a time object (either an interval or a position) using a Max message.

Parameters

- tx* The time object.
- s* Message selector.
- argc* Count of arguments.
- argv* Message arguments.

33.50.3.38 void time_stop (*t_timeobject* **x*)

Stop a currently scheduled time object.

Parameters

- x* The time object.

33.50.3.39 void time_tick (*t_timeobject* **x*)

Execute a time object's task, then if it was already set to execute, reschedule for the current interval value of the object.

Parameters

- x* The time object.

33.50.4 Variable Documentation**33.50.4.1 BEGIN_USING_C_LINKAGE *typedef t_object t_itm***

A low-level object for tempo-based scheduling.

See also

- [t_timeobject](#)
- [ITM](#)

33.50.4.2 BEGIN_USING_C_LINKAGE `typedef t_object t_timeobject`

A high-level time object for tempo-based scheduling.

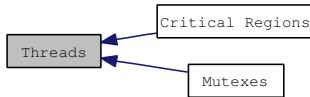
See also

[t_itm](#)
[ITM](#)

33.51 Threads

In Max, there are several threads of execution.

Collaboration diagram for Threads:



Modules

- [Critical Regions](#)

A critical region is a simple mechanism that prevents multiple threads from accessing at once code protected by the same critical region.

- [Mutexes](#)

Defines

- #define [ATOMIC_INCREMENT](#)(pv) (_InterlockedIncrement(pv))

Use this routine for incrementing a global counter using a threadsafe and multiprocessor safe method.

- #define [ATOMIC_DECREMENT](#)(pv) (_InterlockedDecrement(pv))

Use this routine for decrementing a global counter using a threadsafe and multiprocessor safe method.

Typedefs

- [typedef void t_thread](#)

A Max thread.

- [typedef void * t_systhread](#)

An opaque thread instance pointer.

- [typedef void * t_systhread_mutex](#)

An opaque mutex handle.

- [typedef void * t_systhread_cond](#)

An opaque cond handle.

Enumerations

- enum [e_max_systhread_mutex_flags](#) {

 [SYSTHREAD_MUTEX_NORMAL](#) = 0x00000000,

 [SYSTHREAD_MUTEX_ERRORCHECK](#) = 0x00000001,

 [SYSTHREAD_MUTEX_RECURSIVE](#) = 0x00000002
 }

`systhread_mutex_new()` flags

Functions

- void `schedule` (void *ob, method fun, long when, t_symbol *sym, short argc, Atom *argv)
Cause a function to be executed at the timer level at some time in the future.
- void `schedule_delay` (void *ob, method fun, long delay, t_symbol *sym, short argc, t_atom *argv)
Cause a function to be executed at the timer level at some time in the future specified by a delay offset.
- long `isr` (void)
Determine whether your code is executing in the Max scheduler thread.
- void * `defer` (void *ob, method fn, t_symbol *sym, short argc, t_atom *argv)
Defer execution of a function to the main thread if (and only if) your function is executing in the scheduler thread.
- void * `defer_low` (void *ob, method fn, t_symbol *sym, short argc, t_atom *argv)
Defer execution of a function to the back of the queue on the main thread.
- long `systhread_create` (method entryproc, void *arg, long stacksize, long priority, long flags, t_systhread *thread)
Create a new thread.
- long `systhread_terminate` (t_systhread thread)
Forcefully kill a thread -- not recommended.
- void `systhread_sleep` (long milliseconds)
Suspend the execution of the calling thread.
- void `systhread_exit` (long status)
Exit the calling thread.
- long `systhread_join` (t_systhread thread, unsigned int *retval)
Wait for thread to quit and get return value from `systhread_exit()`.
- t_systhread `systhread_self` (void)
Return the thread instance pointer for the calling thread.
- void `systhread_setpriority` (t_systhread thread, int priority)
Set the thread priority for the given thread.
- int `systhread_getpriority` (t_systhread thread)
Get the thread priority for the given thread.
- short `systhread_ismainthread` (void)
Check to see if the function currently being executed is in the main thread.
- short `systhread_istimerthread` (void)
Check to see if the function currently being executed is in the scheduler thread.

33.51.1 Detailed Description

In Max, there are several threads of execution. The details of these threads are highlighted in the article "Event Priority in Max (Scheduler vs. Queue)" located online at <http://www.cycling74.com/story/2005/5/2/133649/9742>.

Not all of the details of Max's threading model are expounded here. Most important to understand is that we typically deal the scheduler (which when overdrive is on runs in a separate and high priority thread) and the low priority queue (which always runs in the main application thread).

See also

<http://www.cycling74.com/twiki/bin/view/ProductDocumentation/JitterSdkSchedQueue>
<http://www.cycling74.com/story/2005/5/2/133649/9742>

33.51.2 Define Documentation

33.51.2.1 #define ATOMIC_DECREMENT(*pv*) (*_InterlockedDecrement(pv)*)

Use this routine for decrementing a global counter using a threadsafe and multiprocessor safe method.

Parameters

pv pointer to the (int) counter.

33.51.2.2 #define ATOMIC_INCREMENT(*pv*) (*_InterlockedIncrement(pv)*)

Use this routine for incrementing a global counter using a threadsafe and multiprocessor safe method.

Parameters

pv pointer to the (int) counter.

33.51.3 Enumeration Type Documentation

33.51.3.1 enum e_max_systhread_mutex_flags

[systhread_mutex_new\(\)](#) flags

Enumerator:

SYSTHREAD_MUTEX_NORMAL Normal.
SYSTHREAD_MUTEX_ERRORCHECK Error-checking.
SYSTHREAD_MUTEX_RECURSIVE Recursive.

33.51.4 Function Documentation

33.51.4.1 void* defer (void * *ob*, method *fn*, t_symbol * *sym*, short *argc*, t_atom * *argv*)

Defer execution of a function to the main thread if (and only if) your function is executing in the scheduler thread.

Parameters

- ob*** First argument passed to the function fun when it executes.
- fn*** Function to be called, see below for how it should be declared.
- sym*** Second argument passed to the function fun when it executes.
- argc*** Count of arguments in argv. argc is also the third argument passed to the function fun when it executes.
- argv*** Array containing a variable number of [t_atom](#) function arguments. If this argument is non-zero, defer allocates memory to make a copy of the arguments (according to the size passed in argc) and passes the copied array to the function fun when it executes as the fourth argument.

Returns

Return values is for internal Cycling '74 use only.

Remarks

This function uses the [isr\(\)](#) routine to determine whether you're at the Max timer interrupt level (in the scheduler thread). If so, [defer\(\)](#) creates a Qelem (see [Qelems](#)), calls [qelem_front\(\)](#), and its queue function calls the function fn you passed with the specified arguments. If you're not in the scheduler thread, the function is executed immediately with the arguments. Note that this implies that [defer\(\)](#) is not appropriate for using in situations such as Device or File manager I/O completion routines. The [defer_low\(\)](#) function is appropriate however, because it always defers.

The deferred function should be declared as follows:

```
void myobject_do (myObject *client, t_symbol *s, short argc, t_atom *argv);
```

See also

[defer_low\(\)](#)

33.51.4.2 void* defer_low (void * ob, method fn, t_symbol * sym, short argc, t_atom * argv)

Defer execution of a function to the back of the queue on the main thread.

Parameters

- ob*** First argument passed to the function fun when it executes.
- fn*** Function to be called, see below for how it should be declared.
- sym*** Second argument passed to the function fun when it executes.
- argc*** Count of arguments in argv. argc is also the third argument passed to the function fun when it executes.
- argv*** Array containing a variable number of [t_atom](#) function arguments. If this argument is non-zero, defer allocates memory to make a copy of the arguments (according to the size passed in argc) and passes the copied array to the function fun when it executes as the fourth argument.

Returns

Return values is for internal Cycling '74 use only.

Remarks

[defer_low\(\)](#) always defers a call to the function fun whether you are already in the main thread or not, and uses [qelem_set\(\)](#), not [qelem_front\(\)](#). This function is recommended for responding to messages that will cause your object to open a dialog box, such as read and write.

The deferred function should be declared as follows:

```
void myobject_do (myObject *client, t_symbol *s, short argc, t_atom *argv);
```

See also

[defer\(\)](#)

33.51.4.3 long isr (void)

Determine whether your code is executing in the Max scheduler thread.

Returns

This function returns non-zero if you are within Max's scheduler thread, zero otherwise. Note that if your code sets up other types of interrupt-level callbacks, such as for other types of device drivers used in asynchronous mode, isr will return false.

33.51.4.4 void schedule (void * ob, method fun, long when, t_symbol * sym, short argc, Atom * argv)

Cause a function to be executed at the timer level at some time in the future.

Parameters

ob First argument passed to the function fun when it executes.

fun Function to be called, see below for how it should be declared.

when The logical time that the function fun will be executed.

sym Second argument passed to the function fun when it executes.

argc Count of arguments in argv. argc is also the third argument passed to the function fun when it executes.

argv Array containing a variable number of [t_atom](#) function arguments. If this argument is non-zero, defer allocates memory to make a copy of the arguments (according to the size passed in argc) and passes the copied array to the function fun when it executes as the fourth argument.

Remarks

[schedule\(\)](#) calls a function at some time in the future. Unlike [defer\(\)](#), the function is called in the scheduling loop when logical time is equal to the specified value when. This means that the function could be called at interrupt level, so it should follow the usual restrictions on interrupt-level conduct. The function fun passed to schedule should be declared as follows:

```
void myobject_do (myObject *client, t_symbol *s, short argc, t_atom *argv);
```

Remarks

One use of [schedule\(\)](#) is as an alternative to using the lockout flag.

See also

[defer\(\)](#)

33.51.4.5 void schedule_delay (void * *ob*, method *fun*, long *delay*, t_symbol * *sym*, short *argc*, t_atom * *argv*)

Cause a function to be executed at the timer level at some time in the future specified by a delay offset.

Parameters

- ob*** First argument passed to the function fun when it executes.
- fun*** Function to be called, see below for how it should be declared.
- delay*** The delay from the current time before the function will be executed.
- sym*** Second argument passed to the function fun when it executes.
- argc*** Count of arguments in argv. argc is also the third argument passed to the function fun when it executes.
- argv*** Array containing a variable number of **t_atom** function arguments. If this argument is non-zero, defer allocates memory to make a copy of the arguments (according to the size passed in argc) and passes the copied array to the function fun when it executes as the fourth argument.

Remarks

[schedule_delay\(\)](#) is similar to [schedule\(\)](#) but allows you to specify the time as a delay rather than a specific logical time.

One use of [schedule\(\)](#) or [schedule_delay\(\)](#) is as an alternative to using the lockout flag. Here is an example click method that calls [schedule\(\)](#) instead of [outlet_int\(\)](#) surrounded by [lockout_set\(\)](#) calls.

```
void myobject_click (t_myobject *x, Point pt, short modifiers)
{
    t_atom a[1];
    a[0].a_type = A_LONG;
    a[0].a_w.w_long = Random();
    schedule_delay(x, myobject_sched, 0, 0, 1, a);
}

void myobject_sched (t_myobject *x, t_symbol *s, short ac, t_atom *av)
{
    outlet_int(x->m_out, av->a_w.w_long);
}
```

See also

[schedule\(\)](#)

33.51.4.6 long systhread_create (method *entryproc*, void * *arg*, long *stacksize*, long *priority*, long *flags*, t_systhread * *thread*)

Create a new thread.

Parameters

- entryproc*** A method to call in the new thread when the thread is created.
- arg*** An argument to pass to the method specified for entryproc. Typically this might be a pointer to your object's struct.
- stacksize*** Not used. Pass 0 for this argument.

priority Pass 0 for default priority. The priority can range from -32 to 32 where -32 is low, 0 is default and 32 is high.

flags Not used. Pass 0 for this argument.

thread The address of a [t_systhread](#) where this thread's instance pointer will be stored.

Returns

A Max error code as defined in [e_max_errorcodes](#).

33.51.4.7 void systhread_exit (long *status*)

Exit the calling thread.

Call this from within a thread made using [systhread_create\(\)](#) when the thread is no longer needed.

Parameters

status You will typically pass 0 for status. This value will be accessible by [systhread_join\(\)](#), if needed.

33.51.4.8 int systhread_getpriority (t_systhread *thread*)

Get the thread priority for the given thread.

Parameters

thread The thread for which to find the priority.

Returns

The current priority value for the given thread.

33.51.4.9 short systhread_ismainthread (void)

Check to see if the function currently being executed is in the main thread.

Returns

Returns true if the function is being executed in the main thread, otherwise false.

33.51.4.10 short systhread_istimerthread (void)

Check to see if the function currently being executed is in the scheduler thread.

Returns

Returns true if the function is being executed in the main thread, otherwise false.

33.51.4.11 long systhread_join (t_systhread *thread*, unsigned int * *retval*)

Wait for thread to quit and get return value from [systhread_exit\(\)](#).

Parameters

thread The thread to join.

retval The address of a long to hold the return value (status) from [systhread_exit\(\)](#).

Returns

A Max error code as defined in [e_max_errorcodes](#).

Remarks

If your object is freed, and your thread function accesses memory from your object, then you will obviously have a memory violation. A common use of [systhread_join\(\)](#) is to prevent this situation by waiting (in your free method) for the thread to exit.

33.51.4.12 t_systhread systhread_self (void)

Return the thread instance pointer for the calling thread.

Returns

The thread instance pointer for the thread from which this function is called.

33.51.4.13 void systhread_setpriority (t_systhread *thread*, int *priority*)

Set the thread priority for the given thread.

Parameters

thread The thread for which to set the priority.

priority A value in the range -32 to 32 where -32 is lowest, 0 is default, and 32 is highest.

33.51.4.14 void systhread_sleep (long *milliseconds*)

Suspend the execution of the calling thread.

Parameters

milliseconds The number of milliseconds to suspend the execution of the calling thread. The actual amount of time may be longer depending on various factors.

33.51.4.15 long systhread_terminate (t_systhread *thread*)

Forcefully kill a thread -- not recommended.

Parameters

thread The thread to kill.

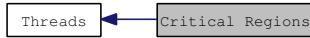
Returns

A Max error code as defined in [e_max_errorcodes](#).

33.52 Critical Regions

A critical region is a simple mechanism that prevents multiple threads from accessing at once code protected by the same critical region.

Collaboration diagram for Critical Regions:



Typedefs

- `typedef void t_critical`

A Max critical region.

Functions

- `void critical_new (t_critical *x)`

Create a new critical region.

- `void critical_enter (t_critical x)`

Enter a critical region.

- `void critical_exit (t_critical x)`

Leave a critical region.

- `void critical_free (t_critical x)`

Free a critical region created with `critical_new()`.

- `short critical_tryenter (t_critical x)`

Try to enter a critical region if it is not locked.

33.52.1 Detailed Description

A critical region is a simple mechanism that prevents multiple threads from accessing at once code protected by the same critical region. The code fragments could be different, and in completely different modules, but as long as the critical region is the same, no two threads should call the protected code at the same time. If one thread is inside a critical region, and another thread wants to execute code protected by the same critical region, the second thread must wait for the first thread to exit the critical region. In some implementations a critical region can be set so that if it takes too long for the first thread to exit said critical region, the second thread is allowed to execute, dangerously and potentially causing crashes. This is the case for the critical regions exposed by Max and the default upper limit for a given thread to remain inside a critical region is two seconds. Despite the fact that there are two seconds of leeway provided before two threads can dangerously enter a critical region, it is important to only protect as small a portion of code as necessary with a critical region.

Under Max 4.1 and earlier there was a simple protective mechanism called "lockout" that would prevent the scheduler from interrupting the low priority thread during sensitive operations such as sending data out

an outlet or modifying members of a linked list. This lockout mechanism has been deprecated, and under the Mac OS X and Windows XP versions (Max 4.2 and later) does nothing. So how do you protect thread sensitive operations? Use critical regions (also known as critical sections). However, it is very important to mention that all outlet calls are now thread safe and should never be contained inside a critical region. Otherwise, this could result in serious timing problems. For other tasks which are not thread safe, such as accessing a linked list, critical regions or some other thread protection mechanism are appropriate.

In Max, the `critical_enter()` function is used to enter a critical region, and the `critical_exit()` function is used to exit a critical region. It is important that in any function which uses critical regions, all control paths protected by the critical region, exit the critical region (watch out for goto or return statements). The `critical_enter()` and `critical_exit()` functions take a critical region as an argument. However, for almost all purposes, we recommend using the global critical region in which case this argument is zero. The use of multiple critical regions can cause problems such as deadlock, i.e. when thread #1 is inside critical region A waiting on critical region B, but thread #2 is inside critical region B and is waiting on critical region A. In a flexible programming environment such as Max, deadlock conditions are easier to generate than you might think. So unless you are completely sure of what you are doing, and absolutely need to make use of multiple critical regions to protect your code, we suggest you use the global critical region.

In the following example code we show how one might use critical regions to protect the traversal of a linked list, testing to find the first element whose value is equal to "val". If this code were not protected, another thread which was modifying the linked list could invalidate assumptions in the traversal code.

```
critical_enter(0);
for (p = head; p; p = p->next) {
    if (p->value == val)
        break;
}
critical_exit(0);
return p;
```

And just to illustrate how to ensure a critical region is exited when multiple control paths are protected by a critical region, here's a slight variant.

```
critical_enter(0);
for (p = head; p; p = p->next) {
    if (p->value == val) {
        critical_exit(0);
        return p;
    }
}
critical_exit(0);
return NULL;
```

For more information on multi-threaded programming, hardware interrupts, and related topics, we suggest you perform some research online or read the relevant chapters of "Modern Operating Systems" by Andrew S. Tanenbaum (Prentice Hall). At the time of writing, some relevant chapters from this book are available for download in PDF format on Prentice Hall's web site. See:

http://www.prenhall.com/divisions/esm/app/author_tanenbaum/custom/mos2e/

Look under "sample sections".

33.52.2 Function Documentation

33.52.2.1 void critical_enter (t_critical x)

Enter a critical region.

Typically you will want the argument to be zero to enter the global critical region, although you could pass your own critical created with [critical_new\(\)](#). It is important to try to keep the amount of code in the critical region to a minimum. Exit the critical region with [critical_exit\(\)](#).

Parameters

- x* A pointer to a [t_critical](#) struct, or zero to uses Max's global critical region.

See also

[critical_exit\(\)](#)

33.52.2.2 void critical_exit ([t_critical](#) *x*)

Leave a critical region.

Typically you will want the argument to be zero to exit the global critical region, although, you if you are using your own critical regions you will want to pass the same one that you previously passed to [critical_enter\(\)](#).

Parameters

- x* A pointer to a [t_critical](#) struct, or zero to uses Max's global critical region.

33.52.2.3 void critical_free ([t_critical](#) *x*)

Free a critical region created with [critical_new\(\)](#).

If you created your own critical region, you will need to free it in your object's free method.

Parameters

- x* The [t_critical](#) struct that will be freed.

33.52.2.4 void critical_new ([t_critical](#) * *x*)

Create a new critical region.

Normally, you do not need to create your own critical region, because you can use Max's global critical region. Only use this function (in your object's instance creation method) if you are certain you are not able to use the global critical region.

Parameters

- x* A [t_critical](#) struct will be returned to you via this pointer.

33.52.2.5 short critical_tryenter ([t_critical](#) *x*)

Try to enter a critical region if it is not locked.

Parameters

- x* A pointer to a [t_critical](#) struct, or zero to uses Max's global critical region.

Returns

returns non-zero if there was a problem entering

See also

[critical_enter\(\)](#)

33.53 Mutexes

Collaboration diagram for Mutexes:



Functions

- long `systhread_mutex_new` (`t_systhread_mutex` *`pmutex`, long `flags`)
Create a new mutex, which can be used to place thread locks around critical code.
- long `systhread_mutex_free` (`t_systhread_mutex` `pmutex`)
Free a mutex created with `systhread_mutex_new()`.
- long `systhread_mutex_lock` (`t_systhread_mutex` `pmutex`)
Enter block of locked code code until a `systhread_mutex_unlock()` is reached.
- long `systhread_mutex_unlock` (`t_systhread_mutex` `pmutex`)
Exit a block of code locked with `systhread_mutex_lock()`.
- long `systhread_mutex_trylock` (`t_systhread_mutex` `pmutex`)
Try to enter block of locked code code until a `systhread_mutex_unlock()` is reached.
- long `systhread_mutex_newlock` (`t_systhread_mutex` *`pmutex`, long `flags`)
Convenience utility that combines `systhread_mutex_new()` and `systhread_mutex_lock()`.

33.53.1 Detailed Description

See also

[Critical Regions](#)

33.53.2 Function Documentation

33.53.2.1 long `systhread_mutex_free` (`t_systhread_mutex` `pmutex`)

Free a mutex created with `systhread_mutex_new()`.

Parameters

`pmutex` The mutex instance pointer.

Returns

A Max error code as defined in `e_max_errorcodes`.

33.53.2.2 long systhread_mutex_lock (*t_systhread_mutex pmutex*)

Enter block of locked code code until a [systhread_mutex_unlock\(\)](#) is reached.

It is important to keep the code in this block as small as possible.

Parameters

pmutex The mutex instance pointer.

Returns

A Max error code as defined in [e_max_errorcodes](#).

See also

[systhread_mutex_trylock\(\)](#)

33.53.2.3 long systhread_mutex_new (*t_systhread_mutex * pmutex, long flags*)

Create a new mutex, which can be used to place thread locks around critical code.

The mutex should be freed with [systhread_mutex_free\(\)](#).

Parameters

pmutex The address of a variable to store the mutex pointer.

flags Flags to determine the behaviour of the mutex, as defined in [e_max_systhread_mutex_flags](#).

Returns

A Max error code as defined in [e_max_errorcodes](#).

Remarks

One reason to use [systhread_mutex_new\(\)](#) instead of [Critical Regions](#) is to create non-recursive locks, which are lighter-weight than recursive locks.

33.53.2.4 long systhread_mutex_newlock (*t_systhread_mutex * pmutex, long flags*)

Convenience utility that combines [systhread_mutex_new\(\)](#) and [systhread_mutex_lock\(\)](#).

Parameters

pmutex The address of a variable to store the mutex pointer.

flags Flags to determine the behaviour of the mutex, as defined in [e_max_systhread_mutex_flags](#).

Returns

A Max error code as defined in [e_max_errorcodes](#).

33.53.2.5 long systhread_mutex_trylock (t_systhread_mutex *pmutex*)

Try to enter block of locked code until a [systhread_mutex_unlock\(\)](#) is reached.

If the lock cannot be entered, this function will return non-zero.

Parameters

pmutex The mutex instance pointer.

Returns

Returns non-zero if there was a problem entering.

See also

[systhread_mutex_lock\(\)](#)

33.53.2.6 long systhread_mutex_unlock (t_systhread_mutex *pmutex*)

Exit a block of code locked with [systhread_mutex_lock\(\)](#).

Parameters

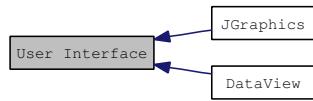
pmutex The mutex instance pointer.

Returns

A Max error code as defined in [e_max_errorcodes](#).

33.54 User Interface

Collaboration diagram for User Interface:



Modules

- [JGraphics](#)

JGraphics is the API for creating user interface objects introduced with Max 5.

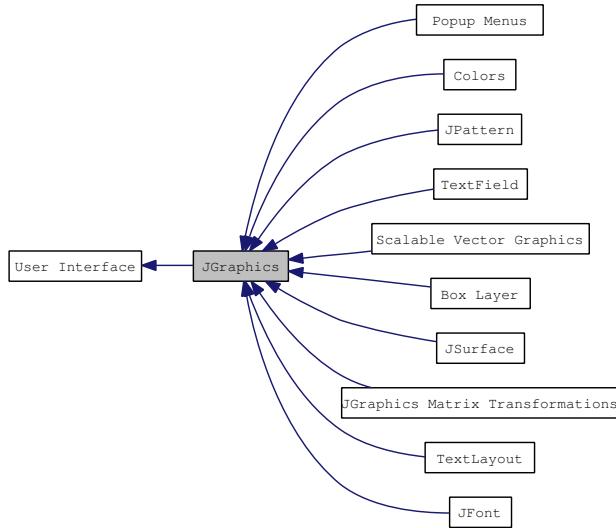
- [DataView](#)

The jdataview object provides a mechanism to display data in a tabular format.

33.55 JGraphics

JGraphics is the API for creating user interface objects introduced with Max 5.

Collaboration diagram for JGraphics:



Data Structures

- struct [t_jgraphics_font_extents](#)

A structure for holding information related to how much space the rendering of a given font will use.

Modules

- [JSurface](#)

A surface is an abstract base class for something you render to.

- [Scalable Vector Graphics](#)
- [JFont](#)
- [JGraphics Matrix Transformations](#)

The [t_jmatrix](#) is one way to represent a transformation.

- [JPatter](#)

A pattern is like a brush that is used to fill a path with.

- [Colors](#)
- [TextField](#)

The textfield is a high-level text display object that may be used by a UI object to represent text in a patcher.

- [TextLayout](#)

A textlayout is lower-level text rendering object used by higher-level entities such as [TextField](#).

- **Popup Menus**

Popup menu API so externals can create popup menus that can also be drawn into.

- **Box Layer**

The boxlayer functions provide way to make it easier to use cached offscreen images (layers) in your drawing.

Defines

- #define **JGRAPHICS_RECT_BOTTOM**(rect) (((rect)->y)+((rect)->height))
Determine the coordinate of the bottom of a rect.
- #define **JGRAPHICS_RECT_RIGHT**(rect) (((rect)->x)+((rect)->width))
Determine the coordinate of the right side of a rect.
- #define **JGRAPHICS_PI** (3.1415926535897932384626433832795028842)
Utility macro to return the value of Pi.
- #define **JGRAPHICS_2PI** (2. * 3.1415926535897932384626433832795028842)
Utility macro to return the value of twice Pi.
- #define **JGRAPHICS_PIOVER2** (0.5 * 3.1415926535897932384626433832795028842)
Utility macro to return the value of half of Pi.
- #define **JGRAPHICS_3PIOVER2** ((3.0 * JGRAPHICS_PI) / 2.0)
Utility macro to return the 270° Case.

TypeDefs

- typedef typedefBEGIN_USING_C_LINKAGE struct _jgraphics **t_jgraphics**
An instance of a jgraphics drawing context.
- typedef struct _jpath **t_jpath**
An instance of a jgraphics path.
- typedef struct _jtextlayout **t_jtextlayout**
An instance of a jgraphics text layout object.
- typedef struct _jtransform **t_jtransform**
An instance of a jgraphics transform.
- typedef struct _jsvg **t_jsvg**
An instance of an SVG object.
- typedef struct _jpopupmenu **t_jpopupmenu**
An instance of a pop-up menu.

Enumerations

- enum `t_jgraphics_format` {

 `JGRAPHICS_FORMAT_ARGB32`,

 `JGRAPHICS_FORMAT_RGB24`,

 `JGRAPHICS_FORMAT_A8` }

Enumeration of color formats used by jgraphics surfaces.

- enum `t_jgraphics_fileformat` {

 `JGRAPHICS_FILEFORMAT_PNG`,

 `JGRAPHICS_FILEFORMAT_JPEG` }

Enumeration of file formats usable for jgraphics surfaces.

- enum `t_jgraphics_text_justification` {

 `JGRAPHICS_TEXT JUSTIFICATION LEFT` = 1,

 `JGRAPHICS_TEXT JUSTIFICATION RIGHT` = 2,

 `JGRAPHICS_TEXT JUSTIFICATION HCENTERED` = 4,

 `JGRAPHICS_TEXT JUSTIFICATION TOP` = 8,

 `JGRAPHICS_TEXT JUSTIFICATION BOTTOM` = 16,

 `JGRAPHICS_TEXT JUSTIFICATION VCENTERED` = 32,

 `JGRAPHICS_TEXT JUSTIFICATION HJUSTIFIED` = 64,

 `JGRAPHICS_TEXT JUSTIFICATION CENTERED` = `JGRAPHICS_TEXT JUSTIFICATION_-HCENTERED + JGRAPHICS_TEXT JUSTIFICATION VCENTERED` }

Enumeration of text justification options, which are specified as a bitmask.

Functions

- int `jgraphics_round` (double d)

Utility for rounding a double to an int.
- `t_jgraphics * jgraphics_reference (t_jgraphics *g)`

Get a reference to a graphics context.
- void `jgraphics_destroy (t_jgraphics *g)`

Release or free a graphics context.
- void `jgraphics_new_path (t_jgraphics *g)`

Begin a new path.
- `t_jpath * jgraphics_copy_path (t_jgraphics *g)`

Get a copy of the current path from a context.
- void `jgraphics_path_destroy (t_jpath *path)`

Release/free a path.
- void `jgraphics_append_path (t_jgraphics *g, t_jpath *path)`

Add a path to a graphics context.

- **void jgraphics_close_path (t_jgraphics *g)**
Close the current path in a context.
- **void jgraphics_path_roundcorners (t_jgraphics *g, double cornerRadius)**
Round out any corners in a path.
- **void jgraphics_get_current_point (t_jgraphics *g, double *x, double *y)**
Get the current location of the cursor in a graphics context.
- **void jgraphics_arc (t_jgraphics *g, double xc, double yc, double radius, double angle1, double angle2)**
Add a circular, clockwise, arc to the current path.
- **void jgraphics_ovalarc (t_jgraphics *g, double xc, double yc, double radiusx, double radiusy, double angle1, double angle2)**
Add a non-circular arc to the current path.
- **void jgraphics_arc_negative (t_jgraphics *g, double xc, double yc, double radius, double angle1, double angle2)**
Add a circular, counter-clockwise, arc to the current path.
- **void jgraphics_curve_to (t_jgraphics *g, double x1, double y1, double x2, double y2, double x3, double y3)**
Add a cubic Bezier spline to the current path.
- **void jgraphics_rel_curve_to (t_jgraphics *g, double x1, double y1, double x2, double y2, double x3, double y3)**
Add a cubic Bezier spline to the current path, using coordinates relative to the current point.
- **void jgraphics_line_to (t_jgraphics *g, double x, double y)**
Add a line segment to the current path.
- **void jgraphics_rel_line_to (t_jgraphics *g, double x, double y)**
Add a line segment to the current path, using coordinates relative to the current point.
- **void jgraphics_move_to (t_jgraphics *g, double x, double y)**
Move the cursor to a new point and begin a new subpath.
- **void jgraphics_rel_move_to (t_jgraphics *g, double x, double y)**
Move the cursor to a new point and begin a new subpath, using coordinates relative to the current point.
- **void jgraphics_rectangle (t_jgraphics *g, double x, double y, double width, double height)**
Add a closed rectangle path in the context.
- **void jgraphics_oval (t_jgraphics *g, double x, double y, double width, double height)**
Deprecated -- do not use.
- **void jgraphics_rectangle_rounded (t_jgraphics *g, double x, double y, double width, double height, double ovalwidth, double ovalheight)**

Add a closed rounded-rectangle path in the context.

- void **jgraphics_ellipse** (**t_jgraphics** *g, double x, double y, double width, double height)
Add a closed elliptical path in the context.
- void **jgraphics_select_font_face** (**t_jgraphics** *g, const char *family, **t_jgraphics_font_slant** slant, **t_jgraphics_font_weight** weight)
Specify a font for a graphics context.
- void **jgraphics_select_jfont** (**t_jgraphics** *g, **t_jfont** *jfont)
*Specify a font for a graphics context by passing a **t_jfont** object.*
- void **jgraphics_set_font_size** (**t_jgraphics** *g, double size)
Specify the font size for a context.
- void **jgraphics_set_underline** (**t_jgraphics** *g, char underline)
Turn underlining on/off for text in a context.
- void **jgraphics_show_text** (**t_jgraphics** *g, const char *utf8)
Display text at the current position in a context.
- void **jgraphics_font_extents** (**t_jgraphics** *g, **t_jgraphics_font_extents** *extents)
Return the extents of the currently selected font for a given graphics context.
- void **jgraphics_text_measure** (**t_jgraphics** *g, const char *utf8, double *width, double *height)
Return the height and width of a string given current graphics settings in a context.
- void **jgraphics_text_measure_wrapped** (**t_jgraphics** *g, const char *utf8, double wrapwidth, long includewhitespace, double *width, double *height, long *numlines)
Return the height, width, and number of lines that will be used to render a given string.
- long **jgraphics_system_canantialiastexttotransparentbg** ()
Determine if you can anti-alias text to a transparent background.
- void **jgraphics_user_to_device** (**t_jgraphics** *g, double *x, double *y)
*User coordinates are those passed to drawing functions in a given **t_jgraphics** context.*
- void **jgraphics_device_to_user** (**t_jgraphics** *g, double *x, double *y)
*User coordinates are those passed to drawing functions in a given **t_jgraphics** context.*
- void **jgraphics_getfiletypes** (void *dummy, long *count, long **filetypes, char *alloc)
Get a list of filetypes appropriate for use with jgraphics surfaces.
- long **jgraphics_rectintersectsrect** (**t_rect** *r1, **t_rect** *r2)
Simple utility to test for rectangle intersection.
- long **jgraphics_rectcontainsrect** (**t_rect** *outer, **t_rect** *inner)
Simple utility to test for rectangle containment.
- void **jgraphics_position_one_rect_near_another_rect_but_keep_inside_a_third_rect** (**t_rect** *positioned_rect, const **t_rect** *positioned_near_this_rect, const **t_rect** *keep_inside_this_rect)

Generate a `t_rect` according to positioning rules.

33.55.1 Detailed Description

JGraphics is the API for creating user interface objects introduced with Max 5. It includes functions for drawing vector-based shapes, managing pop-up menus, rendering text, and importing graphics resources. The API design is inspired by and analogous to the `Cairo` API, though the underlying implementation is actually drawn using `JUCE` (JUCE functions, however, cannot be called directly).

33.55.2 Define Documentation

33.55.2.1 `#define JGRAPHICS_2PI (2. * 3.1415926535897932384626433832795028842)`

Utility macro to return the value of twice Pi.

33.55.2.2 `#define JGRAPHICS_3PIOVER2 ((3.0 * JGRAPHICS_PI) / 2.0)`

Utility macro to return the 270° Case.

33.55.2.3 `#define JGRAPHICS_PI (3.1415926535897932384626433832795028842)`

Utility macro to return the value of Pi.

33.55.2.4 `#define JGRAPHICS_PIOVER2 (0.5 * 3.1415926535897932384626433832795028842)`

Utility macro to return the value of half of Pi.

33.55.3 Enumeration Type Documentation

33.55.3.1 `enum t_jgraphics_fileformat`

Enumeration of file formats usable for jgraphics surfaces.

Enumerator:

`JGRAPHICS_FILEFORMAT_PNG` Portable Network Graphics (PNG) format.

`JGRAPHICS_FILEFORMAT_JPEG` JPEG format.

33.55.3.2 `enum t_jgraphics_format`

Enumeration of color formats used by jgraphics surfaces.

Enumerator:

`JGRAPHICS_FORMAT_ARGB32` Color is represented using 32 bits, 8 bits each for the components, and including an alpha component.

JGRAPHICS_FORMAT_RGB24 Color is represented using 32 bits, 8 bits each for the components.
There is no alpha component.

JGRAPHICS_FORMAT_A8 The color is represented only as an 8-bit alpha mask.

33.55.3.3 enum t_jgraphics_text_justification

Enumeration of text justification options, which are specified as a bitmask.

Enumerator:

JGRAPHICS_TEXT_JUSTIFICATION_LEFT Justify left.
JGRAPHICS_TEXT_JUSTIFICATION_RIGHT Justify right.
JGRAPHICS_TEXT_JUSTIFICATION_HCENTERED Centered horizontally.
JGRAPHICS_TEXT_JUSTIFICATION_TOP Justified to the top.
JGRAPHICS_TEXT_JUSTIFICATION_BOTTOM Justified to the bottom.
JGRAPHICS_TEXT_JUSTIFICATION_VCENTERED Centered vertically.
JGRAPHICS_TEXT_JUSTIFICATION_HJUSTIFIED Horizontally justified.
JGRAPHICS_TEXT_JUSTIFICATION_CENTERED Shortcut for Centering both vertically and horizontally.

33.55.4 Function Documentation

33.55.4.1 void jgraphics_append_path (t_jgraphics *g, t_jpath *path)

Add a path to a graphics context.

Parameters

g The graphics context.
path The path to add.

33.55.4.2 void jgraphics_arc (t_jgraphics *g, double xc, double yc, double radius, double angle1, double angle2)

Add a circular, clockwise, arc to the current path.

Parameters

g The graphics context.
xc The horizontal coordinate of the arc's center.
yc The vertical coordinate of the arc's center.
radius The radius of the arc.
angle1 The starting angle of the arc in radians. Zero radians is center right (positive x axis).
angle2 The terminal angle of the arc in radians. Zero radians is center right (positive x axis).

33.55.4.3 void jgraphics_arc_negative (t_jgraphics *g, double xc, double yc, double radius, double angle1, double angle2)

Add a circular, counter-clockwise, arc to the current path.

Parameters

g The graphics context.

xc The horizontal coordinate of the arc's center.

yc The vertical coordinate of the arc's center.

radius The radius of the arc.

angle1 The starting angle of the arc in radians. Zero radians is center right (positive x axis).

angle2 The terminal angle of the arc in radians. Zero radians is center right (positive x axis).

33.55.4.4 void jgraphics_close_path (t_jgraphics *g)

Close the current path in a context.

This will add a line segment to close current subpath.

Parameters

g The graphics context.

33.55.4.5 t_jpath* jgraphics_copy_path (t_jgraphics *g)

Get a copy of the current path from a context.

Parameters

g A copy of the current path.

33.55.4.6 void jgraphics_curve_to (t_jgraphics *g, double x1, double y1, double x2, double y2, double x3, double y3)

Add a cubic Bezier spline to the current path.

Parameters

g The graphics context.

x1 The first control point.

y1 The first control point.

x2 The second control point.

y2 The second control point.

x3 The destination point.

y3 The destination point.

33.55.4.7 void jgraphics_destroy (*t_jgraphics* **g*)

Release or free a graphics context.

Parameters

g The context to release.

33.55.4.8 void jgraphics_device_to_user (*t_jgraphics* **g*, *double* **x*, *double* **y*)

User coordinates are those passed to drawing functions in a given *t_jgraphics* context.

Device coordinates refer to patcher canvas coordinates, before any zooming.

33.55.4.9 void jgraphics_ellipse (*t_jgraphics* **g*, *double* *x*, *double* *y*, *double* *width*, *double* *height*)

Add a closed elliptical path in the context.

Parameters

g The graphics context.

x The horizontal origin.

y The vertical origin.

width The width of the rect.

height The height of the rect.

33.55.4.10 void jgraphics_font_extents (*t_jgraphics* **g*, *t_jgraphics_font_extents* **extents*)

Return the extents of the currently selected font for a given graphics context.

Parameters

g Pointer to a jgraphics context.

extents The address of a *t_jgraphics_font_extents* structure to be filled with the results.

33.55.4.11 void jgraphics_get_current_point (*t_jgraphics* **g*, *double* **x*, *double* **y*)

Get the current location of the cursor in a graphics context.

Parameters

g The graphics context.

x The address of a variable that will be set to the horizontal cursor location upon return.

y The address of a variable that will be set to the vertical cursor location upon return.

33.55.4.12 void jgraphics_getfiletypes (void * *dummy*, long * *count*, long ***filetypes*, char * *alloc*)

Get a list of of filetypes appropriate for use with jgraphics surfaces.

Parameters

- dummy* Unused.
- count* The address of a variable to be set with the number of types in filetypes upon return.
- filetypes* The address of a variable that will represent the array of file types upon return.
- alloc* The address of a char that will be flagged with a 1 or a 0 depending on whether or not memory was allocated for the filetypes member.

Remarks

This example shows a common usage of [jgraphics_getfiletypes\(\)](#).

```
char      filename[MAX_PATH_CHARS];
long      *type = NULL;
long      ntype;
long      outtype;
t_max_err err;
char      alloc;
short     path;
t_jsurface *surface;

if (want_to_show_dialog) {
    jgraphics_getfiletypes(x, &ntype, &type, &alloc);
    err = open_dialog(filename, &path, (void *)&outtype, (void *)type, ntype);
    if (err)
        goto out;
}
else {
    strncpy_zero(filename, s->s_name, MAX_PATH_CHARS);
    err = locatefile_extended(filename, &path, &outtype, type, ntype);
    if (err)
        goto out;
}
surface = jgraphics_image_surface_create_referenced(filename, path);
out:
if (alloc)
    sysmem_freet((char *)type);
```

33.55.4.13 void jgraphics_line_to (t_jgraphics * *g*, double *x*, double *y*)

Add a line segment to the current path.

Parameters

- g* The graphics context.
- x* The destination point.
- y* The destination point.

33.55.4.14 void jgraphics_move_to (t_jgraphics * *g*, double *x*, double *y*)

Move the cursor to a new point and begin a new subpath.

Parameters

- g* The graphics context.
- x* The new location.
- y* The new location.

33.55.4.15 void jgraphics_new_path (t_jgraphics **g*)

Begin a new path.

This action clears any current path in the context.

Parameters

- g* The graphics context.

33.55.4.16 void jgraphics_oval (t_jgraphics **g*, double *x*, double *y*, double *width*, double *height*)

Deprecated -- do not use.

Adds a closed oval path in the context, however, it does not scale appropriately.

Parameters

- g* The graphics context.
- x* The horizontal origin.
- y* The vertical origin.
- width* The width of the oval.
- height* The height of the oval.

33.55.4.17 void jgraphics_ovalarc (t_jgraphics **g*, double *xc*, double *yc*, double *radiusx*, double *radiusy*, double *angle1*, double *angle2*)

Add a non-circular arc to the current path.

Parameters

- g* The graphics context.
- xc* The horizontal coordinate of the arc's center.
- yc* The vertical coordinate of the arc's center.
- radiusx* The horizontal radius of the arc.
- radiusy* The vertical radius of the arc.
- angle1* The starting angle of the arc in radians. Zero radians is center right (positive x axis).
- angle2* The terminal angle of the arc in radians. Zero radians is center right (positive x axis).

33.55.4.18 void jgraphics_path_destroy (*t_jpath* **path*)

Release/free a path.

Parameters

path The path to release.

33.55.4.19 void jgraphics_path_roundcorners (*t_jgraphics* **g*, double *cornerRadius*)

Round out any corners in a path.

This action clears any current path in the context.

Parameters

g The graphics context.

cornerRadius The amount by which to round corners.

33.55.4.20 void jgraphics_position_one_rect_near_another_rect_but_keep_inside_a_third_rect (*t_rect* **positioned_rect*, const *t_rect* **positioned_near_this_rect*, const *t_rect* **keep_inside_this_rect*)

Generate a *t_rect* according to positioning rules.

Parameters

positioned_rect The address of a valid *t_rect* whose members will be filled in upon return.

positioned_near_this_rect A pointer to a rect near which this rect should be positioned.

keep_inside_this_rect A pointer to a rect defining the limits within which the new rect must reside.

33.55.4.21 void jgraphics_rectangle (*t_jgraphics* **g*, double *x*, double *y*, double *width*, double *height*)

Add a closed rectangle path in the context.

Parameters

g The graphics context.

x The horizontal origin.

y The vertical origin.

width The width of the rect.

height The height of the rect.

33.55.4.22 void jgraphics_rectangle_rounded (t_jgraphics *g, double x, double y, double width, double height, double ovalwidth, double ovalheight)

Add a closed rounded-rectangle path in the context.

Parameters

- g* The graphics context.
- x* The horizontal origin.
- y* The vertical origin.
- width* The width of the rect.
- height* The height of the rect.
- ovalwidth* The width of the oval used for the round corners.
- ovalheight* The height of the oval used for the round corners.

33.55.4.23 long jgraphics_rectcontainsrect (t_rect *outer, t_rect *inner)

Simple utility to test for rectangle containment.

Parameters

- outer* The address of the first rect for the test.
- inner* The address of the second rect for the test.

Returns

Returns true if the inner rect is completely inside the outer rect, otherwise false.

33.55.4.24 long jgraphics_rectintersectsrect (t_rect *r1, t_rect *r2)

Simple utility to test for rectangle intersection.

Parameters

- r1* The address of the first rect for the test.
- r2* The address of the second rect for the test.

Returns

Returns true if the rects intersect, otherwise false.

33.55.4.25 t_jgraphics* jgraphics_reference (t_jgraphics *g)

Get a reference to a graphics context.

When you are done you should release your reference with [jgraphics_destroy\(\)](#).

Parameters

- g* The context you wish to reference.

Returns

A new reference to the context.

33.55.4.26 void jgraphics_rel_curve_to (t_jgraphics *g, double x1, double y1, double x2, double y2, double x3, double y3)

Add a cubic Bezier spline to the current path, using coordinates relative to the current point.

Parameters

- g* The graphics context.
- x1* The first control point.
- y1* The first control point.
- x2* The second control point.
- y2* The second control point.
- x3* The destination point.
- y3* The destination point.

33.55.4.27 void jgraphics_rel_line_to (t_jgraphics *g, double x, double y)

Add a line segment to the current path, using coordinates relative to the current point.

Parameters

- g* The graphics context.
- x* The destination point.
- y* The destination point.

33.55.4.28 void jgraphics_rel_move_to (t_jgraphics *g, double x, double y)

Move the cursor to a new point and begin a new subpath, using coordinates relative to the current point.

Parameters

- g* The graphics context.
- x* The new location.
- y* The new location.

33.55.4.29 int jgraphics_round (double d)

Utility for rounding a double to an int.

Parameters

- d* floating-point input.

Returns

rounded int output.

33.55.4.30 void jgraphics_select_font_face (t_jgraphics **g*, const char **family*, t_jgraphics_font_slant *slant*, t_jgraphics_font_weight *weight*)

Specify a font for a graphics context.

Parameters

- g* The graphics context.
- family* The name of the font family (e.g. "Arial").
- slant* Define the slant to use for the font.
- weight* Define the weight to use for the font.

33.55.4.31 void jgraphics_select_jfont (t_jgraphics **g*, t_jfont **jfont*)

Specify a font for a graphics context by passing a [t_jfont](#) object.

Parameters

- g* The graphics context.
- jfont* A jfont object whose attributes will be copied to the context.

33.55.4.32 void jgraphics_set_font_size (t_jgraphics **g*, double *size*)

Specify the font size for a context.

Parameters

- g* The graphics context.
- size* The font size.

33.55.4.33 void jgraphics_set_underline (t_jgraphics **g*, char *underline*)

Turn underlining on/off for text in a context.

Parameters

- g* The graphics context.
- underline* Pass true or false to set the appropriate behavior.

33.55.4.34 void jgraphics_show_text (t_jgraphics **g*, const char **utf8*)

Display text at the current position in a context.

Parameters

- g* The graphics context.
- utf8* The text to display.

33.55.4.35 long jgraphics_system_canantialiastexttotransparentbg ()

Determine if you can anti-alias text to a transparent background.

You might want to call this and then disable "useimagebuffer" if false *and* you are rendering text on a transparent background.

Returns

Non-zero if you can anti-alias text to a transparent background.

33.55.4.36 void jgraphics_text_measure (t_jgraphics *g, const char *utf8, double *width, double *height)

Return the height and width of a string given current graphics settings in a context.

Parameters

g Pointer to a jgraphics context.

utf8 A string containing the text whose dimensions we wish to find.

width The address of a variable to be filled with the width of the rendered text.

height The address of a variable to be filled with the height of the rendered text.

33.55.4.37 void jgraphics_text_measure_wrapped (t_jgraphics *g, const char *utf8, double wrapwidth, long includewhitespace, double *width, double *height, long *numlines)

Return the height, width, and number of lines that will be used to render a given string.

Parameters

g Pointer to a jgraphics context.

utf8 A string containing the text whose dimensions we wish to find.

wrapwidth The number of pixels in width at which the text should be wrapped if it is too long.

includewhitespace Set zero to not include white space in the calculation, otherwise set this parameter to 1.

width The address of a variable to be filled with the width of the rendered text.

height The address of a variable to be filled with the height of the rendered text.

numlines The address of a variable to be filled with the number of lines required to render the text.

33.55.4.38 void jgraphics_user_to_device (t_jgraphics *g, double *x, double *y)

User coordinates are those passed to drawing functions in a given [t_jgraphics](#) context.

Device coordinates refer to patcher canvas coordinates, before any zooming.

33.56 JSurface

A surface is an abstract base class for something you render to.

Collaboration diagram for JSurface:



Typedefs

- `t_jsurface * jgraphics_image_surface_create (t_jgraphics_format format, int width, int height)`

An instance of a jgraphics surface.

Functions

- `t_jsurface * jgraphics_image_surface_create (t_jgraphics_format format, int width, int height)`
Create an image surface.
- `t_jsurface * jgraphics_image_surface_create_referenced (const char *filename, short path)`
Create an image surface, filling it with the contents of a file, and get a reference to the surface.
- `t_jsurface * jgraphics_image_surface_create_from_file (const char *filename, short path)`
Create an image surface, filling it with the contents of a file.
- `t_jsurface * jgraphics_image_surface_create_for_data (unsigned char *data, t_jgraphics_format format, int width, int height, int stride, method freefun, void *freearg)`
Create an image surface from given pixel data.
- `t_jsurface * jgraphics_image_surface_create_from_filedata (const void *data, unsigned long datalen)`
Create a new surface from file data.
- `t_jsurface * jgraphics_image_surface_create_from_resource (const void *moduleRef, const char *resname)`
Create a new surface from a resource in your external.
- `t_max_err jgraphics_get_resource_data (const void *moduleRef, const char *resname, long extcount, t_atom *exts, void **data, unsigned long *datasize)`
Low-level routine to access an object's resource data.
- `t_jsurface * jgraphics_surface_reference (t_jsurface *s)`
Create a reference to an existing surface.
- `void jgraphics_surface_destroy (t_jsurface *s)`
Release or free a surface.
- `t_max_err jgraphics_image_surface_writepng (t_jsurface *surface, const char *filename, short path, long dpi)`

Export a PNG file of the contents of a surface.

- int [jgraphics_image_surface_get_width \(t_jsurface *s\)](#)
Retrieve the width of a surface.
- int [jgraphics_image_surface_get_height \(t_jsurface *s\)](#)
Retrieve the height of a surface.
- void [jgraphics_image_surface_set_pixel \(t_jsurface *s, int x, int y, t_jrgba color\)](#)
Set the color of an individual pixel in a surface.
- void [jgraphics_image_surface_get_pixel \(t_jsurface *s, int x, int y, t_jrgba *color\)](#)
Retrieve the color of an individual pixel in a surface.
- void [jgraphics_image_surface_scroll \(t_jsurface *s, int x, int y, int width, int height, int dx, int dy, t_jpath **path\)](#)
- void [jgraphics_image_surface_draw \(t_jgraphics *g, t_jsurface *s, t_rect srcRect, t_rect destRect\)](#)
Draw an image surface.
- void [jgraphics_image_surface_draw_fast \(t_jgraphics *g, t_jsurface *s\)](#)
Draw an image surface quickly.
- void [jgraphics_write_image_surface_to_filedata \(t_jsurface *surf, long fmt, void **data, long *size\)](#)
Get surface data ready for manually writing to a file.
- void [jgraphics_image_surface_clear \(t_jsurface *s, int x, int y, int width, int height\)](#)
Set all pixels in rect to 0.
- [t_jgraphics * jgraphics_create \(t_jsurface *target\)](#)
Create a context to draw on a particular surface.

33.56.1 Detailed Description

A surface is an abstract base class for something you render to. An image surface is a concrete instance that renders to an image in memory, essentially an offscreen bitmap.

33.56.2 Function Documentation

33.56.2.1 [t_jgraphics* jgraphics_create \(t_jsurface * target\)](#)

Create a context to draw on a particular surface.

When you are done, call [jgraphics_destroy\(\)](#).

Parameters

target The surface to which to draw.

Returns

The new graphics context.

33.56.2.2 `t_max_err jgraphics_get_resource_data (const void * moduleRef, const char * resname, long extcount, t_atom * exts, void ** data, unsigned long * datasize)`

Low-level routine to access an object's resource data.

Parameters

moduleRef A pointer to your external's module, which is passed to your external's main() function when the class is loaded.

resname Base name of the resource data (without an extension)

extcount Count of possible extensions (ignored on Windows)

exts Array of symbol atoms containing possible filename extensions (ignored on Windows)

data Returned resource data assigned to a pointer you supply

datasize Size of the data returned

Remarks

You are responsible for freeing any data returned in the data pointer

Returns

A Max error code.

33.56.2.3 `void jgraphics_image_surface_clear (t_jsurface * s, int x, int y, int width, int height)`

Set all pixels in rect to 0.

Parameters

s The surface to clear.

x The horizontal origin of the rect to clear.

y The vertical origin of the rect to clear.

width The width of the rect to clear.

height The height of the rect to clear.

33.56.2.4 `t_jsurface* jgraphics_image_surface_create (t_jgraphics_format format, int width, int height)`

Create an image surface.

Use [jgraphics_surface_destroy\(\)](#) to free it when you are done.

Parameters

format Defines the color format for the new surface.

width Defines the width of the new surface.

height Defines the height of the new surface.

Returns

A pointer to the new surface.

33.56.2.5 `t_jsurface* jgraphics_image_surface_create_for_data (unsigned char * data, t_jgraphics_format format, int width, int height, int stride, method freefun, void * freearg)`

Create an image surface from given pixel data.

Data should point to start of top line of bitmap, stride tells how to get to next line. For upside down windows bitmaps, data = (pBits-(height-1)*stride) and stride is a negative number.

Parameters

data The data. For example, an RGBA image loaded in memory.
format The format of the data.
width The width of the new surface.
height The height of the new surface.
stride The number of bytes between the start of rows in the dat buffer.
freefun If not NULL, freefun will be called when the surface is destroyed
freearg This will be passed to freefun if/when freefun is called.

Returns

A pointer to the new surface.

33.56.2.6 `t_jsurface* jgraphics_image_surface_create_from_file (const char * filename, short path)`

Create an image surface, filling it with the contents of a file.

Use [jgraphics_surface_destroy\(\)](#) to free it when you are done.

Parameters

filename The name of the file.
path The path id of the file.

Returns

A pointer to the new surface.

33.56.2.7 `t_jsurface* jgraphics_image_surface_create_from_filedata (const void * data, unsigned long datalen)`

Create a new surface from file data.

Parameters

data A pointer to the raw PNG or JPG bits.
datalen The number of bytes in data.

Returns

The new surface.

See also

[jgraphics_write_image_surface_to_filedata\(\)](#)

33.56.2.8 `t_jsurface* jgraphics_image_surface_create_from_resource (const void * moduleRef, const char * resname)`

Create a new surface from a resource in your external.

Parameters

moduleRef A pointer to your external's module, which is passed to your external's main() function when the class is loaded.

resname The name of the resource in the external.

Remarks

The following example shows an example of how this might be used in an external.

```
static s_my_surface = NULL;

int main(void *moduleRef)
{
    // (Do typical class initialization here)

    // now create the surface from a resource that we added to the Xcode/Visual
    // Studio project
    s_my_surface = jgraphics_image_surface_create_from_resource(moduleRef, "myC
oolImage");

    return 0;
}
```

33.56.2.9 `t_jsurface* jgraphics_image_surface_create_referenced (const char *filename, short path)`

Create an image surface, filling it with the contents of a file, and get a reference to the surface.

Use [jgraphics_surface_destroy\(\)](#) to release your reference to the surface when you are done.

Parameters

filename The name of the file.

path The path id of the file.

Returns

A pointer to the new surface.

33.56.2.10 `void jgraphics_image_surface_draw (t_jgraphics *g, t_jsurface *s, t_rect srcRect, t_rect destRect)`

Draw an image surface.

This not in cairo, but, it seems silly to have to make a brush to just draw an image. This doesn't support rotations, however.

Parameters

g The graphics context in which to draw the surface.

s The surface to draw.

srcRect The rect within the surface that should be drawn.

destRect The rect in the context to which to draw the srcRect.

See also

[jgraphics_image_surface_draw_fast\(\)](#)

33.56.2.11 void jgraphics_image_surface_draw_fast (*t_jgraphics *g, t_jsurface *s*)

Draw an image surface quickly.

The draw_fast version won't scale based on zoom factor or user transforms so make sure that this is what you want! Draws entire image, origin *can* be shifted via zoom and user transforms (even though image is not scaled based on those same transforms)

Parameters

g The graphics context in which to draw the surface.

s The surface to draw.

See also

[jgraphics_image_surface_draw](#)

33.56.2.12 int jgraphics_image_surface_get_height (*t_jsurface *s*)

Retrieve the height of a surface.

Parameters

s The surface to query.

Returns

The height of the surface.

33.56.2.13 void jgraphics_image_surface_get_pixel (*t_jsurface *s, int x, int y, t_jrgba *color*)

Retrieve the color of an individual pixel in a surface.

Parameters

s The surface.

x The horizontal coordinate of the pixel.

y The vertical coordinate of the pixel.

color The address of a valid [t_jrgba](#) struct whose values will be filled in with the color of the pixel upon return.

33.56.2.14 int jgraphics_image_surface_get_width (t_jsurface * s)

Retrieve the width of a surface.

Parameters

s The surface to query.

Returns

The width of the surface.

33.56.2.15 void jgraphics_image_surface_scroll (t_jsurface * s, int x, int y, int width, int height, int dx, int dy, t_jpath ** path)**Parameters**

s The surface to scroll.

x The origin of the rect to scroll.

y The origin of the rect to scroll.

width The width of the rect to scroll.

height The height of the rect to scroll.

dx The amount to scroll the surface horizontally.

dy The amount to scroll the surface vertically.

path Can pass NULL if you are not interested in this info. Otherwise pass a pointer and it will be returned with a path containing the invalid region.

33.56.2.16 void jgraphics_image_surface_set_pixel (t_jsurface * s, int x, int y, t_jrgba color)

Set the color of an individual pixel in a surface.

Parameters

s The surface.

x The horizontal coordinate of the pixel.

y The vertical coordinate of the pixel.

color The color of the pixel.

33.56.2.17 t_max_err jgraphics_image_surface_writepng (t_jsurface * surface, const char * filename, short path, long dpi)

Export a PNG file of the contents of a surface.

Parameters

surface The surface to export.

filename Specify the name of the file to create.

path Specify the path id for where to create the file.

dpi Define the resolution of the image (e.g. 72).

Returns

A Max error code.

33.56.2.18 void jgraphics_surface_destroy (**t_jsurface** * *s*)

Release or free a surface.

Parameters

s The surface to release.

33.56.2.19 **t_jsurface*** jgraphics_surface_reference (**t_jsurface** * *s*)

Create a reference to an existing surface.

Use [jgraphics_surface_destroy\(\)](#) to release your reference to the surface when you are done.

Parameters

s The surface to reference.

Returns

The new reference to the surface.

33.56.2.20 void jgraphics_write_image_surface_to_filedata (**t_jsurface** * *surf*, **long** *fmt*, **void** ** *data*, **long** * *size*)

Get surface data ready for manually writing to a file.

Parameters

surf The surface whose data will be retrieved.

fmt The format for the data. This should be a selection from [t_jgraphics_fileformat](#).

data The address of a pointer that will be allocated and filled. When you are done with this data you should free it using [sysmem_freeptr\(\)](#).

size The address of a variable to hold the size of the data upon return.

Remarks

A good example of this is to embed the surface as a PNG in a patcher file.

```
long size = 0;
void *data = NULL;

jgraphics_write_image_surface_to_filedata(x->j_surface,
JGRAPHICS_FILEFORMAT_PNG, &data, &size);
if (size) {
x->j_format = gensym("png");
binarydata_appendtodictionary(data, size, gensym("data"), x->j_format, d);
x->j_imagedata = data;
x->j_imagedatasize = size;
}
```

See also

[jgraphics_image_surface_create_from_filedata\(\)](#)

33.57 Scalable Vector Graphics

Collaboration diagram for Scalable Vector Graphics:



Functions

- `t_jsvg * jsvg_create_from_file (const char *filename, short path)`
Read an SVG file, return a `t_jsvg` object.
- `t_jsvg * jsvg_create_from_resource (const void *moduleRef, const char *resname)`
Read an SVG file from a resource.
- `t_jsvg * jsvg_create_from_xmlstring (const char *svgXML)`
Create an SVG object from a string containing the SVG's XML.
- `void jsvg_get_size (t_jsvg *svg, double *width, double *height)`
Retrieve the size of an SVG object.
- `void jsvg_destroy (t_jsvg *svg)`
Free a `t_jsvg` object.
- `void jsvg_render (t_jsvg *svg, t_jgraphics *g)`
Render an SVG into a graphics context.

33.57.1 Function Documentation

33.57.1.1 `t_jsvg* jsvg_create_from_file (const char *filename, short path)`

Read an SVG file, return a `t_jsvg` object.

Parameters

- filename* The name of the file to read.
path The path id of the file to read.

Returns

A new SVG object.

33.57.1.2 `t_jsvg* jsvg_create_from_resource (const void *moduleRef, const char *resname)`

Read an SVG file from a resource.

Parameters

- moduleRef* The external's moduleRef.

resname The name of the SVG resource.

Returns

A new SVG object.

See also

[jgraphics_image_surface_create_from_resource\(\)](#)

33.57.1.3 `t_jsvg* jsvg_create_from_xmlstring (const char *svgXML)`

Create an SVG object from a string containing the SVG's XML.

Parameters

svgXML The SVG source.

Returns

A new SVG object.

33.57.1.4 `void jsvg_destroy (t_jsvg *svg)`

Free a `t_jsvg` object.

Parameters

svg The object to free.

33.57.1.5 `void jsvg_get_size (t_jsvg *svg, double *width, double *height)`

Retrieve the size of an SVG object.

Parameters

svg An SVG object.

width The address of a variable that will be set to the width upon return.

height The address of a variable that will be set to the height upon return.

33.57.1.6 `void jsvg_render (t_jsvg *svg, t_jgraphics *g)`

Render an SVG into a graphics context.

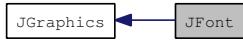
Parameters

svg The SVG object to render.

g The graphics context in which to render.

33.58 JFont

Collaboration diagram for JFont:



Typedefs

- `typedef struct _jfont t_jfont`

An instance of a jgraphics font.

Enumerations

- `enum t_jgraphics_font_slant {
 JGRAPHICS_FONT_SLANT_NORMAL,
 JGRAPHICS_FONT_SLANT_ITALIC }`

Enumeration of slanting options for font display.

- `enum t_jgraphics_font_weight {
 JGRAPHICS_FONT_WEIGHT_NORMAL,
 JGRAPHICS_FONT_WEIGHT_BOLD }`

Enumeration of font weight options for font display.

Functions

- `t_jfont * jfont_create (const char *family, t_jgraphics_font_slant slant, t_jgraphics_font_weight weight, double size)`

Create a new font object.

- `t_jfont * jfont_reference (t_jfont *font)`

Create new reference to an existing font object.

- `void jfont_destroy (t_jfont *font)`

Release or free a font object.

- `void jfont_set_font_size (t_jfont *font, double size)`

Set the size of a font object.

- `void jfont_set_underline (t_jfont *font, char ul)`

Set the underlining of a font object.

- `void jfont_extents (t_jfont *font, t_jgraphics_font_extents *extents)`

Get extents of this font.

- void [jfont_text_measure](#) ([t_jfont](#) *font, const char *utf8, double *width, double *height)
Given a font, find out how much area is required to render a string of text.
- void [jfont_text_measure_wrapped](#) ([t_jfont](#) *font, const char *utf8, double wrapwidth, long includewhitespace, double *width, double *height, long *numlines)
Given a font, find out how much area is required to render a string of text, provided a horizontal maximum limit at which the text is wrapped.
- [t_max_err jfont_getfontlist](#) (long *count, [t_symbol](#) ***list)
Get a list of font names.
- long [jbox_get_font_weight](#) ([t_object](#) *b)
Get the slant box's font.
- long [jbox_get_font_slant](#) ([t_object](#) *b)
Get the slant box's font.

33.58.1 Enumeration Type Documentation

33.58.1.1 enum [t_jgraphics_font_slant](#)

Enumeration of slanting options for font display.

Enumerator:

- JGRAPHICS_FONT_SLANT_NORMAL*** Normal slanting (typically this means no slanting).
JGRAPHICS_FONT_SLANT_ITALIC Italic slanting.

33.58.1.2 enum [t_jgraphics_font_weight](#)

Enumeration of font weight options for font display.

Enumerator:

- JGRAPHICS_FONT_WEIGHT_NORMAL*** Normal font weight.
JGRAPHICS_FONT_WEIGHT_BOLD Bold font weight.

33.58.2 Function Documentation

33.58.2.1 long [jbox_get_font_slant](#) ([t_object](#) * b)

Get the slant box's font.

Parameters

b An object's box.

Returns

A value from the [t_jgraphics_font_slant](#) enum.

33.58.2.2 long jbox_get_font_weight (*t_object *b*)

Get the slant box's font.

Parameters

b An object's box.

Returns

A value from the [t_jgraphics_font_weight](#) enum.

33.58.2.3 t_jfont* jfont_create (const char **family*, *t_jgraphics_font_slant slant*, *t_jgraphics_font_weight weight*, *double size*)

Create a new font object.

Parameters

family The name of the font family (e.g. Arial).

slant The type of slant for the font.

weight The type of weight for the font.

size The size of the font.

Returns

The new font object.

33.58.2.4 void jfont_destroy (*t_jfont *font*)

Release or free a font object.

Parameters

font The font object to release.

33.58.2.5 void jfont_extents (*t_jfont *font*, *t_jgraphics_font_extents *extents*)

Get extents of this font.

Parameters

font The font object.

extents The font extents upon return/

33.58.2.6 `t_max_err jfont_getfontlist (long * count, t_symbol * list)`**

Get a list of font names.

Parameters

count The address of a variable to hold the count of font names in list upon return.

list The address of a `t_symbol**` initialized to NULL. Upon return this will be set to an array of `count` `t_symbol` pointers. This array should be freed using `sysmem_freeptr()` when you are done with it.

Returns

A Max error code.

33.58.2.7 `t_jfont* jfont_reference (t_jfont * font)`

Create new reference to an existing font object.

Parameters

font The font object for which to obtain a reference.

Returns

The new font object reference.

33.58.2.8 `void jfont_set_font_size (t_jfont * font, double size)`

Set the size of a font object.

Parameters

font The font object.

size The new size for the font object.

33.58.2.9 `void jfont_set_underline (t_jfont * font, char ul)`

Set the underlining of a font object.

Parameters

font The font object.

ul Pass true to underline, or false for no underlining.

33.58.2.10 void jfont_text_measure (*t_jfont *font*, *const char *utf8*, *double *width*, *double *height*)

Given a font, find out how much area is required to render a string of text.

Parameters

font The font object.

utf8 The text whose rendering will be measured.

width The address of a variable to hold the width upon return.

height The address of a variable to hold the height upon return.

33.58.2.11 void jfont_text_measure_wrapped (*t_jfont *font*, *const char *utf8*, *double wrapwidth*, *long includewhitespace*, *double *width*, *double *height*, *long *numlines*)

Given a font, find out how much area is required to render a string of text, provided a horizontal maximum limit at which the text is wrapped.

Parameters

font The font object.

utf8 The text whose rendering will be measured.

wrapwidth The maximum width, above which text should wrap onto a new line.

includewhitespace If non-zero, include whitespace in the measurement.

width The address of a variable to hold the width upon return.

height The address of a variable to hold the height upon return.

numlines The address of a variable to hold the number of lines of text after wrapping upon return.

33.59 JGraphics Matrix Transformations

The `t_jmatrix` is one way to represent a transformation.

Collaboration diagram for JGraphics Matrix Transformations:



Data Structures

- struct `t_jmatrix`
An affine transformation (such as scale, shear, etc).

Functions

- void `jgraphics_matrix_init` (`t_jmatrix` *x, double xx, double yx, double xy, double yy, double x0, double y0)
Set a `t_jmatrix` to an affine transformation.
- void `jgraphics_matrix_init_identity` (`t_jmatrix` *x)
Modify a matrix to be an identity transform.
- void `jgraphics_matrix_init_translate` (`t_jmatrix` *x, double tx, double ty)
Initialize a `t_jmatrix` to translate (offset) a point.
- void `jgraphics_matrix_init_scale` (`t_jmatrix` *x, double sx, double sy)
Initialize a `t_jmatrix` to scale (offset) a point.
- void `jgraphics_matrix_init_rotate` (`t_jmatrix` *x, double radians)
Initialize a `t_jmatrix` to rotate (offset) a point.
- void `jgraphics_matrix_translate` (`t_jmatrix` *x, double tx, double ty)
Apply a translation to an existing matrix.
- void `jgraphics_matrix_scale` (`t_jmatrix` *x, double sx, double sy)
Apply a scaling to an existing matrix.
- void `jgraphics_matrix_rotate` (`t_jmatrix` *x, double radians)
Apply a rotation to an existing matrix.
- void `jgraphics_matrix_invert` (`t_jmatrix` *x)
Invert an existing matrix.
- void `jgraphics_matrix_multiply` (`t_jmatrix` *result, const `t_jmatrix` *a, const `t_jmatrix` *b)
Multiply two matrices: resulting matrix has effect of first applying a and then applying b.
- void `jgraphics_matrix_transform_point` (const `t_jmatrix` *matrix, double *x, double *y)
Transform a point using a `t_jmatrix` transformation.

33.59.1 Detailed Description

The `t_jmatrix` is one way to represent a transformation. You can use the `t_jmatrix` in the call to `jgraphics_transform()`, `jgraphics_setmatrix()`, and `jgraphics_pattern_set_matrix` for specifying transformations.

33.59.2 Function Documentation

33.59.2.1 void jgraphics_matrix_init (`t_jmatrix *x, double xx, double yx, double xy, double yy, double x0, double y0`)

Set a `t_jmatrix` to an affine transformation.

Parameters

`x`

`xx`

`yx`

`xy`

`yy`

`x0`

`y0`

Remarks

given `x,y` the matrix specifies the following transformation:

```
xnew = xx * x + xy * y + x0;
ynew = yx * x + yy * y + y0;
```

33.59.2.2 void jgraphics_matrix_init_identity (`t_jmatrix *x`)

Modify a matrix to be an identity transform.

Parameters

`x` The `t_jmatrix`.

33.59.2.3 void jgraphics_matrix_init_rotate (`t_jmatrix *x, double radians`)

Initialize a `t_jmatrix` to rotate (offset) a point.

Parameters

`x` The `t_jmatrix`.

`radians` The angle or rotation in radians.

33.59.2.4 void jgraphics_matrix_init_scale (*t_jmatrix* **x*, double *sx*, double *sy*)

Initialize a [t_jmatrix](#) to scale (offset) a point.

Parameters

- x* The [t_jmatrix](#).
- sx* The horizontal scale factor.
- sy* The vertical scale factor.

33.59.2.5 void jgraphics_matrix_init_translate (*t_jmatrix* **x*, double *tx*, double *ty*)

Initialize a [t_jmatrix](#) to translate (offset) a point.

Parameters

- x* The [t_jmatrix](#).
- tx* The amount of x-axis translation.
- ty* The amount of y-axis translation.

33.59.2.6 void jgraphics_matrix_invert (*t_jmatrix* **x*)

Invert an existing matrix.

Parameters

- x* The [t_jmatrix](#).

33.59.2.7 void jgraphics_matrix_multiply (*t_jmatrix* **result*, const *t_jmatrix* **a*, const *t_jmatrix* **b*)

Multiply two matrices: resulting matrix has effect of first applying a and then applying b.

Parameters

- result* The resulting product [t_jmatrix](#).
- a* The first operand.
- b* The second operand.

33.59.2.8 void jgraphics_matrix_rotate (*t_jmatrix* **x*, double *radians*)

Apply a rotation to an existing matrix.

Parameters

- x* The [t_jmatrix](#).
- radians* The angle or rotation in radians.

33.59.2.9 void jgraphics_matrix_scale (*t_jmatrix* **x*, **double *sx*, **double** *sy*)**

Apply a scaling to an existing matrix.

Parameters

- x* The [t_jmatrix](#).
- sx* The horizontal scale factor.
- sy* The vertical scale factor.

33.59.2.10 void jgraphics_matrix_transform_point (const** *t_jmatrix* **matrix*, **double** **x*, **double** **y*)**

Transform a point using a [t_jmatrix](#) transformation.

Parameters

- matrix* The [t_jmatrix](#).
- x* The address of the variable holding the x coordinate.
- y* The address of the variable holding the y coordinate.

33.59.2.11 void jgraphics_matrix_translate (*t_jmatrix* **x*, **double *tx*, **double** *ty*)**

Apply a translation to an existing matrix.

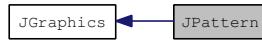
Parameters

- x* The [t_jmatrix](#).
- tx* The amount of x-axis translation.
- ty* The amount of y-axis translation.

33.60 JPattern

A pattern is like a brush that is used to fill a path with.

Collaboration diagram for JPattern:



Typedefs

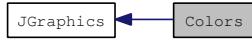
- `typedef struct _jpattern t_jpattern`
An instance of a jgraphics pattern.

33.60.1 Detailed Description

A pattern is like a brush that is used to fill a path with. It could be a solid color but it could also be an image. You can draw to a surface and then from that surface create a pattern that can be used to fill another surface. For example, `jgraphics_patter_create_for_surface()`. There are also gradients: see `jgraphics_pattern_create_linear()` and `jgraphics_pattern_create_radial()`.

33.61 Colors

Collaboration diagram for Colors:



Data Structures

- struct [t_jrgb](#)
A color composed of red, green, and blue components.
- struct [t_jrgba](#)
A color composed of red, green, blue, and alpha components.

Functions

- void [jrgba_to_atoms](#) ([t_jrgba](#) *c, [t_atom](#) *argv)
Get the components of a color in an array of pre-allocated atoms.
- [t_max_err atoms_to_jrgba](#) (long argc, [t_atom](#) *argv, [t_jrgba](#) *c)
Set the components of a color by providing an array of atoms.
- void [jrgba_set](#) ([t_jrgba](#) *prgba, double r, double g, double b, double a)
Set the components of a color.
- void [jrgba_copy](#) ([t_jrgba](#) *dest, [t_jrgba](#) *src)
Copy a color.
- long [jrgba_compare](#) ([t_jrgba](#) *rgba1, [t_jrgba](#) *rgba2)
Compare two colors for equality.
- [t_max_err jrgba_attr_get](#) ([t_jrgba](#) *jrgba, long *argc, [t_atom](#) **argv)
Get the value of a [t_jrgba](#) struct, returned as an array of atoms with the values for each component.
- [t_max_err jrgba_attr_set](#) ([t_jrgba](#) *jrgba, long argc, [t_atom](#) *argv)
Set the value of a [t_jrgba](#) struct, given an array of atoms with the values to use.

33.61.1 Function Documentation

33.61.1.1 [t_max_err atoms_to_jrgba](#) (long argc, t_atom * argv, t_rgba * c)

Set the components of a color by providing an array of atoms.

If it is an array of 3 atoms, then the atoms provided should define the red, green, and blue components (in this order) in a range of [0.0, 1.0]. If a 4th atom is provided, it will define the alpha channel. If the alpha channel is not defined then it is assumed to be 1.0.

Parameters

- argc*** The number of atoms in the array provided in ***argv***. This should be 3 or 4 depending on whether or not the alpha channel is being provided.
- argv*** The address to the first of an array of atoms that define the color.
- c*** The address of a **t_jrgba** struct for which the color will be defined.

Returns

A Max error code.

33.61.1.2 t_max_err jrgba_attr_get (t_jrgba **jrgba*, long **argc*, t_atom *argv*)**

Get the value of a **t_jrgba** struct, returned as an array of atoms with the values for each component.

Parameters

- jrgba*** The color struct whose color will be retrieved.
- argc*** The address of a variable that will be set with the number of atoms in the ***argv*** array. The returned value should be 4. The value of the int should be set to 0 prior to calling this function.
- argv*** The address of a **t_atom** pointer that will receive the a new array of atoms set to the values of the ***jrgba*** struct. The pointer should be set to NULL prior to calling this function. There should be 4 atoms returned, representing alpha, red, green, and blue components. When you are done using the atoms, you are responsible for freeing the pointer using **sysmem_freeptr()**.

Returns

This function returns the error code **MAX_ERR_NONE** if successful, or one of the other error codes defined in **e_max_errorcodes** if unsuccessful.

33.61.1.3 t_max_err jrgba_attr_set (t_jrgba **jrgba*, long *argc*, t_atom **argv*)

Set the value of a **t_jrgba** struct, given an array of atoms with the values to use.

Parameters

- jrgba*** The color struct whose color will be set.
- argc*** The number of atoms in the array. This must be 4.
- argv*** The address of the first of the atoms in the array. There must be 4 atoms, representing alpha, red, green, and blue components.

Returns

This function returns the error code **MAX_ERR_NONE** if successful, or one of the other error codes defined in **e_max_errorcodes** if unsuccessful.

33.61.1.4 long jrgba_compare (t_jrgba **rgba1*, t_jrgba **rgba2*)

Compare two colors for equality.

Parameters

- rgba1* The address of a [t_jrgba](#) struct to compare.
- rgba2* The address of another [t_jrgba](#) struct to compare.

Returns

returns 1 if *rgba1* == *rgba2*.

33.61.1.5 void jrgba_copy ([t_jrgba](#) * *dest*, [t_jrgba](#) * *src*)

Copy a color.

Parameters

- dest* The address of a [t_jrgba](#) struct to which the color will be copied.
- src* The address of a [t_jrgba](#) struct from which the color will be copied.

33.61.1.6 void jrgba_set ([t_jrgba](#) * *prgba*, double *r*, double *g*, double *b*, double *a*)

Set the components of a color.

Parameters

- prgba* The address of a [t_jrgba](#) struct for which the color will be defined.
- r* The value of the red component in a range of [0.0, 1.0].
- g* The value of the green component in a range of [0.0, 1.0].
- b* The value of the blue component in a range of [0.0, 1.0].
- a* The value of the alpha component in a range of [0.0, 1.0].

33.61.1.7 void jrgba_to_atoms ([t_jrgba](#) * *c*, [t_atom](#) * *argv*)

Get the components of a color in an array of pre-allocated atoms.

Parameters

- argv* The address to the first of an array of atoms that will hold the result. At least 4 atoms must be allocated, as 4 atoms will be set by this function for the red, green, blue, and alpha components.
- c* The address of a [t_jrgba](#) struct from which the color components will be fetched.

33.62 TextField

The textfield is a high-level text display object that may be used by a UI object to represent text in a patcher.

Collaboration diagram for TextField:



Functions

- `t_object * textfield_get_owner (t_object *tf)`
Return the object that owns a particular textfield.
- `t_max_err textfield_get_textcolor (t_object *tf, t_jrgba *prgba)`
Retrieve the color of the text in a textfield.
- `t_max_err textfield_set_textcolor (t_object *tf, t_jrgba *prgba)`
Set the color of the text in a textfield.
- `t_max_err textfield_get_bgcolor (t_object *tf, t_jrgba *prgba)`
Retrieve the background color of a textfield.
- `t_max_err textfield_set_bgcolor (t_object *tf, t_jrgba *prgba)`
Set the background color of a textfield.
- `t_max_err textfield_get_textmargins (t_object *tf, double *pleft, double *ptop, double *pright, double *pbottom)`
Retrieve the margins from the edge of the textfield to the text itself in a textfield.
- `t_max_err textfield_set_textmargins (t_object *tf, double left, double top, double right, double bottom)`
Set the margins from the edge of the textfield to the text itself in a textfield.
- `char textfield_get_editonclick (t_object *tf)`
Return the value of the 'editonclick' attribute of a textfield.
- `t_max_err textfield_set_editonclick (t_object *tf, char c)`
Set the 'editonclick' attribute of a textfield.
- `char textfield_get_selectallonedit (t_object *tf)`
Return the value of the 'selectallonedit' attribute of a textfield.
- `t_max_err textfield_set_selectallonedit (t_object *tf, char c)`
Set the 'selectallonedit' attribute of a textfield.
- `char textfield_get_noactivate (t_object *tf)`
Return the value of the 'noactivate' attribute of a textfield.
- `t_max_err textfield_set_noactivate (t_object *tf, char c)`

Set the 'noactivate' attribute of a textfield.

- `char textfield_get_READONLY (t_object *tf)`

Return the value of the 'readonly' attribute of a textfield.

- `t_max_err textfield_set_READONLY (t_object *tf, char c)`

Set the 'readonly' attribute of a textfield.

- `char textfield_get_wordwrap (t_object *tf)`

Return the value of the 'wordwrap' attribute of a textfield.

- `t_max_err textfield_set_wordwrap (t_object *tf, char c)`

Set the 'wordwrap' attribute of a textfield.

- `char textfield_get_useellipsis (t_object *tf)`

Return the value of the 'useellipsis' attribute of a textfield.

- `t_max_err textfield_set_useellipsis (t_object *tf, char c)`

Set the 'useellipsis' attribute of a textfield.

- `char textfield_get_autoscroll (t_object *tf)`

Return the value of the 'autoscroll' attribute of a textfield.

- `t_max_err textfield_set_autoscroll (t_object *tf, char c)`

Set the 'autoscroll' attribute of a textfield.

- `char textfield_get_wantsreturn (t_object *tf)`

Return the value of the 'wantsreturn' attribute of a textfield.

- `t_max_err textfield_set_wantsreturn (t_object *tf, char c)`

Set the 'wantsreturn' attribute of a textfield.

- `char textfield_get_wantstab (t_object *tf)`

Return the value of the 'wantstab' attribute of a textfield.

- `t_max_err textfield_set_wantstab (t_object *tf, char c)`

Set the 'wantstab' attribute of a textfield.

- `char textfield_get_underline (t_object *tf)`

Return the value of the 'underline' attribute of a textfield.

- `t_max_err textfield_set_underline (t_object *tf, char c)`

Set the 'underline' attribute of a textfield.

- `t_max_err textfield_set_emptytext (t_object *tf, t_symbol *txt)`

Set the 'empty' text of a textfield.

- `t_symbol * textfield_get_emptytext (t_object *tf)`

Retrieve the 'empty' text of a textfield.

33.62.1 Detailed Description

The textfield is a high-level text display object that may be used by a UI object to represent text in a patcher. It is built on the lower-level [TextLayout](#)

33.62.2 Function Documentation

33.62.2.1 `char textfield_get_autoscroll (t_object *tf)`

Return the value of the 'autoscroll' attribute of a textfield.

Parameters

tf The textfield instance pointer.

Returns

A value of the attribute.

33.62.2.2 `t_max_err textfield_get_bgcolor (t_object *tf, t_jrgba *prgba)`

Retrieve the background color of a textfield.

Parameters

tf The textfield instance pointer.

prgba The address of a valid [t_jrgba](#) whose values will be filled-in upon return.

Returns

A Max error code.

33.62.2.3 `char textfield_get_editonclick (t_object *tf)`

Return the value of the 'editonclick' attribute of a textfield.

Parameters

tf The textfield instance pointer.

Returns

A value of the attribute.

33.62.2.4 `t_symbol* textfield_get_emptytext (t_object *tf)`

Retrieve the 'empty' text of a textfield.

The empty text is the text that is displayed in the textfield when no text is present. By default this is gensym("") .

Parameters

tf The textfield instance pointer.

Returns

The current text used as the empty text.

33.62.2.5 char textfield_get_noactivate (t_object * *tf*)

Return the value of the 'noactivate' attribute of a textfield.

Parameters

tf The textfield instance pointer.

Returns

A value of the attribute.

33.62.2.6 t_object* textfield_get_owner (t_object * *tf*)

Return the object that owns a particular textfield.

Parameters

tf The textfield instance pointer.

Returns

A pointer to the owning object.

33.62.2.7 char textfield_get_READONLY (t_object * *tf*)

Return the value of the 'readonly' attribute of a textfield.

Parameters

tf The textfield instance pointer.

Returns

A value of the attribute.

33.62.2.8 char textfield_get_SELECTALLONEDIT (t_object * *tf*)

Return the value of the 'selectallonedit' attribute of a textfield.

Parameters

tf The textfield instance pointer.

Returns

A value of the attribute.

33.62.2.9 t_max_err textfield_get_textcolor (t_object * *tf*, t_jrgba * *prgba*)

Retrieve the color of the text in a textfield.

Parameters

tf The textfield instance pointer.

prgba The address of a valid [t_jrgba](#) whose values will be filled-in upon return.

Returns

A Max error code.

33.62.2.10 t_max_err textfield_get_textmargins (t_object * *tf*, double * *pleft*, double * *ptop*, double * *pright*, double * *pbottom*)

Retrieve the margins from the edge of the textfield to the text itself in a textfield.

Parameters

tf The textfield instance pointer.

pleft The address of a variable to hold the value of the left margin upon return.

ptop The address of a variable to hold the value of the top margin upon return.

pright The address of a variable to hold the value of the right margin upon return.

pbottom The address of a variable to hold the value of the bottom margin upon return.

Returns

A Max error code.

33.62.2.11 char textfield_get_underline (t_object * *tf*)

Return the value of the 'underline' attribute of a textfield.

Parameters

tf The textfield instance pointer.

Returns

A value of the attribute.

33.62.2.12 char textfield_get_useellipsis (t_object * *tf*)

Return the value of the 'useellipsis' attribute of a textfield.

Parameters

tf The textfield instance pointer.

Returns

A value of the attribute.

33.62.2.13 char textfield_get_wantsreturn (t_object * *tf*)

Return the value of the 'wantsreturn' attribute of a textfield.

Parameters

tf The textfield instance pointer.

Returns

A value of the attribute.

33.62.2.14 char textfield_get_wantstab (t_object * *tf*)

Return the value of the 'wantstab' attribute of a textfield.

Parameters

tf The textfield instance pointer.

Returns

A value of the attribute.

33.62.2.15 char textfield_get_wordwrap (t_object * *tf*)

Return the value of the 'wordwrap' attribute of a textfield.

Parameters

tf The textfield instance pointer.

Returns

A value of the attribute.

33.62.2.16 t_max_err textfield_set_autoscroll (t_object * *tf*, char *c*)

Set the 'autoscroll' attribute of a textfield.

Parameters

tf The textfield instance pointer.

c The new value for the attribute.

Returns

A Max error code.

33.62.2.17 t_max_err textfield_set_bgcolor (t_object * *tf*, t_jrgba * *prgba*)

Set the background color of a textfield.

Parameters

- tf* The textfield instance pointer.
- prgba* The address of a [t_jrgba](#) containing the new color to use.

Returns

A Max error code.

33.62.2.18 t_max_err textfield_set_editonclick (t_object * *tf*, char *c*)

Set the 'editonclick' attribute of a textfield.

Parameters

- tf* The textfield instance pointer.
- c* The new value for the attribute.

Returns

A Max error code.

33.62.2.19 t_max_err textfield_set_emptytext (t_object * *tf*, t_symbol * *txt*)

Set the 'empty' text of a textfield.

The empty text is the text that is displayed in the textfield when no text is present. By default this is gensym("") .

Parameters

- tf* The textfield instance pointer.
- txt* A symbol containing the new text to display when the textfield has no content.

Returns

A Max error code.

33.62.2.20 t_max_err textfield_set_noactivate (t_object * *tf*, char *c*)

Set the 'noactivate' attribute of a textfield.

Parameters

- tf* The textfield instance pointer.
- c* The new value for the attribute.

Returns

A Max error code.

33.62.2.21 t_max_err textfield_set_READONLY (t_object * *tf*, char *c*)

Set the 'readonly' attribute of a textfield.

Parameters

- tf* The textfield instance pointer.
- c* The new value for the attribute.

Returns

A Max error code.

33.62.2.22 t_max_err textfield_set_SELECTALLONEDIT (t_object * *tf*, char *c*)

Set the 'selectallonedit' attribute of a textfield.

Parameters

- tf* The textfield instance pointer.
- c* The new value for the attribute.

Returns

A Max error code.

33.62.2.23 t_max_err textfield_set_textcolor (t_object * *tf*, t_jrgba * *prgba*)

Set the color of the text in a textfield.

Parameters

- tf* The textfield instance pointer.
- prgba* The address of a [t_jrgba](#) containing the new color to use.

Returns

A Max error code.

33.62.2.24 t_max_err textfield_set_textmargins (t_object * *tf*, double *left*, double *top*, double *right*, double *bottom*)

Set the margins from the edge of the textfield to the text itself in a textfield.

Parameters

- tf* The textfield instance pointer.
- left* The new value for the left margin.
- top* The new value for the top margin.
- right* The new value for the right margin.
- bottom* The new value for the bottom margin.

Returns

A Max error code.

33.62.2.25 t_max_err textfield_set_underline (t_object * *tf*, char *c*)

Set the 'underline' attribute of a textfield.

Parameters

- tf* The textfield instance pointer.
- c* The new value for the attribute.

Returns

A Max error code.

33.62.2.26 t_max_err textfield_set_useellipsis (t_object * *tf*, char *c*)

Set the 'useellipsis' attribute of a textfield.

Parameters

- tf* The textfield instance pointer.
- c* The new value for the attribute.

Returns

A Max error code.

33.62.2.27 t_max_err textfield_set_wantsreturn (t_object * *tf*, char *c*)

Set the 'wantsreturn' attribute of a textfield.

Parameters

- tf* The textfield instance pointer.
- c* The new value for the attribute.

Returns

A Max error code.

33.62.2.28 t_max_err textfield_set_wantstab (t_object * *tf*, char *c*)

Set the 'wantstab' attribute of a textfield.

Parameters

- tf* The textfield instance pointer.
- c* The new value for the attribute.

Returns

A Max error code.

33.62.2.29 t_max_err textfield_set_wordwrap (t_object **tf*, char *c*)

Set the 'wordwrap' attribute of a textfield.

Parameters

- tf* The textfield instance pointer.
- c* The new value for the attribute.

Returns

A Max error code.

33.63 TextLayout

A textlayout is lower-level text rendering object used by higher-level entities such as [TextField](#).

Collaboration diagram for TextLayout:



Enumerations

- enum `t_jgraphics_textlayout_flags` {

`JGRAPHICS_TEXTLAYOUT_NOWRAP` = 1,
`JGRAPHICS_TEXTLAYOUT_USEELLIPSIS` = 3 }
- Flags for setting text layout options.*

Functions

- `t_jtextlayout * jtextlayout_create ()`
Create a new textlayout object.
- `t_jtextlayout * jtextlayout_withbgcolor (t_jgraphics *g, t_jrgba *bgcolor)`
Create a new textlayout object.
- `void jtextlayout_destroy (t_jtextlayout *textlayout)`
Release/free a textlayout object.
- `void jtextlayout_set (t_jtextlayout *textlayout, const char *utf8, t_jfont *jfont, double x, double y, double width, double height, t_jgraphics_text_justification justification, t_jgraphics_textlayout_flags flags)`
Set the text and attributes of a textlayout object.
- `void jtextlayout_settextcolor (t_jtextlayout *textlayout, t_jrgba *textcolor)`
Set the color to render text in a textlayout object.
- `void jtextlayout_measure (t_jtextlayout *textlayout, long startindex, long numchars, long includewhitespace, double *width, double *height, long *numlines)`
Return a measurement of how much space will be required to draw the text of a textlayout.
- `void jtextlayout_draw (t_jtextlayout *tl, t_jgraphics *g)`
Draw a textlayout in a given graphics context.
- `long jtextlayout_getnumchars (t_jtextlayout *tl)`
Retrieve a count of the number of characters in a textlayout object.
- `t_max_err jtextlayout_getcharbox (t_jtextlayout *tl, long index, t_rect *rect)`
Retrieve the `t_rect` containing a character at a given index.

- `t_max_err jtextlayout_getchar (t_jtextlayout *tl, long index, long *pch)`

Retrieve the unicode character at a given index.

33.63.1 Detailed Description

A textlayout is lower-level text rendering object used by higher-level entities such as [TextField](#).

33.63.2 Enumeration Type Documentation

33.63.2.1 enum t_jgraphics_textlayout_flags

Flags for setting text layout options.

Enumerator:

`JGRAPHICS_TEXTLAYOUT_NOWRAP` disable word wrapping

`JGRAPHICS_TEXTLAYOUT_USEELLIPSIS` show ... if a line doesn't fit (implies NOWRAP too)

33.63.3 Function Documentation

33.63.3.1 t_jtextlayout* jtextlayout_create ()

Create a new textlayout object.

Returns

The new textlayout object.

33.63.3.2 void jtextlayout_destroy (t_jtextlayout * *textlayout*)

Release/free a textlayout object.

Parameters

textlayout The textlayout object to release.

33.63.3.3 void jtextlayout_draw (t_jtextlayout * *tl*, t_jgraphics * *g*)

Draw a textlayout in a given graphics context.

Parameters

tl The textlayout object to query.

g The graphics context in which to draw the text.

33.63.3.4 t_max_err jtextlayout_getchar (t_jtextlayout * *tl*, long *index*, long * *pch*)

Retrieve the unicode character at a given index.

Parameters

tl The textlayout object to query.

index The index from which to fetch the unicode character.

pch The address of a variable to hold the unicode character value upon return.

Returns

A Max error code.

33.63.3.5 t_max_err jtextlayout_getcharbox (t_jtextlayout * *tl*, long *index*, t_rect * *rect*)

Retrieve the [t_rect](#) containing a character at a given index.

Parameters

tl The textlayout object to query.

index The index from which to fetch the unicode character.

rect The address of a valid [t_rect](#) which will be filled in upon return.

Returns

A Max error code.

33.63.3.6 long jtextlayout_getnumchars (t_jtextlayout * *tl*)

Retrieve a count of the number of characters in a textlayout object.

Parameters

tl The textlayout object to query.

Returns

The number of characters.

33.63.3.7 void jtextlayout_measure (t_jtextlayout * *textlayout*, long *startindex*, long *numchars*, long *includewhitespace*, double * *width*, double * *height*, long * *numlines*)

Return a measurement of how much space will be required to draw the text of a textlayout.

Parameters

textlayout The textlayout object to query.

startindex You can measure a subset of the characters. This defines the character from which to start.

numchars Pass -1 for all characters from startindex to end

includewhitespace Define whether to measure with or without whitespace truncated from edges.

width Returns the width of text not including any margins.

height Returns the height of text not including any margins.

numlines Returns the number of lines of text.

33.63.3.8 void jtextlayout_set (t_jtextlayout * *textlayout*, const char * *utf8*, t_jfont * *jfont*, double *x*, double *y*, double *width*, double *height*, t_jgraphics_text_justification *justification*, t_jgraphics_textlayout_flags *flags*)

Set the text and attributes of a textlayout object.

Parameters

textlayout The textlayout object.

utf8 The text to render.

jfont The font with which to render the text.

x The text is placed within rect specified by x, y, width, height.

y The text is placed within rect specified by x, y, width, height.

width The text is placed within rect specified by x, y, width, height.

height The text is placed within rect specified by x, y, width, height.

justification How to justify the text within the rect.

flags Additional flags to control behaviour.

33.63.3.9 void jtextlayout_settextcolor (t_jtextlayout * *textlayout*, t_jrgba * *textcolor*)

Set the color to render text in a textlayout object.

Parameters

textlayout The textlayout object for which to set the color.

textcolor The color for the text.

33.63.3.10 t_jtextlayout* jtextlayout_withbgcolor (t_jgraphics * *g*, t_jrgba * *bgcolor*)

Create a new textlayout object.

This gives a hint to the textlayout as to what the text bgcolor will be. It won't actually paint the bg for you. But, it does let it do a better job.

Parameters

g The graphics context for the textlayout.

bgcolor The background color for the textlayout.

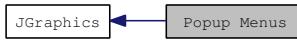
Returns

The new textlayout object.

33.64 Popup Menus

Popup menu API so externals can create popup menus that can also be drawn into.

Collaboration diagram for Popup Menus:



Functions

- `t_jpopupmenu * jpopupmenu_create ()`
Create a pop-up menu.
- `void jpopupmenu_destroy (t_jpopupmenu *menu)`
Free a pop-up menu created with `jpopupmenu_create()`.
- `void jpopupmenu_clear (t_jpopupmenu *menu)`
Clear the contents of a pop-up menu.
- `void jpopupmenu_setcolors (t_jpopupmenu *menu, t_jrgba text, t_jrgba bg, t_jrgba highlightedtext, t_jrgba highlightedbg)`
Set the colors used by a pop-up menu.
- `void jpopupmenu_setfont (t_jpopupmenu *menu, t_jfont *font)`
Set the font used by a pop-up menu.
- `void jpopupmenu_additem (t_jpopupmenu *menu, int itemid, const char *utf8Text, t_jrgba *textColor, int checked, int disabled, t_jsurface *icon)`
Add an item to a pop-up menu.
- `void jpopupmenu_addsubmenu (t_jpopupmenu *menu, const char *utf8Name, t_jpopupmenu *submenu, int disabled)`
Add a pop-menu to another pop-menu as a submenu.
- `void jpopupmenu_addseperator (t_jpopupmenu *menu)`
Add a separator to a pop-menu.
- `int jpopupmenu_popup (t_jpopupmenu *menu, t_pt screen, int defitemid)`
Tell a menu to display at a specified location.
- `int jpopupmenu_popup_abovebox (t_jpopupmenu *menu, t_object *box, t_object *view, int offset, int defitemid)`
Tell a menu to display above a given box in a patcher.
- `int jpopupmenu_popup_nearbox (t_jpopupmenu *menu, t_object *box, t_object *view, int defitemid)`
Tell a menu to display near a given box in a patcher.

33.64.1 Detailed Description

Popup menu API so externals can create popup menus that can also be drawn into.

33.64.2 Function Documentation

33.64.2.1 void jpopupmenu_additem (*t_jpopupmenu * menu*, *int itemid*, *const char * utf8Text*, *t_jrgba * textColor*, *int checked*, *int disabled*, *t_jsurface * icon*)

Add an item to a pop-up menu.

Parameters

menu The pop-up menu to which the item will be added.

itemid Each menu item should be assigned a unique integer id using this parameter.

utf8Text The text to display in for the menu item.

textColor The color to use for the menu item, or NULL to use the default color.

checked A non-zero value indicates that the item should have a check-mark next to it.

disabled A non-zero value indicates that the item should be disabled.

icon A [t_jsurface](#) will be used as an icon for the menu item if provided here. Pass NULL for no icon.

33.64.2.2 void jpopupmenu_addseperator (*t_jpopupmenu * menu*)

Add a separator to a pop-menu.

Parameters

menu The pop-up menu to which the separator will be added.

33.64.2.3 void jpopupmenu_addsubmenu (*t_jpopupmenu * menu*, *const char * utf8Name*, *t_jpopupmenu * submenu*, *int disabled*)

Add a pop-menu to another pop-menu as a submenu.

Parameters

menu The pop-up menu to which a menu will be added as a submenu.

utf8Name The name of the menu item.

submenu The pop-up menu which will be used as the submenu.

disabled Pass a non-zero value to disable the menu item.

33.64.2.4 void jpopupmenu_clear (*t_jpopupmenu * menu*)

Clear the contents of a pop-up menu.

Parameters

menu The pop-up menu whose contents will be cleared.

33.64.2.5 `t_jpopupmenu* jpopupmenu_create()`

Create a pop-up menu.

Free this pop-up menu using [jpopupmenu_destroy\(\)](#).

Returns

A pointer to the newly created jpopupmenu object.

33.64.2.6 `void jpopupmenu_destroy (t_jpopupmenu * menu)`

Free a pop-up menu created with [jpopupmenu_create\(\)](#).

Parameters

menu The pop-up menu to be freed.

33.64.2.7 `int jpopupmenu_popup (t_jpopupmenu * menu, t_pt screen, int defitemid)`

Tell a menu to display at a specified location.

Parameters

menu The pop-up menu to display.

screen The point at which to display in screen coordinates.

defitemid The initially choosen item id.

Returns

The item id for the item in the menu choosen by the user.

33.64.2.8 `int jpopupmenu_popup_abovebox (t_jpopupmenu * menu, t_object * box, t_object * view, int offset, int defitemid)`

Tell a menu to display above a given box in a patcher.

Parameters

menu The pop-up menu to display.

box The box above which to display the menu.

view The patcherview for the box in which to display the menu.

offset An offset from the box position at which to display the menu.

defitemid The initially choosen item id.

Returns

The item id for the item in the menu choosen by the user.

33.64.2.9 int jpopupmenu_popup_nearbox (t_jpopupmenu * *menu*, t_object * *box*, t_object * *view*, int *defitemid*)

Tell a menu to display near a given box in a patcher.

Parameters

menu The pop-up menu to display.
box The box above which to display the menu.
view The patcherview for the box in which to display the menu.
defitemid The initially choosen item id.

Returns

The item id for the item in the menu choosen by the user.

33.64.2.10 void jpopupmenu_setcolors (t_jpopupmenu * *menu*, t_jrgba *text*, t_jrgba *bg*, t_jrgba *highlightedtext*, t_jrgba *highlightedb*)

Set the colors used by a pop-up menu.

Parameters

menu The pop-up menu to which the colors will be applied.
text The text color for menu items.
bg The background color for menu items.
highlightedtext The text color for the highlighted menu item.
highlightedb The background color the highlighted menu item.

33.64.2.11 void jpopupmenu_setfont (t_jpopupmenu * *menu*, t_jfont * *font*)

Set the font used by a pop-up menu.

Parameters

menu The pop-up menu whose font will be set.
font A pointer to a font object, whose font info will be copied to the pop-up menu.

33.65 Box Layer

The boxlayer functions provide way to make it easier to use cached offscreen images (layers) in your drawing.

Collaboration diagram for Box Layer:



Functions

- `t_max_err jbox_invalidate_layer (t_object *b, t_object *view, t_symbol *name)`
Invalidate a layer, indicating that it needs to be re-drawn.
- `t_jgraphics * jbox_start_layer (t_object *b, t_object *view, t_symbol *name, double width, double height)`
Create a layer, and ready it for drawing commands.
- `t_max_err jbox_end_layer (t_object *b, t_object *view, t_symbol *name)`
Conclude a layer, indicating that it is complete and ready for painting.
- `t_max_err jbox_paint_layer (t_object *b, t_object *view, t_symbol *name, double x, double y)`
Paint a layer at a given position.

33.65.1 Detailed Description

The boxlayer functions provide way to make it easier to use cached offscreen images (layers) in your drawing. The general idea is to do something like this:

```

t_jgraphics *g;
g = jbox_start_layer(box, view, layername, width, height);
if (g) {
    // draw to your new offscreen context here
    // the second time you call jbox_start_layer() it will return NULL
    // since you already drew it -- you don't have to do drawing the second time
    e
    jbox_end_layer(box, view, layername);
}
jbox_paint_layer(box, view, layername, xpos, ypos);

```

Then, if something changes where you would need to redraw the layer you invalidate it:

```
jbox_invalidate_layer(box, view, layername);
```

or

```
jbox_invalidate_layer(box, NULL, layername); // to invalidate for all views
```

Each view has its own layer stored since if a patcher has multiple views each could be at a different zoom level.

33.65.2 Function Documentation

33.65.2.1 `t_max_err jbox_end_layer (t_object * b, t_object * view, t_symbol * name)`

Conclude a layer, indicating that it is complete and ready for painting.

Parameters

- b* The object/box for the layer opened by [jbox_start_layer\(\)](#).
- view* The patcherview for the object opened by [jbox_start_layer\(\)](#).
- name* The name of the layer.

Returns

A Max error code.

33.65.2.2 `t_max_err jbox_invalidate_layer (t_object * b, t_object * view, t_symbol * name)`

Invalidate a layer, indicating that it needs to be re-drawn.

Parameters

- b* The object/box to invalidate.
- view* The patcherview for the object which should be invalidated, or NULL for all patcherviews.
- name* The name of the layer to invalidate.

Returns

A Max error code.

33.65.2.3 `t_max_err jbox_paint_layer (t_object * b, t_object * view, t_symbol * name, double x, double y)`

Paint a layer at a given position.

Note that the current color alpha value is used when painting layers to allow you to blend layers. The same is also true for [jgraphics_image_surface_draw\(\)](#) and [jgraphics_image_surface_draw_fast\(\)](#).

Parameters

- b* The object/box to be painted.
- view* The patcherview for the object which should be painted, or NULL for all patcherviews.
- name* The name of the layer to paint.
- x* The x-coordinate for the position at which to paint the layer.
- y* The y-coordinate for the position at which to paint the layer.

Returns

A Max error code.

33.65.2.4 `t_jgraphics* jbox_start_layer (t_object * b, t_object * view, t_symbol * name, double width, double height)`

Create a layer, and ready it for drawing commands.

The layer drawing commands must be wrapped with a matching call to [jbox_end_layer\(\)](#) prior to calling [jbox_paint_layer\(\)](#).

Parameters

- b*** The object/box to which the layer is attached.
- view*** The patcherview for the object to which the layer is attached.
- name*** A name for this layer.
- width*** The width of the layer.
- height*** The height of the layer.

Returns

A [t_jgraphics](#) context for drawing into the layer.

33.66 DataView

The jdatatable object provides a mechanism to display data in a tabular format.

Collaboration diagram for DataView:



Data Structures

- struct [t_celldesc](#)
A datatable cell description.
- struct [t_jcolumn](#)
A datatable column.
- struct [t_jdatatable](#)
The datatable object.
- struct [t_privatesortrec](#)
used to pass data to a client sort function

Functions

- void * [jdatatable_new](#) (void)
Create a datatable.
- void [jdatatable_setclient](#) ([t_object](#) *dv, [t_object](#) *client)
Set a datatable's client.
- [t_object](#) * [jdatatable_getclient](#) ([t_object](#) *dv)
Get a pointer to a datatable's client.

33.66.1 Detailed Description

The jdatatable object provides a mechanism to display data in a tabular format. In Max this is used internally for the implementation of the inspectors, file browser, preferences, and jit.cellblock object, among others.

A jdatatable object does not contain the information that it presents. The object you create will maintain the data and then make the data available to the datatable using the provided api.

33.66.2 Function Documentation

33.66.2.1 [t_object*](#) [jdatatable_getclient](#) ([t_object](#) * dv)

Get a pointer to a datatable's client.

The client is the object to which the dataview will send messages to get data, notify of changes to cells, etc.

Parameters

dv The dataview instance.

Returns

A pointer to the dataview's client object.

33.66.2.2 void* jdataview_new (void)

Create a dataview.

You should free it with [object_free\(\)](#).

Returns

A pointer to the new instance.

33.66.2.3 void jdataview_setclient (t_object * *dv*, t_object * *client*)

Set a dataview's client.

The client is the object to which the dataview will send messages to get data, notify of changes to cells, etc. Typically this is the object in which you are creating the dataview.

Parameters

dv The dataview instance.

client The object to be assigned as the dataview's client.

33.67 Unicode

Data Structures

- struct [t_charset_converter](#)

The charset_converter object.

Functions

- [t_max_err charset_convert \(t_symbol *src_encoding, const char *in, long inbytes, t_symbol *dest_encoding, char **out, long *outbytes\)](#)
A convenience function that simplifies usage by wrapping the other charset functions.
- unsigned short * [charset_utf8tounicode \(char *s, long *outlen\)](#)
Convert a UTF8 C-String into a 16-bit-wide-character array.
- char * [charset_unicodetutf8 \(unsigned short *s, long len, long *outlen\)](#)
Convert a 16-bit-wide-character array into a UTF C-string.
- long [charset_utf8_count \(char *utf8, long *bytecount\)](#)
Returns utf8 character count, and optionally bytecount.
- char * [charset_utf8_offset \(char *utf8, long charoffset, long *byteoffset\)](#)
Returns utf8 character offset (positive or negative), and optionally byte offset.

33.67.1 Detailed Description

33.67.2 Character Encodings

Currently supported character encodings

- _sym_utf_8; // utf-8, no bom
- _sym_utf_16; // utf-16, big-endian
- _sym_utf_16be; // utf-16, big-endian
- _sym_utf_16le; // utf-16, little-endian
- _sym_iso_8859_1; // iso-8859-1 (latin-1)
- _sym_us_ascii; // us-ascii 7-bit
- _sym_ms_ansi; // ms-ansi (microsoft code page 1252)
- _sym_macroman; // mac roman
-
- _sym_charset_converter;
- _sym_convert;

33.67.2.1 Example Usage

```
t_charset_converter *conv = object_new(CLASS_NOBOX, gensym("charset_converter")
    ), ps_macroman, ps_ms_ansi);
char *cstr = "Text to convert";
char *cvtbuffer = NULL; // to-be-allocated data buffer
long cvtbuflen = 0; // length of buffer on output

if (conv) {
    // note that it isn't necessary to send in a 0-terminated string, although
    we do so here
    if (object_method(conv, gensym("convert"), cstr, strlen(cstr) + 1, &cvtbuff
er, &cvtbuflen) == ERR_NONE) {
        // do something with the converted buffer
        sysmem_freeptr(cvtbuffer); // free newly allocated data buffer
    }
    object_free(conv); // free converter
}
```

33.67.3 Function Documentation

33.67.3.1 t_max_err charset_convert (t_symbol * *src_encoding*, const char * *in*, long *inbytes*, t_symbol * *dest_encoding*, char ** *out*, long * *outbytes*)

A convenience function that simplifies usage by wrapping the other charset functions.

Parameters

src_encoding The name encoding of the input.
in The input string.
inbytes The number of bytes in the input string.
dest_encoding The name of the encoding to use for the output.
out The address of a char*, which will be allocated and filled with the string in the new encoding.
outbytes The address of a value that will hold the number of bytes long the output is upon return.

Returns

A Max error code.

Remarks

Remember to call sysmem_freeptr(*out) to free any allocated memory.

33.67.3.2 char* charset_unicodetoutf8 (unsigned short * *s*, long *len*, long * *outlen*)

Convert a 16-bit-wide-character array into a UTF C-string.

Accepts either null termination, or not (len is zero in the latter case).

Parameters

s An array of wide (16-bit) unicode characters.
len The length of s.
outlen The address of a variable to hold the size of the number of chars but does not include the NULL terminator in the count.

Returns

A UTF8-encoded C-string.

33.67.3.3 long charset_utf8_count (char * *utf8*, long * *bytecount*)

Returns utf8 character count, and optionally bytecount.

Parameters

utf8 The UTF-8 encoded string whose characters are to be counted.

bytecount The address of a variable to hold the byte count on return. Pass NULL if you don't require the byte count.

Returns

The number of characters in the UTF8 string.

33.67.3.4 char* charset_utf8_offset (char * *utf8*, long *charoffset*, long * *byteoffset*)

Returns utf8 character offset (positive or negative), and optionally byte offset.

Parameters

utf8 A UTF-8 encoded string.

charoffset The char offset into the string at which to find the byte offset.

byteoffset The address of a variable to hold the byte offset on return. Pass NULL if you don't require the byte offset.

Returns

The character offset.

33.67.3.5 unsigned short* charset_utf8tounicode (char * *s*, long * *outlen*)

Convert a UTF8 C-String into a 16-bit-wide-character array.

Parameters

s The string to be converted to unicode.

outlen The address of a variable to hold the size of the number of chars but does not include the NULL terminator in the count.

Returns

A pointer to the buffer of unicode (wide) characters.

33.68 Atom Module

Collaboration diagram for Atom Module:



Functions

- `t_jit_err jit_atom_setlong (t_atom *a, long b)`
Sets atom value to long integer.
- `t_jit_err jit_atom_setfloat (t_atom *a, double b)`
Sets atom value to floating point number.
- `t_jit_err jit_atom_setsym (t_atom *a, t_symbol *b)`
Sets atom value to symbol.
- `t_jit_err jit_atom_setobj (t_atom *a, void *b)`
Sets atom value to object pointer.
- `long jit_atom_getlong (t_atom *a)`
Retrieves atom value as long integer.
- `double jit_atom_getfloat (t_atom *a)`
Retrieves atom value as floating point number.
- `t_symbol * jit_atom_getsym (t_atom *a)`
Retrieves atom value as symbol pointer.
- `void * jit_atom_getobj (t_atom *a)`
Retrieves atom value as object pointer.
- `long jit_atom_getcharfix (t_atom *a)`
Retrieves atom value as an 8 bit fixed point number.
- `long jit_atom_arg_getlong (long *c, long idx, long ac, t_atom *av)`
Retrieves atom argument at index as long integer if present.
- `long jit_atom_arg_getfloat (float *c, long idx, long ac, t_atom *av)`
Retrieves atom argument at index as floating point number if present.
- `long jit_atom_arg_getdouble (double *c, long idx, long ac, t_atom *av)`
Retrieves atom argument at index as double precision floating point number if present.
- `long jit_atom_arg_getsym (t_symbol **c, long idx, long ac, t_atom *av)`
Retrieves atom argument at index as symbol pointer if present.

33.68.1 Function Documentation

33.68.1.1 long jit_atom_arg_getdouble (double * *c*, long *idx*, long *ac*, t_atom * *av*)

Retrieves atom argument at index as double precision floating point number if present.

This function is useful for setting the values only if there is an argument at the specified index, otherwise, the input value is untouched.

Parameters

- c* pointer to double (should contain desired default)
- idx* atom argument index
- ac* atom argument count
- av* atom argument vector

Returns

t_jit_err error code. JIT_ERR_NONE if successful.

33.68.1.2 long jit_atom_arg_getfloat (float * *c*, long *idx*, long *ac*, t_atom * *av*)

Retrieves atom argument at index as floating point number if present.

This function is useful for setting the values only if there is an argument at the specified index, otherwise, the input value is untouched.

Parameters

- c* pointer to float (should contain desired default)
- idx* atom argument index
- ac* atom argument count
- av* atom argument vector

Returns

t_jit_err error code. JIT_ERR_NONE if successful.

33.68.1.3 long jit_atom_arg_getlong (long * *c*, long *idx*, long *ac*, t_atom * *av*)

Retrieves atom argument at index as long integer if present.

This function is useful for setting the values only if there is an argument at the specified index, otherwise, the input value is untouched.

Parameters

- c* pointer to long (should contain desired default)
- idx* atom argument index
- ac* atom argument count
- av* atom argument vector

Returns

t_jit_err error code. JIT_ERR_NONE if successful.

33.68.1.4 long jit_atom_arg_getsym (t_symbol ** c, long idx, long ac, t_atom * av)

Retrieves atom argument at index as symbol pointer if present.

This function is useful for setting the values only if there is an argument at the specified index, otherwise, the input value is untouched.

Parameters

c pointer to symbol pointer (should contain desired default)
idx atom argument index
ac atom argument count
av atom argument vector

Returns

t_jit_err error code. JIT_ERR_NONE if successful.

33.68.1.5 long jit_atom_getcharfix (t_atom * a)

Retrieves atom value as an 8 bit fixed point number.

Parameters

a atom pointer

Returns

8 bit fixed point value in the range 0-255. 0 if atom has no numeric value.

33.68.1.6 double jit_atom_getfloat (t_atom * a)

Retrieves atom value as floating point number.

Parameters

a atom pointer

Returns

floating point value. 0 if atom has no numeric value.

33.68.1.7 long jit_atom_getlong (t_atom * a)

Retrieves atom value as long integer.

Parameters

a atom pointer

Returns

long integer value. 0 if atom has no numeric value.

33.68.1.8 void* jit_atom_getobj (t_atom * *a*)

Retrieves atom value as object pointer.

Parameters

a atom pointer

Returns

object pointer. NULL if atom has no object value.

33.68.1.9 t_symbol * jit_atom_getsym (t_atom * *a*)

Retrieves atom value as symbol pointer.

Parameters

a atom pointer

Returns

symbol pointer. _jit_sym_nothing if atom has no symbolic value.

33.68.1.10 t_jit_err jit_atom_setfloat (t_atom * *a*, double *b*)

Sets atom value to floating point number.

Parameters

a atom pointer

b floating point value

Returns

t_jit_err error code.

33.68.1.11 t_jit_err jit_atom_setlong (t_atom * *a*, long *b*)

Sets atom value to long integer.

Parameters

a atom pointer

b integer value

Returns

t_jit_err error code.

33.68.1.12 `t_jit_err jit_atom_setobj (t_atom * a, void * b)`

Sets atom value to object pointer.

Parameters

a atom pointer

b object pointer

Returns

`t_jit_err` error code.

33.68.1.13 `t_jit_err jit_atom_setsym (t_atom * a, t_symbol * b)`

Sets atom value to symbol.

Parameters

a atom pointer

b symbol value

Returns

`t_jit_err` error code.

33.69 Attribute Module

Collaboration diagram for Attribute Module:



Data Structures

- struct `t_jit_attribute`
`t_jit_attribute` object struct.
- struct `t_jit_attr_offset`
`t_jit_attr_offset` object struct.
- struct `t_jit_attr_offset_array`
`t_jit_attr_offset_array` object struct.
- struct `t_jit_attr_filter_clip`
`t_jit_attr_filter_clip` object struct.
- struct `t_jit_attr_filter_proc`
`t_jit_attr_filter_proc` object struct.
- struct `t_jit_attr`
Common attribute struct.

Functions

- `t_symbol * jit_attr_getname (t_jit_attr *x)`
Retrieves attribute name.
- `t_symbol * jit_attr_gettype (t_jit_attr *x)`
Retrieves attribute type.
- `long jit_attr_canget (t_jit_attr *x)`
Retrieves attribute gettable flag.
- `long jit_attr_canset (t_jit_attr *x)`
Retrieves attribute settable flag.
- `long jit_attr_usercanget (t_jit_attr *x)`
Retrieves attribute user gettable flag.
- `long jit_attr_usercanset (t_jit_attr *x)`
Retrieves attribute user settable flag.
- `method jit_attr_getmethod (t_jit_attr *x, t_symbol *methodname)`

Retrieves attribute getter or setter method.

- `t_jit_err jit_attr_filterget (t_jit_attr *x, void *y)`
Sets attribute getter filter.
- `t_jit_err jit_attr_filterset (t_jit_attr *x, void *y)`
Sets attribute setter filter.
- `t_jit_err jit_attr_get (t_jit_attr *x, void *parent, long *ac, t_atom **av)`
Calls attribute getter to retrieve from parent object.
- `t_jit_err jit_attr_set (t_jit_attr *x, void *parent, long ac, t_atom *av)`
Calls attribute setter to set in parent object.
- `t_jit_object * jit_attribute_new (char *name, t_symbol *type, long flags, method mget, method mset)`
Constructs instance of `t_jit_attribute`.
- `t_jit_object * jit_attr_offset_new (char *name, t_symbol *type, long flags, method mget, method mset, long offset)`
Constructs instance of `t_jit_attr_offset`.
- `t_jit_object * jit_attr_offset_array_new (char *name, t_symbol *type, long size, long flags, method mget, method mset, long offsetcount, long offset)`
Constructs instance of `t_jit_attr_offset_array`.
- `t_jit_object * jit_attr_filter_clip_new (void)`
Constructs instance of `t_jit_attr_filter_clip`.
- `t_jit_object * jit_attr_filter_proc_new (method proc)`
Constructs instance of `t_jit_attr_filter_proc`.
- `long jit_attr_getlong (void *x, t_symbol *s)`
Retrieves attribute value as a long integer value.
- `t_jit_err jit_attr_setlong (void *x, t_symbol *s, long c)`
Sets attribute value as a long integer value.
- `float jit_attr_getfloat (void *x, t_symbol *s)`
Retrieves attribute value as a floating point value.
- `t_jit_err jit_attr_setfloat (void *x, t_symbol *s, float c)`
Sets attribute value as a floating point value.
- `t_symbol * jit_attr_getsym (void *x, t_symbol *s)`
Retrieves attribute value as a symbol value.
- `t_jit_err jit_attr_setsym (void *x, t_symbol *s, t_symbol *c)`
Sets attribute value as a symbol value.

- long `jit_attr_getlong_array` (void **x*, `t_symbol` **s*, long max, long **vals*)
Retrieves attribute value as an array of long integer values.
- `t_jit_err jit_attr_setlong_array` (void **x*, `t_symbol` **s*, long count, long **vals*)
Sets attribute value as an array of long integer values.
- long `jit_attr_getchar_array` (void **x*, `t_symbol` **s*, long max, `uchar` **vals*)
Retrieves attribute value as an array of char values.
- `t_jit_err jit_attr_setchar_array` (void **x*, `t_symbol` **s*, long count, `uchar` **vals*)
Sets attribute value as an array of char values.
- long `jit_attr_getfloat_array` (void **x*, `t_symbol` **s*, long max, float **vals*)
Retrieves attribute value as an array of floating point values.
- `t_jit_err jit_attr_setfloat_array` (void **x*, `t_symbol` **s*, long count, float **vals*)
Sets attribute value as an array of floating point values.
- long `jit_attr_getdouble_array` (void **x*, `t_symbol` **s*, long max, double **vals*)
Retrieves attribute value as an array of double precision floating point values.
- `t_jit_err jit_attr_setdouble_array` (void **x*, `t_symbol` **s*, long count, double **vals*)
Sets attribute value as an array of double precision floating point values.
- long `jit_attr_getsym_array` (void **x*, `t_symbol` **s*, long max, `t_symbol` ***vals*)
Retrieves attribute value as an array of symbol values.
- `t_jit_err jit_attr_setsym_array` (void **x*, `t_symbol` **s*, long count, `t_symbol` ***vals*)
Sets attribute value as an array of symbol values.
- long `jit_attr_symcompare` (void **x*, `t_symbol` **name*)
Compares symbol name with name provided.

33.69.1 Function Documentation

33.69.1.1 long `jit_attr_canget` (`t_jit_attr` **x*)

Retrieves attribute gettable flag.

Parameters

x attribute object pointer

Returns

gettable flag

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of any attribute object.

33.69.1.2 long jit_attr_canset (t_jit_attr *x)

Retrieves attribute settable flag.

Parameters

x attribute object pointer

Returns

settable flag

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of any attribute obejct.

33.69.1.3 t_jit_object * jit_attr_filter_clip_new (void)

Constructs instance of [t_jit_attr_filter_clip](#).

Returns

[t_jit_attr_filter_clip](#) object pointer

Warning

This function is not exported, but is provided for reference when calling via jit_object_new.

33.69.1.4 t_jit_object * jit_attr_filter_proc_new (method *proc*)

Constructs instance of [t_jit_attr_filter_proc](#).

Parameters

proc filter procedure

Returns

[t_jit_attr_filter_clip](#) object pointer

Warning

This function is not exported, but is provided for reference when calling via jit_object_new.

33.69.1.5 t_jit_err jit_attr_filterget (t_jit_attr *x, void *y)

Sets attribute getter filter.

Parameters

x attribute object pointer

y getter filter object

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of any attribute obejct.

33.69.1.6 t_jit_err jit_attr_filterset (t_jit_attr **x*, void **y*)

Sets attribute setter filter.

Parameters

x attribute object pointer

y setter filter object

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of any attribute obejct.

33.69.1.7 t_jit_err jit_attr_get (t_jit_attr **x*, void **parent*, long **ac*, t_atom ***av*)

Calls attribute getter to retrieve from parent object.

Parameters

x attribute object pointer

parent target object pointer

ac pointer to argument count

av pointer to argument vector

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of any attribute obejct.

33.69.1.8 long jit_attr_getchar_array (void **x*, t_symbol **s*, long *max*, uchar **vals*)

Retrieves attribute value as an array of char values.

Parameters

- x* object pointer
- s* attribute name
- max* maximum number of values to copy
- vals* pointer to retrieved values

Returns

number of values retrieved.

33.69.1.9 long jit_attr_getdouble_array (void **x*, t_symbol **s*, long *max*, double **vals*)

Retrieves attribute value as an array of double precision floating point values.

Parameters

- x* object pointer
- s* attribute name
- max* maximum number of values to copy
- vals* pointer to retrieved values

Returns

number of values retrieved.

33.69.1.10 float jit_attr_getfloat (void **x*, t_symbol **s*)

Retrieves attribute value as a floating point value.

Parameters

- x* object pointer
- s* attribute name

Returns

floating point value

33.69.1.11 long jit_attr_getfloat_array (void **x*, t_symbol **s*, long *max*, float **vals*)

Retrieves attribute value as an array of floating point values.

Parameters

- x* object pointer

s attribute name
max maximum number of values to copy
vals pointer to retrieved values

Returns

number of values retrieved.

33.69.1.12 long jit_attr_getlong (void *x, t_symbol *s)

Retrieves attribute value as a long integer value.

Parameters

x object pointer
s attribute name

Returns

long integer value

33.69.1.13 long jit_attr_getlong_array (void *x, t_symbol *s, long max, long *vals)

Retrieves attribute value as an array of long integer values.

Parameters

x object pointer
s attribute name
max maximum number of values to copy
vals pointer to retrieved values

Returns

number of values retrieved.

33.69.1.14 method jit_attr_getmethod (t_jit_attr *x, t_symbol *methodname)

Retrieves attribute getter or setter method.

Parameters

x attribute object pointer
methodname "get" or "set" symbol

Returns

getter or setter method

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of any attribute object.

33.69.1.15 `t_symbol * jit_attr_getname (t_jit_attr * x)`

Retrieves attribute name.

Parameters

x attribute object pointer

Returns

attribute name

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of any attribute object.

33.69.1.16 `t_symbol* jit_attr_getsym (void * x, t_symbol * s)`

Retrieves attribute value as a symbol value.

Parameters

x object pointer

s attribute name

Returns

symbol value

33.69.1.17 `long jit_attr_getsym_array (void * x, t_symbol * s, long max, t_symbol ** vals)`

Retrieves attribute value as an array of symbol values.

Parameters

x object pointer

s attribute name

max maximum number of values to copy

vals pointer to retrieved values

Returns

number of values retrieved.

33.69.1.18 `t_symbol * jit_attr_gettype (t_jit_attr * x)`

Retrieves attribute type.

Parameters

x attribute object pointer

Returns

attribute type

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of any attribute object.

33.69.1.19 `t_jit_object * jit_attr_offset_array_new (char * name, t_symbol * type, long size, long flags, method mget, method mset, long offsetcount, long offset)`

Constructs instance of [t_jit_attr_offset_array](#).

Parameters

name attribute name

type data type

size maximum size

flags privacy flags

mget getter method

mset setter method

offsetcount byte offset to count struct member (if zero, remain fixed size with max size)

offset byte offset to array struct member

Returns

[t_jit_attr_offset_array](#) object pointer

Warning

This function is not exported, but is provided for reference when calling via jit_object_new.

33.69.1.20 `t_jit_object * jit_attr_offset_new (char * name, t_symbol * type, long flags, method mget, method mset, long offset)`

Constructs instance of [t_jit_attr_offset](#).

Parameters

name attribute name

type data type

flags privacy flags

mget getter method

mset setter method

offset byte offset to struct member

Returns

[t_jit_attr_offset](#) object pointer

Warning

This function is not exported, but is provided for reference when calling via jit_object_new.

33.69.1.21 t_jit_err jit_attr_set (t_jit_attr *x, void *parent, long ac, t_atom *av)

Calls attribute setter to set in parent object.

Parameters

x attribute object pointer
parent target object pointer
ac argument count
av argument vector

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of any attribute obejct.

33.69.1.22 t_jit_err jit_attr_setchar_array (void *x, t_symbol *s, long count, uchar *vals)

Sets attribute value as an array of char values.

Parameters

x object pointer
s attribute name
count number of values
vals pointer to values

Returns

t_jit_err error code.

33.69.1.23 t_jit_err jit_attr_setdouble_array (void *x, t_symbol *s, long count, double *vals)

Sets attribute value as an array of double precision floating point values.

Parameters

x object pointer
s attribute name
count number of values
vals pointer to values

Returns

t_jit_err error code.

33.69.1.24 t_jit_err jit_attr_setfloat (void *x, t_symbol *s, float c)

Sets attribute value as a floating point value.

Parameters

- x* object pointer
- s* attribute name
- c* value

Returns

t_jit_err error code.

33.69.1.25 t_jit_err jit_attr_setfloat_array (void *x, t_symbol *s, long count, float *vals)

Sets attribute value as an array of floating point values.

Parameters

- x* object pointer
- s* attribute name
- count* number of values
- vals* pointer to values

Returns

t_jit_err error code.

33.69.1.26 t_jit_err jit_attr_setlong (void *x, t_symbol *s, long c)

Sets attribute value as a long integer value.

Parameters

- x* object pointer
- s* attribute name
- c* value

Returns

t_jit_err error code.

33.69.1.27 t_jit_err jit_attr_setlong_array (void *x, t_symbol *s, long count, long *vals)

Sets attribute value as an array of long integer values.

Parameters

- x* object pointer

s attribute name
count number of values
vals pointer to values

Returns

t_jit_err error code.

33.69.1.28 t_jit_err jit_attr_setsym (void **x*, t_symbol **s*, t_symbol **c*)

Sets attribute value as a symbol value.

Parameters

x object pointer
s attribute name
c value

Returns

t_jit_err error code.

33.69.1.29 t_jit_err jit_attr_setsym_array (void **x*, t_symbol **s*, long *count*, t_symbol *vals*)**

Sets attribute value as an array of symbol values.

Parameters

x object pointer
s attribute name
count number of values
vals pointer to values

Returns

t_jit_err error code.

33.69.1.30 long jit_attr_symcompare (void **x*, t_symbol **name*)

Compares symbol name with name provided.

Parameters

x attribute object pointer
name attribute name

Returns

1 if equal, 0 if not equal

33.69.1.31 long jit_attr_usercanget (t_jit_attr *x)

Retrieves attribute user gettable flag.

Parameters

x attribute object pointer

Returns

user gettable flag

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of any attribute object.

33.69.1.32 long jit_attr_usercanset (t_jit_attr *x)

Retrieves attribute user settable flag.

Parameters

x attribute object pointer

Returns

user settable flag

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of any attribute object.

33.69.1.33 t_jit_object * jit_attribute_new (char * name, t_symbol * type, long flags, method mget, method mset)

Constructs instance of [t_jit_attribute](#).

Parameters

name attribute name
type data type
flags privacy flags
mget getter method
mset setter method

Returns

[t_jit_attribute](#) object pointer

Warning

This function is not exported, but is provided for reference when calling via jit_object_new.

33.70 Binary Module

Collaboration diagram for Binary Module:



Functions

- `t_jit_err jit_bin_read_header (t_filehandle fh, ulong *version, long *filesize)`
Reads the header of a JXF binary file.
- `t_jit_err jit_bin_read_chunk_info (t_filehandle fh, ulong *ckid, long *ckszie)`
Reads the info of a chunk from a JXF binary file.
- `t_jit_err jit_bin_write_header (t_filehandle fh, long filesize)`
Writes the header of a JXF binary file.
- `t_jit_err jit_bin_read_matrix (t_filehandle fh, void *matrix)`
Reads matrix data from a JXF binary file.
- `t_jit_err jit_bin_write_matrix (t_filehandle fh, void *matrix)`
Writes a matrix to a JXF binary file.

33.70.1 Function Documentation

33.70.1.1 `t_jit_err jit_bin_read_chunk_info (t_filehandle fh, ulong * ckid, long * ckszie)`

Reads the info of a chunk from a JXF binary file.

Parameters

- fh*** t_filehandle file handle
ckid chunk ID (ie JIT_BIN_CHUNK_CONTAINER, JIT_BIN_CHUNK_MATRIX)
ckszie the size of the chunk

Returns

`t_jit_err` error code.

33.70.1.2 `t_jit_err jit_bin_read_header (t_filehandle fh, ulong * version, long * filesize)`

Reads the header of a JXF binary file.

Parameters

- fh*** t_filehandle file handle
version version of the binary file format (ie JIT_BIN_VERSION_1)

filesize the size of the file

Returns

t_jit_err error code.

33.70.1.3 t_jit_err jit_bin_read_matrix (t_filehandle *fh*, void * *matrix*)

Reads matrix data from a JXF binary file.

Parameters

fh t_filehandle file handle

matrix the matrix data

Returns

t_jit_err error code.

33.70.1.4 t_jit_err jit_bin_write_header (t_filehandle *fh*, long *filesize*)

Writes the header of a JXF binary file.

Parameters

fh t_filehandle file handle

filesize the size of the file

Returns

t_jit_err error code.

33.70.1.5 t_jit_err jit_bin_write_matrix (t_filehandle *fh*, void * *matrix*)

Writes a matrix to a JXF binary file.

Parameters

fh t_filehandle file handle

matrix the matrix data

Returns

t_jit_err error code.

33.71 Class Module

Collaboration diagram for Class Module:



Functions

- `void * jit_class_new (char *name, method mnew, method mfree, long size,...)`
Creates a new class with the name specified by the name argument.
- `t_jit_err jit_class_addmethod (void *c, method m, char *name,...)`
Adds a named method to a class.
- `t_jit_err jit_class_addattr (void *c, t_jit_object *attr)`
Adds an attribute to a class.
- `t_jit_err jit_class_adornment (void *c, t_jit_object *o)`
Adds an adornment to a class.
- `void * jit_class_adornment_get (void *c, t_symbol *classname)`
Retrieves an adornment from a class.
- `t_jit_err jit_class_free (void *c)`
Frees a class.
- `t_symbol * jit_class_nameget (void *c)`
Retrieves the name of a class.
- `long jit_class_symcompare (void *c, t_symbol *name)`
Compares name of class with the name provided.
- `t_jit_err jit_class_register (void *c)`
Registers class in the class registry.
- `method jit_class_method (void *c, t_symbol *methodname)`
Retrieves method function pointer for named method.
- `t_messlist * jit_class_mess (t_jit_class *c, t_symbol *methodname)`
Retrieves messlist entry for named method.
- `void * jit_class_attr_get (void *c, t_symbol *attrname)`
Retrieves attribute pointer associated with name provided.
- `void * jit_class_findbyname (t_symbol *classname)`
Retrieves class pointer associated with name provided.
- `t_jit_err jit_class_addtypedwrapper (void *c, method m, char *name,...)`

Adds a typed wrapper method to a class.

- **t_messlist * jit_class_typedwrapper_get** (void *c, t_symbol *s)
Retrieves typed wrapper messlist pointer associated with name provided.
- **t_jit_err jit_class_method_addargsafe** (void *c, char *argname, char *methodname)
Marks a method as safe to call as an attribute style argument.
- **t_symbol * jit_class_method_argsafe_get** (void *c, t_symbol *s)
Checks to see if symbol is safe to call as an attribute style argument.

33.71.1 Function Documentation

33.71.1.1 t_jit_err jit_class_adddornment (void * c, t_jit_object * o)

Adds an adornment to a class.

Adornments provide additional state and behavior to a class. This is most commonly used for the jit_mop adornment.

Parameters

- c* class pointer
o object to use as adornment

Returns

t_jit_err error code

33.71.1.2 t_jit_err jit_class_addattr (void * c, t_jit_object * attr)

Adds an attribute to a class.

Parameters

- c* class pointer
attr attribute object

Returns

t_jit_err error code

33.71.1.3 t_jit_err jit_class_addmethod (void * c, method m, char * name, ...)

Adds a named method to a class.

Parameters

- c* class pointer
m function called when method is invoked

name method name

... type signature for the method in the standard Max type list format (see Chapter 3 of the Writing External in Max document for more information)

Returns

t_jit_err error code

33.71.1.4 t_jit_err jit_class_adddtypedwrapper (void * *c*, method *m*, char * *name*, ...)

Adds a typed wrapper method to a class.

Typed wrappers typically are used when there is an existing private, untyped method defined for a Jitter class, but it is desirable to expose the method to language bindings which require a typed interface--e.g. Java or JavaScript.

Parameters

c class pointer

m function called when method is invoked

name method name

... type signature for the method in the standard Max type list format (see Chapter 3 of the Writing External in Max document for more information)

Returns

t_jit_err error code

33.71.1.5 void* jit_class_adornment_get (void * *c*, t_symbol * *classname*)

Retrieves an adornment from a class.

Adornments provide additional state and behavior to a class. This is most commonly used for the jit_mop adornment.

Parameters

c class pointer

classname classname of adornment to retrieve

Returns

t_jit_err error code

33.71.1.6 void* jit_class_attr_get (void * *c*, t_symbol * *attrname*)

Retrieves attribute pointer associated with name provided.

Parameters

c class pointer

attrname attribute name

Returns

attribute object pointer

33.71.1.7 void* jit_class_findbyname (t_symbol * *classname*)

Retrieves class pointer associated with name provided.

Parameters

classname class name

Returns

class pointer

33.71.1.8 t_jit_err jit_class_free (void * *c*)

Frees a class.

Warning

This function is not typically used outside of jitlib.

Parameters

c class pointer

Returns

t_jit_err error code

33.71.1.9 t_messlist* jit_class_mess (t_jit_class * *c*, t_symbol * *methodname*)

Retrieves messlist entry for named method.

Parameters

c class pointer

methodname method name

Returns

[t_messlist](#) pointer.

33.71.1.10 method jit_class_method (void * *c*, t_symbol * *methodname*)

Retrieves method function pointer for named method.

Parameters

c class pointer

methodname method name

Returns

method function pointer.

33.71.1.11 `t_jit_err jit_class_method_addargsafe (void * c, char * argname, char * methodname)`

Marks a method as safe to call as an attribute style argument.

Warning

It is important that no argument settable method causes any output into the patcher, or else it could lead to a crash, or other undesired behavior.

Parameters

c class pointer

argname name as used via argument

methodname name of method to map the argument name to

Returns

`t_jit_err` error code

33.71.1.12 `t_symbol* jit_class_method_argsafe_get (void * c, t_symbol * s)`

Checks to see if symbol is safe to call as an attribute style argument.

Parameters

c class pointer

s name as used via argument

Returns

If successful, name of method to map the argument name to. Otherwise, NULL.

33.71.1.13 `t_symbol* jit_class_nameget (void * c)`

Retrieves the name of a class.

Parameters

c class pointer

Returns

`t_symbol` pointer containing name of class

33.71.1.14 `void* jit_class_new (char * name, method mnew, method mfree, long size, ...)`

Creates a new class with the name specified by the name argument.

Parameters

name class name

mnew class constructor

mfree class destructor

size object struct size in bytes

... type signature for the constructor in the standard Max type list format (see Chapter 3 of the Writing External in Max document for more information)

Warning

In order for the Jitter class to be exposed to JavaScript and Java, it is important that the constructor is typed, even if no arguments are provided--i.e. do not use the older strategy of defining Jitter constructors as private and untyped with A_CANT.

Returns

class pointer to be used in other class functions

33.71.1.15 **t_jit_err jit_class_register (void * c)**

Registers class in the class registry.

Parameters

c class pointer

Returns

t_jit_err error code

33.71.1.16 **long jit_class_symcompare (void * c, t_symbol * name)**

Compares name of class with the name provided.

Parameters

c class pointer

name name to compare with class name

Returns

1 if equal, 0 if not equal

33.71.1.17 **t_messlist* jit_class_typedwrapper_get (void * c, t_symbol * s)**

Retrieves typed wrapper messlist pointer associated with name provided.

Parameters

c class pointer

s name

Returns

t_messlist pointer

33.72 Object Module

Collaboration diagram for Object Module:



Functions

- `long jit_object_classname_compare (void *x, t_symbol *name)`
Compares object's class name with the name provided.
- `t_symbol * jit_object_method_argsafe_get (void *x, t_symbol *s)`
Checks to see if symbol is safe to call as an attribute style argument.
- `void * jit_object_new (t_symbol *classname,...)`
Instantiates an object specified by class name.
- `void * jit_object_method (void *x, t_symbol *s,...)`
Calls an object method specified by method name.
- `void * jit_object_method_typed (void *x, t_symbol *s, long ac, t_atom *av, t_atom *rv)`
Calls a typed object method specified by method name.
- `method jit_object_getmethod (void *x, t_symbol *s)`
Retrieves an object method specified by method name.
- `long jit_object_attr_usercanset (void *x, t_symbol *s)`
Determines if an object attribute is user settable.
- `long jit_object_attr_usercanget (void *x, t_symbol *s)`
Determines if an object attribute is user gettable.
- `void * jit_object_attr_get (void *x, t_symbol *attrname)`
Retrieves an object's attribute pointer specified by attribute name.
- `t_jit_err jit_object_free (void *x)`
Frees an object.
- `t_symbol * jit_object_classname (void *x)`
Retrieves an object's class name.
- `void * jit_object_class (void *x)`
Retrieves an object's class pointer.
- `void * jit_object_register (void *x, t_symbol *s)`
Registers an object in the named object registry.
- `t_jit_err jit_object_unregister (void *x)`

Unregisters an object from the named object registry.

- `void * jit_object_findregistered (t_symbol *s)`
Retrieves a registered object associated with name.
- `t_symbol * jit_object_findregisteredbyptr (void *x)`
Retrieves a registered object's name.
- `void * jit_object_attach (t_symbol *s, void **x)`
Attaches an object as a client of a named server object for notification.
- `t_jit_err jit_object_detach (t_symbol *s, void *x)`
Detaches a client object from a named server object.
- `t_jit_err jit_object_notify (void *x, t_symbol *s, void *data)`
Notifies all client objects for a named server object.
- `t_jit_err jit_object_importattrs (void *x, t_symbol *s, long argc, t_atom *argv)`
Imports object attributes from an XML file.
- `t_jit_err jit_object_exportattrs (void *x, t_symbol *s, long argc, t_atom *argv)`
Exports object attributes to an XML file.
- `t_jit_err jit_object_exportsummary (void *x, t_symbol *s, long argc, t_atom *argv)`
Exports object summary to an XML file.

33.72.1 Function Documentation

33.72.1.1 `void* jit_object_attach (t_symbol * s, void * x)`

Attaches an object as a client of a named server object for notification.

Parameters

`s` name of server object
`x` client object pointer

Returns

If successful, server object pointer. Otherwise NULL.

33.72.1.2 `void* jit_object_attr_get (void * x, t_symbol * attrname)`

Retrieves an object's attribute pointer specified by attribute name.

Parameters

`x` object pointer
`attrname` attribute name

Returns

x attribute object pointer

33.72.1.3 long jit_object_attr_usercanget (void * *x*, t_symbol * *s*)

Determines if an object attribute is user gettable.

Parameters

x object pointer

s attribute name

Returns

1 if gettable, 0 if not gettable

33.72.1.4 long jit_object_attr_usercanset (void * *x*, t_symbol * *s*)

Determines if an object attribute is user settable.

Parameters

x object pointer

s attribute name

Returns

1 if settable, 0 if not settable

33.72.1.5 void* jit_object_class (void * *x*)

Retrieves an object's class pointer.

Parameters

x object pointer

Returns

class pointer

33.72.1.6 t_symbol* jit_object_classname (void * *x*)

Retrieves an object's class name.

Parameters

x object pointer

Returns

class name t_symbol pointer

33.72.1.7 long jit_object_classname_compare (void **x*, t_symbol **name*)

Compares object's class name with the name provided.

Parameters

- x* object pointer
- name* name to compare with class name

Returns

1 if equal, 0 if not equal

33.72.1.8 t_jit_err jit_object_detach (t_symbol **s*, void **x*)

Detaches a client object from a named server object.

Parameters

- s* name of server object
- x* client object pointer

Returns

t_jit_err error code

33.72.1.9 t_jit_err jit_object_exportattrs (void **x*, t_symbol **s*, long *argc*, t_atom **argv*)

Exports object attributes to an XML file.

Parameters

- x* object pointer
- s* ignored
- argc* argument count
- argv* argument vector

Returns

t_jit_err error code

33.72.1.10 t_jit_err jit_object_exportsummary (void **x*, t_symbol **s*, long *argc*, t_atom **argv*)

Exports object summary to an XML file.

Warning

Currently this function does nothing, but is reserved for future use.

Parameters

- x* object pointer

s ignored
argc argument count
argv argument vector

Returns

`t_jit_err` error code

33.72.1.11 void * jit_object_findregistered (t_symbol * s)

Retrieves a registered object associated with name.

Parameters

s registered name

Returns

If successful, object pointer. Otherwise NULL.

33.72.1.12 t_symbol* jit_object_findregisteredbyptr (void * x)

Retrieves a registered object's name.

Parameters

x object pointer

Returns

If successful, `t_symbol` pointer name. Otherwise NULL.

33.72.1.13 t_jit_err jit_object_free (void * x)

Frees an object.

Parameters

x object pointer

Returns

`t_jit_err` error code

33.72.1.14 method jit_object_getmethod (void * x, t_symbol * s)

Retrieves an object method specified by method name.

Parameters

x object pointer

s method name

Returns

method

33.72.1.15 `t_jit_err jit_object_importattrs (void *x, t_symbol *s, long argc, t_atom *argv)`

Imports object attributes from an XML file.

Parameters

x object pointer

s ignored

argc argument count

argv argument vector

Returns

`t_jit_err` error code

33.72.1.16 `void * jit_object_method (void *x, t_symbol *s, ...)`

Calls an object method specified by method name.

This operation is untyped, and the contents of the stack following the method name argument are blindly passed to the method called.

Parameters

x object pointer

s method name

... untyped arguments passed on to the method

Warning

It is important to know any necessary arguments for untyped constructors such as those used by `jit_matrix` or `jit_attr_offset`.

Returns

method dependent, but uses `void *` as a super type.

33.72.1.17 `t_symbol* jit_object_method_argsafe_get (void *x, t_symbol *s)`

Checks to see if symbol is safe to call as an attribute style argument.

Parameters

x object pointer

s name as used via argument

Returns

If successful, name of method to map the argument name to. Otherwise, NULL.

33.72.1.18 void* jit_object_method_typed (void * *x*, t_symbol * *s*, long *ac*, t_atom * *av*, t_atom * *rv*)

Calls a typed object method specified by method name.

This operation only supports methods which are typed--i.e. it cannot be used to call private, untyped A_CANT methods.

Parameters

- x* object pointer
- s* method name
- ac* argument count
- av* argument vector
- rv* return value for A_GIMMEBACK methods

Returns

method dependent, but uses void * as a super type.

33.72.1.19 void * jit_object_new (t_symbol * *classname*, ...)

Instantiates an object specified by class name.

This function may be used to create instances of any Jitter object.

Parameters

- classname* class name
- ... untyped arguments passed on to the constructor

Warning

It is important to know any necessary arguments for untyped constructors such as those used by jit_matrix or jit_attr_offset.

Returns

If successful, a valid object pointer. Otherwise, NULL.

33.72.1.20 t_jit_err jit_object_notify (void * *x*, t_symbol * *s*, void * *data*)

Notifies all client objects for a named server object.

Parameters

- x* server object pointer
- s* notification message
- data* message specific data

Returns

t_jit_err error code

33.72.1.21 void* jit_object_register (void *x, t_symbol *s)

Registers an object in the named object registry.

Parameters

- x* object pointer
- s* object name

Returns

- object pointer

Warning

It is important to use the object pointer returned by jit_object_register, since if there is an existing object with the same name and class, it could free the input object and pass back a reference to the previously defined object.

33.72.1.22 t_jit_err jit_object_unregister (void *x)

Unregisters an object from the named object registry.

Parameters

- x* object pointer

Returns

- t_jit_err error code

33.73 Miscellaneous Utility Module

Collaboration diagram for Miscellaneous Utility Module:



Functions

- float [swapf32](#) (float f)
Byte swaps 32 bit floating point number.
- double [swapf64](#) (double f)
Byte swaps 64 bit floating point number.
- void [jit_global_critical_enter](#) (void)
Enters the global Jitter critical region.
- void [jit_global_critical_exit](#) (void)
Exits the global Jitter critical region.
- void [jit_error_sym](#) (void *x, [t_symbol](#) *s)
Sends symbol based error message to Max console (safe from all threads).
- void [jit_error_code](#) (void *x, [t_jit_err](#) v)
Sends error code based error message to Max console (safe from all threads).
- void [jit_post_sym](#) (void *x, [t_symbol](#) *s)
Sends symbol based message to Max console (safe from all threads).
- [t_jit_err jit_err_from_max_err](#) ([t_max_err](#) err)
Converts Max style error codes to Jitter style error codes.
- void [jit_rand_setseed](#) (long n)
Sets global random number generator seed.
- long [jit_rand](#) (void)
Generates a random value as a signed long integer.

33.73.1 Function Documentation

33.73.1.1 [t_jit_err jit_err_from_max_err](#) ([t_max_err](#) err)

Converts Max style error codes to Jitter style error codes.

Parameters

err Max error code

Returns

`t_jit_err` error code

33.73.1.2 void jit_error_code (void **x*, t_jit_err *v*)

Sends error code based error message to Max console (safe from all threads).

Parameters

x object pointer

v error code

33.73.1.3 void jit_error_sym (void **x*, t_symbol **s*)

Sends symbol based error message to Max console (safe from all threads).

Parameters

x object pointer

s error message symbol

33.73.1.4 void jit_global_critical_enter (void)

Enters the global Jitter critical region.

This function is useful for simple protection of thread sensitive operations. However, it may be too broad a lock, as it prevents any other operations that use the global critical region from working. For more localized control, I would suggest using either MaxMSP's systhread API or the platform specific locking mechanisms however, be sensitive to the possibility deadlock when locking code which calls code which may require the locking off unknown resources.

33.73.1.5 void jit_global_critical_exit (void)

Exits the global Jitter critical region.

This function is useful for simple protection of thread sensitive operations. However, it may be too broad a lock, as it prevents any other operations that use the global critical region from working. For more localized control, I would suggest using either MaxMSP's systhread API or the platform specific locking mechanisms however, be sensitive to the possibility deadlock when locking code which calls code which may require the locking off unknown resources.

33.73.1.6 void jit_post_sym (void **x*, t_symbol **s*)

Sends symbol based message to Max console (safe from all threads).

Parameters

x object pointer

s message symbol

33.73.1.7 long jit_rand (void)

Generates a random value as a signed long integer.

Returns

random value

33.73.1.8 void jit_rand_setseed (long *n*)

Sets global random number generator seed.

Parameters

n seed

33.73.1.9 float swapf32 (float *f*)

Byte swaps 32 bit floating point number.

Parameters

f input float

Returns

byte swapped float

33.73.1.10 double swapf64 (double *f*)

Byte swaps 64 bit floating point number.

Parameters

f input double

Returns

byte swapped double

33.74 Linked List Module

Collaboration diagram for Linked List Module:



Functions

- `void * jit_linklist_new (void)`
Constructs instance of t_jit_linklist.
- `long jit_linklist_getsize (t_jit_linklist *x)`
Retrieves the linked list size.
- `void * jit_linklist_getindex (t_jit_linklist *x, long index)`
Retrieves the object at the specified list index.
- `long jit_linklist_objptr2index (t_jit_linklist *x, void *p)`
Retrieves the list index for an object pointer.
- `long jit_linklist_makearray (t_jit_linklist *x, void **a, long max)`
Flatten the linked list into an array.
- `long jit_linklist_insertindex (t_jit_linklist *x, void *o, long index)`
Insert object at specified index.
- `long jit_linklist_append (t_jit_linklist *x, void *o)`
Append object to the end of the linked list.
- `long jit_linklist_deleteindex (t_jit_linklist *x, long index)`
Delete object at specified index, freeing the object.
- `long jit_linklist_chuckindex (t_jit_linklist *x, long index)`
Remove object at specified index, without freeing the object.
- `void jit_linklist_clear (t_jit_linklist *x)`
Clears the linked list, freeing all objects in list.
- `void jit_linklist_chuck (t_jit_linklist *x)`
Removes all objects from the linked list, without freeing any objects in list.
- `void jit_linklist_reverse (t_jit_linklist *x)`
Reverses the order of objects in the linked list.
- `void jit_linklist_rotate (t_jit_linklist *x, long i)`
Rotates the order of objects in the linked list, by the specified number of indeces.
- `void jit_linklist_shuffle (t_jit_linklist *x)`

Randomizes the order of objects in the linked list.

- void [jit_linklist_swap](#) (*t_jit_linklist *x*, long *a*, long *b*)
Swap list location of the indeces specified.
- void [jit_linklist_findfirst](#) (*t_jit_linklist *x*, void ***o*, long *cmpfn(void *, void *)*, void **cmpdata*)
Retrieves the first object that satisfies the comparison function.
- void [jit_linklist_findall](#) (*t_jit_linklist *x*, *t_jit_linklist **out*, long *cmpfn(void *, void *)*, void **cmpdata*)
Retrieves a linked list of all objects that satisfy the comparison function.
- long [jit_linklist_findcount](#) (*t_jit_linklist *x*, long *cmpfn(void *, void *)*, void **cmpdata*)
Retrieves the number of objects that satisfy the comparison function.
- void [jit_linklist_methodall](#) (*t_jit_linklist *x*, *t_symbol *s,...*)
Calls a method on all objects in linked list.
- void * [jit_linklist_methodindex](#) (*t_jit_linklist *x*, long *i*, *t_symbol *s,...*)
Calls a method on the object at the specified index.
- void [jit_linklist_sort](#) (*t_jit_linklist *x*, long *cmpfn(void *, void *)*)
Sorts linked list based on the provided comparison function.

33.74.1 Function Documentation

33.74.1.1 long jit_linklist_append (*t_jit_linklist *x*, void **o*)

Append object to the end of the linked list.

Parameters

- x* *t_jit_linklist* object pointer
- o* object pointer

Returns

new list length, or -1 if unsuccessful

Warning

While exported, it is recommend to use *jit_object_method* to call methods on an object when the object may not be an instance of *t_jit_linklist*, but instead an object that supports some portion of the *t_jit_linklist* interface. One instance where this is the case is inside of a MOP *matrix_calc* method, where the arguments can be either an instance of *t_jit_linklist*, or *t_jit_matrix* which has a *getindex* method.

33.74.1.2 void jit_linklist_chuck (t_jit_linklist * *x*)

Removes all objects from the linked list, without freeing any objects in list.

To remove all objects from the linked list, reeing the objects, use the jit_linklist_clear method.

Parameters

x t_jit_linklist object pointer

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit_linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.3 long jit_linklist_chuckindex (t_jit_linklist * *x*, long *index*)

Remove object at specified index, without freeing the object.

This method will not free the object. To remove from the linked list and free the object, use the jit_linklist_deleteindex method.

Parameters

x t_jit_linklist object pointer

index index to remove (zero based)

Returns

index removed, or -1 if unsuccessful

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit_linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.4 void jit_linklist_clear (t_jit_linklist * *x*)

Clears the linked list, freeing all objects in list.

To remove all elements from the linked list without freeing the objects, use the jit_linklist_chuck method.

Parameters

x t_jit_linklist object pointer

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit_linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.5 long jit_linklist_deleteindex (t_jit_linklist **x*, long *index*)

Delete object at specified index, freeing the object.

To remove from the linked list without freeing the object, use the jit_linklist_chuckindex method.

Parameters

x t_jit_linklist object pointer
index index to delete (zero based)

Returns

index deleted, or -1 if unsuccessful

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit_linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.6 void jit_linklist_findall (t_jit_linklist **x*, t_jit_linklist *out*, long *cmpfnvoid* *, void *, void * *cmpdata*)**

Retrieves a linked list of all objects that satisfy the comparison function.

Parameters

x t_jit_linklist object pointer
out pointer to linked list containing all objects found (set to NULL, if not found)
cmpfn comparison function pointer (should returns 1 if object matches data, otherwise 0)
cmpdata opaque data used in comparison function

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit_linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.7 long jit_linklist_findcount (t_jit_linklist **x*, long *cmpfnvoid* *, void *, void * *cmpdata*)

Retrieves the number of objects that satisfy the comparison function.

Parameters

x t_jit_linklist object pointer
cmpfn comparison function pointer (should returns 1 if object matches data, otherwise 0)
cmpdata opaque data used in comparison function

Returns

number object objects that satisfy the comparison function

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit-linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.8 void jit_linklist_findfirst (t_jit_linklist * *x*, void ** *o*, long *cmpfn*void *, void *, void * *cmpdata*)

Retrieves the first object that satisfies the comparison function.

Parameters

x t_jit_linklist object pointer

o pointer to object pointer found (set to NULL, if not found)

cmpfn comparison function pointer (should returns 1 if object matches data, otherwise 0)

cmpdata opaque data used in comparison function

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit-linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.9 void* jit_linklist_getindex (t_jit_linklist * *x*, long *index*)

Retrieves the object at the specified list index.

Parameters

x t_jit_linklist object pointer

index list index ()

Returns

object pointer

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit-linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.10 long jit_linklist_getsize (t_jit_linklist * *x*)

Retrieves the linked list size.

Parameters

x t_jit_linklist object pointer

Returns

linked list size

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit_linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.11 long jit_linklist_insertindex (t_jit_linklist * *x*, void * *o*, long *index*)

Insert object at specified index.

Parameters

x t_jit_linklist object pointer

o object pointer

index index (zero based)

Returns

index inserted at, or -1 if unsuccessful

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit_linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.12 long jit_linklist_makearray (t_jit_linklist * *x*, void ** *a*, long *max*)

Flatten the linked list into an array.

Parameters

x t_jit_linklist object pointer

a array pointer

max maximum array size

Returns

number of object pointers copied into array

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit_linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.13 void jit_linklist_methodall (t_jit_linklist *x, t_symbol *s, ...)

Calls a method on all objects in linked list.

Equivalent to calling jit_object_method on the object at each index.

Parameters

- x* t_jit_linklist object pointer
- s* method name
- ... untyped arguments

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit_linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.14 void* jit_linklist_methodindex (t_jit_linklist *x, long i, t_symbol *s, ...)

Calls a method on the object at the specified index.

Equivalent to calling jit_object_method on the object.

Parameters

- x* t_jit_linklist object pointer
- i* index
- s* method name
- ... untyped arguments

Returns

method return value

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit_linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.15 void* jit_linklist_new (void)

Constructs instance of t_jit_linklist.

Returns

t_jit_linklist object pointer

Warning

While exported, it is recommend to use jit_object_new to construct a t_jit_linklist object.

33.74.1.16 long jit_linklist_objptr2index (t_jit_linklist * x, void * p)

Retrieves the list index for an object pointer.

Parameters

x t_jit_linklist object pointer

p object pointer

Returns

object's list index (zero based), or -1 if not present

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit_linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.17 void jit_linklist_reverse (t_jit_linklist * x)

Reverses the order of objects in the linked list.

Parameters

x t_jit_linklist object pointer

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit_linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.18 void jit_linklist_rotate (t_jit_linklist * x, long i)

Rotates the order of objects in the linked list, by the specified number of indeces.

Parameters

x t_jit_linklist object pointer

i rotation index count

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit-linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.19 void jit_linklist_shuffle (t_jit_linklist *x)

Randomizes the order of objects in the linked list.

Parameters

x t_jit_linklist object pointer

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit-linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.20 void jit_linklist_sort (t_jit_linklist *x, long cmpfnvoid *, void *)

Sorts linked list based on the provided comparison function.

Parameters

x t_jit_linklist object pointer

cmpfn comparison function pointer (returns 0 if a>b, otherwise 1)

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit-linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.74.1.21 void jit_linklist_swap (t_jit_linklist *x, long a, long b)

Swap list location of the indeces specified.

Parameters

x t_jit_linklist object pointer

a index a

b index b

Warning

While exported, it is recommend to use jit_object_method to call methods on an object when the object may not be an instance of t_jit_linklist, but instead an object that supports some portion of the t_jit_linklist interface. One instance where this is the case is inside of a MOP matrix_calc method, where the arguments can be either an instance of t_jit_linklist, or t_jit_matrix which has a getindex method.

33.75 Math Module

Collaboration diagram for Math Module:



Functions

- double [jit_math_cos](#) (double x)
Calculates the cosine.
- double [jit_math_sin](#) (double x)
Calculates the sine.
- double [jit_math_tan](#) (double x)
Calculates the tangent.
- double [jit_math_acos](#) (double x)
Calculates the arccosine.
- double [jit_math_asin](#) (double x)
Calculates the arcsine.
- double [jit_math_atan](#) (double x)
Calculates the arctangent.
- double [jit_math_atan2](#) (double y, double x)
Calculates the four quadrant arctangent.
- double [jit_math_cosh](#) (double x)
Calculates the hyperbolic cosine.
- double [jit_math_sinh](#) (double x)
Calculates the hyperbolic sine.
- double [jit_math_tanh](#) (double x)
Calculates the hyperbolic tangent.
- double [jit_math_acosh](#) (double x)
Calculates the hyperbolic arccosine.
- double [jit_math_asinh](#) (double x)
Calculates the hyperbolic arcsine.
- double [jit_math_atanh](#) (double x)
Calculates the hyperbolic arctangent.
- double [jit_math_exp](#) (double x)

Calculates the exponent.

- double `jit_math_expm1` (double x)

Calculates the exponent minus 1.

- double `jit_math_exp2` (double x)

Calculates the exponent base 2.

- double `jit_math_log` (double x)

Calculates the logarithm.

- double `jit_math_log2` (double x)

Calculates the logarithm base 2.

- double `jit_math_log10` (double x)

Calculates the logarithm base 10.

- double `jit_math_hypot` (double x, double y)

Calculates the hypotenuse.

- double `jit_math_pow` (double x, double y)

Calculates x raised to the y power.

- double `jit_math_sqrt` (double x)

Calculates the square root.

- double `jit_math_ceil` (double x)

Calculates the ceiling.

- double `jit_math_floor` (double x)

Calculates the floor.

- double `jit_math_round` (double x)

Rounds the input.

- double `jit_math_trunc` (double x)

Truncates the input.

- double `jit_math_fmod` (double x, double y)

Calculates the floating point x modulo y.

- double `jit_math_fold` (double x, double lo, double hi)

Calculates the fold of x between lo and hi.

- double `jit_math_wrap` (double x, double lo, double hi)

Calculates the wrap of x between lo and hi.

- double `jit_math_j1_0` (double x)

Calcuates the j1_0 Bessel function.

- `double jit_math_p1 (double x)`
Calculates the p1 Bessel function.
- `double jit_math_q1 (double x)`
Calculates the q1 Bessel function.
- `double jit_math_j1 (double x)`
Calculates the j1 Bessel function.
- `unsigned long jit_math_roundup_poweroftwo (unsigned long x)`
Rounds up to the nearest power of two.
- `long jit_math_is_finite (float v)`
Checks if input is finite.
- `long jit_math_is_nan (float v)`
Checks if input is not a number (NaN).
- `long jit_math_is_valid (float v)`
Checks if input is both finite and a number.
- `long jit_math_is_poweroftwo (long x)`
Checks if input is a power of two.
- `float jit_math_fast_sqrt (float n)`
Calculates the square root by fast approximation.
- `float jit_math_fast_invsqrt (float x)`
Calculates the inverse square root by fast approximation.
- `float jit_math_fast_sin (float x)`
Calculates the sine by fast approximation.
- `float jit_math_fast_cos (float x)`
Calculates the cosine by fast approximation.
- `float jit_math_fast_tan (float x)`
Calculates the tangent by fast approximation.
- `float jit_math_fast_asin (float x)`
Calculates the arcsine by fast approximation.
- `float jit_math_fast_acos (float x)`
Calculates the arccosine by fast approximation.
- `float jit_math_fast_atan (float x)`
Calculates the arctangent by fast approximation.

33.75.1 Function Documentation

33.75.1.1 double jit_math_acos (double x)

Calculates the arccosine.

Parameters

x input

Returns

output

33.75.1.2 double jit_math_acosh (double x)

Calculates the hyperbolic arccosine.

Parameters

x input

Returns

output

33.75.1.3 double jit_math_asin (double x)

Calculates the arcsine.

Parameters

x input

Returns

output

33.75.1.4 double jit_math_asinh (double x)

Calculates the hyperbolic arcsine.

Parameters

x input

Returns

output

33.75.1.5 double jit_math_atan (double x)

Calculates the arctangent.

Parameters

x input

Returns

output

33.75.1.6 double jit_math_atan2 (double y , double x)

Calculates the four quadrant arctangent.

Parameters

y input

x input

Returns

output

33.75.1.7 double jit_math_atanh (double x)

Calculates the hyperbolic arctangent.

Parameters

x input

Returns

output

33.75.1.8 double jit_math_ceil (double x)

Calculates the ceiling.

Parameters

x input

Returns

output

33.75.1.9 double jit_math_cos (double x)

Calculates the cosine.

Parameters

x input

Returns

output

33.75.1.10 double jit_math_cosh (double x)

Calculates the hyperbolic cosine.

Parameters

x input

Returns

output

33.75.1.11 double jit_math_exp (double x)

Calculates the exponent.

Parameters

x input

Returns

output

33.75.1.12 double jit_math_exp2 (double x)

Calculates the exponent base 2.

Parameters

x input

Returns

output

33.75.1.13 double jit_math_expm1 (double x)

Calculates the exponent minus 1.

Parameters

x input

Returns

output

33.75.1.14 float jit_math_fast_acos (float x)

Calculates the arccosine by fast approximation.

Absolute error of 6.8e-05 for [0, 1]

Parameters

x input

Returns

output

33.75.1.15 float jit_math_fast_asin (float x)

Calculates the arcsine by fast approximation.

Absolute error of 6.8e-05 for [0, 1]

Parameters

x input

Returns

output

33.75.1.16 float jit_math_fast_atan (float x)

Calculates the arctangent by fast approximation.

Absolute error of 1.43-08 for [-1, 1]

Parameters

x input

Returns

output

33.75.1.17 float jit_math_fast_cos (float x)

Calculates the cosine by fast approximation.

Absolute error of 1.2e-03 for [0, PI/2]

Parameters

x input

Returns

output

33.75.1.18 float jit_math_fast_invsqrt (float x)

Calculates the inverse square root by fast approximation.

Parameters

x input

Returns

output

33.75.1.19 float jit_math_fast_sin (float x)

Calculates the sine by fast approximation.

Absolute error of 1.7e-04 for [0, PI/2]

Parameters

x input

Returns

output

33.75.1.20 float jit_math_fast_sqrt (float n)

Calculates the square root by fast approximation.

Parameters

n input

Returns

output

33.75.1.21 float jit_math_fast_tan (float *x*)

Calculates the tangent by fast approximation.

Absolute error of 1.9e-00 for [0, PI/4]

Parameters

x input

Returns

output

33.75.1.22 double jit_math_floor (double *x*)

Calculates the floor.

Parameters

x input

Returns

output

33.75.1.23 double jit_math_fmod (double *x*, double *y*)

Calculates the floating point x modulo y.

Parameters

x input

y input

Returns

output

33.75.1.24 double jit_math_fold (double *x*, double *lo*, double *hi*)

Calculates the fold of x between lo and hi.

Parameters

x input

lo lower bound

hi upper bound

Returns

output

33.75.1.25 double jit_math_hypot (double x , double y)

Calculates the hypotenuse.

Parameters

x input

y input

Returns

output

33.75.1.26 long jit_math_is_finite (float v)

Checks if input is finite.

Parameters

v input

Returns

1 if finite. Otherwise, 0.

33.75.1.27 long jit_math_is_nan (float v)

Checks if input is not a number (NaN).

Parameters

v input

Returns

1 if not a number. Otherwise, 0.

33.75.1.28 long jit_math_is_poweroftwo (long x)

Checks if input is a power of two.

Parameters

x input

Returns

1 if finite. Otherwise, 0.

33.75.1.29 long jit_math_is_valid (float v)

Checks if input is both finite and a number.

Parameters

v input

Returns

1 if valid. Otherwise, 0.

33.75.1.30 double jit_math_j1 (double x)

Calcuates the j1 Bessel function.

Parameters

x input

Returns

output

33.75.1.31 double jit_math_j1_0 (double x)

Calcuates the j1_0 Bessel function.

Parameters

x input

Returns

output

33.75.1.32 double jit_math_log (double x)

Calculates the logarithm.

Parameters

x input

Returns

output

33.75.1.33 double jit_math_log10 (double x)

Calculates the logarithm base 10.

Parameters

x input

Returns

output

33.75.1.34 double jit_math_log2 (double x)

Calculates the logarithm base 2.

Parameters

x input

Returns

output

33.75.1.35 double jit_math_p1 (double x)

Calcuates the p1 Bessel function.

Parameters

x input

Returns

output

33.75.1.36 double jit_math_pow (double x, double y)

Calculates x raised to the y power.

Parameters

x input

y input

Returns

output

33.75.1.37 double jit_math_q1 (double x)

Calculates the q1 Bessel function.

Parameters

x input

Returns

output

33.75.1.38 double jit_math_round (double x)

Rounds the input.

Parameters

x input

Returns

output

33.75.1.39 unsigned long jit_math_roundup_poweroftwo (unsigned long x)

Rounds up to the nearest power of two.

Parameters

x input

Returns

output

33.75.1.40 double jit_math_sin (double x)

Calculates the sine.

Parameters

x input

Returns

output

33.75.1.41 double jit_math_sinh (double x)

Calculates the hyperbolic sine.

Parameters

x input

Returns

output

33.75.1.42 double jit_math_sqrt (double x)

Calculates the square root.

Parameters

x input

Returns

output

33.75.1.43 double jit_math_tan (double x)

Calculates the tangent.

Parameters

x input

Returns

output

33.75.1.44 double jit_math_tanh (double x)

Calculates the hyperbolic tangent.

Parameters

x input

Returns

output

33.75.1.45 double jit_math_trunc (double *x*)

Truncates the input.

Parameters

x input

Returns

output

33.75.1.46 double jit_math_wrap (double *x*, double *lo*, double *hi*)

Calculates the wrap of *x* between *lo* and *hi*.

Parameters

x input

lo lower bound

hi upper bound

Returns

output

33.76 Matrix Module

Collaboration diagram for Matrix Module:



Functions

- `void jit_linklist_free (t_jit_linklist *x)`
Frees instance of t_jit_linklist.
- `void * jit_matrix_new (t_jit_matrix_info *info)`
Constructs instance of t_jit_matrix.
- `void * jit_matrix_newcopy (t_jit_matrix *copyme)`
Constructs instance of t_jit_matrix, copying from input.
- `t_jit_err jit_matrix_free (t_jit_matrix *x)`
Frees instance of t_jit_matrix.
- `t_jit_err jit_matrix_setinfo (t_jit_matrix *x, t_jit_matrix_info *info)`
Sets all attributes according to the t_jit_matrix_info struct provided.
- `t_jit_err jit_matrix_setinfo_ex (t_jit_matrix *x, t_jit_matrix_info *info)`
Sets all attributes according to the t_jit_matrix_info struct provided (including data flags).
- `t_jit_err jit_matrix_getinfo (t_jit_matrix *x, t_jit_matrix_info *info)`
Retrieves all attributes, copying into the t_jit_matrix_info struct provided.
- `t_jit_err jit_matrix_getdata (t_jit_matrix *x, void **data)`
Retrieves matrix data pointer.
- `t_jit_err jit_matrix_data (t_jit_matrix *x, void *data)`
Sets matrix data pointer.
- `t_jit_err jit_matrix_freedata (t_jit_matrix *x)`
Frees matrix's internal data pointer if an internal reference and sets to NULL.
- `t_jit_err jit_matrix_info_default (t_jit_matrix_info *info)`
Initializes matrix info struct to default values.
- `t_jit_err jit_matrix_clear (t_jit_matrix *x)`
Sets all cells in matrix to the zero.
- `t_jit_err jit_matrix_setcell1d (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`
Sets cell at index to the value provided.
- `t_jit_err jit_matrix_setcell2d (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Sets cell at index to the value provided.

- `t_jit_err jit_matrix_setcell3d (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Sets cell at index to the value provided.

- `t_jit_err jit_matrix_setplane1d (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Sets plane of cell at index to the value provided.

- `t_jit_err jit_matrix_setplane2d (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Sets plane of cell at index to the value provided.

- `t_jit_err jit_matrix_setplane3d (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Sets plane of cell at index to the value provided.

- `t_jit_err jit_matrix_setcell (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Sets cell at index to the value provided.

- `t_jit_err jit_matrix_getcell (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv, long *rac, t_atom **rav)`

Gets cell at index to the value provided.

- `t_jit_err jit_matrix_setall (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Sets all cells to the value provided.

- `t_jit_err jit_matrix_fillplane (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Sets the plane specified in all cells to the value provided.

- `t_jit_err jit_matrix_togworld (t_jit_matrix *x, GWorldPtr gp, t_gworld_conv_info *gcinfo)`

Copies Jitter matrix data to GWorld data.

- `t_jit_err jit_matrix_fromgworld (t_jit_matrix *x, GWorldPtr gp, t_gworld_conv_info *gcinfo)`

Copies Jitter matrix data from GWorld data.

- `t_jit_err jit_matrix_frommatrix (t_jit_matrix *dst_matrix, t_jit_matrix *src_matrix, t_matrix_conv_info *mcinfo)`

Copies Jitter matrix data from another matrix.

- `t_jit_err jit_matrix_op (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Applies unary or binary operator to matrix See Jitter user documentation for more information.

- `t_jit_err jit_matrix_exprfill (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Fills cells according to the jit.expr expression provided.

- `t_jit_err jit_matrix_jit_gl_texture (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Copies texture information to matrix.

33.76.1 Function Documentation

33.76.1.1 void jit_linklist_free (*t_jit_linklist* **x*)

Frees instance of *t_jit_linklist*.

Parameters

x *t_jit_linklist* object pointer

Returns

t_jit_err error code

Warning

Use *jit_object_free* instead.

33.76.1.2 *t_jit_err jit_matrix_clear (*t_jit_matrix* **x*)*

Sets all cells in matrix to the zero.

See Jitter user documentation for more information.

Parameters

x *t_jit_matrix* object pointer

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via *jit_object_method* on an instance of *t_jit_matrix*.

33.76.1.3 *t_jit_err jit_matrix_data (*t_jit_matrix* **x*, *void* **data*)*

Sets matrix data pointer.

Parameters

x *t_jit_matrix* object pointer

data data pointer

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via *jit_object_method* on an instance of *t_jit_matrix*.

33.76.1.4 t_jit_err jit_matrix_exprfill (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)

Fills cells according to the jit.expr expression provided.

See Jitter user documentation for more information.

Parameters

x t_jit_matrix object pointer
s message symbol pointer
argc argument count
argv argument vector

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.5 t_jit_err jit_matrix_fillplane (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)

Sets the plane specified in all cells to the value provided.

See Jitter user documentation for more information.

Parameters

x t_jit_matrix object pointer
s message symbol pointer
argc argument count
argv argument vector

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.6 t_jit_err jit_matrix_free (t_jit_matrix *x)

Frees instance of t_jit_matrix.

Parameters

x t_jit_matrix object pointer

Returns

`t_jit_err` error code

Warning

Use `jit_object_free` instead.

33.76.1.7 `t_jit_err jit_matrix_freedata (t_jit_matrix *x)`

Frees matrix's internal data pointer if an internal reference and sets to NULL.

Parameters

`x` `t_jit_matrix` object pointer

Returns

`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

33.76.1.8 `t_jit_err jit_matrix_fromgworld (t_jit_matrix *x, GWorldPtr gp, t_gworld_conv_info *gcinfo)`

Copies Jitter matrix data from GWorld data.

Parameters

`x` `t_jit_matrix` object pointer

`gp` gworld pointer

`gcinfo` conversion information pointer

Returns

`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

33.76.1.9 `t_jit_err jit_matrix_frommatrix (t_jit_matrix *dst_matrix, t_jit_matrix *src_matrix, t_matrix_conv_info *mcinfo)`

Copies Jitter matrix data from another matrix.

Parameters

`dst_matrix` destination `t_jit_matrix` object pointer

src_matrix destination t_jit_matrix object pointer
mcinfo conversion information pointer

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.10 t_jit_err jit_matrix_getcell (t_jit_matrix **x*, t_symbol **s*, long *argc*, t_atom **argv*, long **rac*, t_atom ***rav*)

Gets cell at index to the value provided.

See Jitter user documentation for more information.

Parameters

x t_jit_matrix object pointer
s message symbol pointer
argc argument count
argv argument vector
rac return value atom count
rav return value atom vector

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.11 t_jit_err jit_matrix_getdata (t_jit_matrix **x*, void ***data*)

Retrieves matrix data pointer.

Parameters

x t_jit_matrix object pointer
data pointer to data pointer (set to NULL if matrix is not available)

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.12 `t_jit_err jit_matrix_getinfo (t_jit_matrix *x, t_jit_matrix_info *info)`

Retrieves all attributes, copying into the `t_jit_matrix_info` struct provided.

Parameters

x `t_jit_matrix` object pointer
info `t_jit_matrix_info` pointer

Returns

`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

33.76.1.13 `t_jit_err jit_matrix_info_default (t_jit_matrix_info *info)`

Initializes matrix info struct to default values.

Parameters

info `t_jit_matrix_info` struct pointer

Returns

`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

33.76.1.14 `t_jit_err jit_matrix_jit_gl_texture (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Copies texture information to matrix.

See Jitter user documentation for more information.

Parameters

x `t_jit_matrix` object pointer
s message symbol pointer
argc argument count
argv argument vector

Returns

`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

33.76.1.15 void * jit_matrix_new (t_jit_matrix_info * *info*)

Constructs instance of t_jit_matrix.

Parameters

info t_jit_matrix_info struct pointer

Returns

t_jit_matrix object pointer

Warning

This function is not exported, but is provided for reference when calling via jit_object_new.

33.76.1.16 void * jit_matrix_newcopy (t_jit_matrix * *copyme*)

Constructs instance of t_jit_matrix, copying from input.

Parameters

copyme t_jit_matrix object pointer

Returns

t_jit_matrix object pointer

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.17 t_jit_err jit_matrix_op (t_jit_matrix * *x*, t_symbol * *s*, long *argc*, t_atom * *argv*)

Applies unary or binary operator to matrix See Jitter user documentation for more information.

Parameters

x t_jit_matrix object pointer

s message symbol pointer

argc argument count

argv argument vector

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.18 t_jit_err jit_matrix_setall (t_jit_matrix **x*, t_symbol **s*, long *argc*, t_atom **argv*)

Sets all cells to the value provided.

See Jitter user documentation for more information.

Parameters

x t_jit_matrix object pointer
s message symbol pointer
argc argument count
argv argument vector

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.19 t_jit_err jit_matrix_setcell (t_jit_matrix **x*, t_symbol **s*, long *argc*, t_atom **argv*)

Sets cell at index to the value provided.

See Jitter user documentation for more information.

Parameters

x t_jit_matrix object pointer
s message symbol pointer
argc argument count
argv argument vector

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.20 t_jit_err jit_matrix_setcell1d (t_jit_matrix **x*, t_symbol **s*, long *argc*, t_atom **argv*)

Sets cell at index to the value provided.

See Jitter user documentation for more information.

Parameters

x t_jit_matrix object pointer

s message symbol pointer

argc argument count

argv argument vector

Returns

`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

33.76.1.21 `t_jit_err jit_matrix_setcell2d (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Sets cell at index to the value provided.

See Jitter user documentation for more information.

Parameters

x `t_jit_matrix` object pointer

s message symbol pointer

argc argument count

argv argument vector

Returns

`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

33.76.1.22 `t_jit_err jit_matrix_setcell3d (t_jit_matrix *x, t_symbol *s, long argc, t_atom *argv)`

Sets cell at index to the value provided.

See Jitter user documentation for more information.

Parameters

x `t_jit_matrix` object pointer

s message symbol pointer

argc argument count

argv argument vector

Returns

`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

33.76.1.23 t_jit_err jit_matrix_setinfo (t_jit_matrix **x*, t_jit_matrix_info **info*)

Sets all attributes according to the [t_jit_matrix_info](#) struct provided.

Parameters

x t_jit_matrix object pointer
info t_jit_matrix_info pointer

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.24 t_jit_err jit_matrix_setinfo_ex (t_jit_matrix **x*, t_jit_matrix_info **info*)

Sets all attributes according to the [t_jit_matrix_info](#) struct provided (including data flags).

Parameters

x t_jit_matrix object pointer
info t_jit_matrix_info pointer

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.25 t_jit_err jit_matrix_setplane1d (t_jit_matrix **x*, t_symbol **s*, long *argc*, t_atom **argv*)

Sets plane of cell at index to the value provided.

See Jitter user documentation for more information.

Parameters

x t_jit_matrix object pointer
s message symbol pointer
argc argument count
argv argument vector

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.26 t_jit_err jit_matrix_setplane2d (t_jit_matrix **x*, t_symbol **s*, long *argc*, t_atom **argv*)

Sets plane of cell at index to the value provided.

See Jitter user documentation for more information.

Parameters

x t_jit_matrix object pointer

s message symbol pointer

argc argument count

argv argument vector

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.27 t_jit_err jit_matrix_setplane3d (t_jit_matrix **x*, t_symbol **s*, long *argc*, t_atom **argv*)

Sets plane of cell at index to the value provided.

See Jitter user documentation for more information.

Parameters

x t_jit_matrix object pointer

s message symbol pointer

argc argument count

argv argument vector

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

33.76.1.28 `t_jit_err jit_matrix_togworld (t_jit_matrix *x, GWorldPtr gp, t_gworld_conv_info *gcinfo)`

Copies Jitter matrix data to GWorld data.

Parameters

x `t_jit_matrix` object pointer
gp gworld pointer
gcinfo conversion information pointer

Returns

`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

33.77 Max Wrapper Module

Collaboration diagram for Max Wrapper Module:



Functions

- `void max_jit_attr_set (void *x, t_symbol *s, short ac, t_atom *av)`
Sets attribute value.
- `t_jit_err max_jit_attr_get (void *x, t_symbol *s, long *ac, t_atom **av)`
Retrieves attribute value.
- `void max_jit_attr_getdump (void *x, t_symbol *s, short argc, t_atom *argv)`
Retrieves attribute value and sends out dump outlet.
- `long max_jit_attr_args_offset (short ac, t_atom *av)`
Determines argument offset to first attribute argument.
- `void max_jit_attr_args (void *x, short ac, t_atom *av)`
Processes attribtue arguments.
- `void max_jit_classex_standard_wrap (void *mclass, void *jclass, long flags)`
Adds standard Jitter methods, as well as public methods and attributes of the specified Jitter class.
- `void max_addmethod_defer (method m, char *s)`
Adds method to Max class that calls defer rather than the method directly.
- `void max_addmethod_defer_low (method m, char *s)`
Adds method to Max class that calls defer_low rather than the method directly.
- `void max_addmethod_usurp (method m, char *s)`
Adds method to Max class that uses the usurp mechanism to execute method at low priority without backlog.
- `void max_addmethod_usurp_low (method m, char *s)`
Adds method to Max class that uses the usurp mechanism to execute method at low priority without backlog.
- `void * max_jit_classex_setup (long oboffset)`
Allocates and initializes special t_max_jit_classex data, used by the Max wrapper class.
- `t_jit_err max_jit_classex_addattr (void *x, void *attr)`
Adds an attribute to the Max wrapper class.
- `void * max_jit_obex_new (void *mc, t_symbol *classname)`
Allocates an initializes a new Max wrapper object instance.
- `void max_jit_obex_free (void *x)`

Frees additional resources for the Max wrapper object instance.

- `t_jit_err max_jit_obex_attr_set (void *x, t_symbol *s, long ac, t_atom *av)`
Sets an attribute of the Max wrapper or the wrapped Jitter object.
- `t_jit_err max_jit_obex_attr_get (void *x, t_symbol *s, long *ac, t_atom **av)`
Retrieves an attribute of the Max wrapper or the wrapped Jitter object.
- `void * max_jit_obex_jitob_get (void *x)`
Retrieves the wrapped Jitter object from a Max wrapper object.
- `void max_jit_obex_jitob_set (void *x, void *jitob)`
Sets the wrapped Jitter object for a Max wrapper object.
- `long max_jit_obex_inletnumber_get (void *x)`
Retrieves the current inlet number used by inlet proxies.
- `void max_jit_obex_inletnumber_set (void *x, long inletnumber)`
Sets the current inlet number used by inlet proxies.
- `t_jit_err max_jit_obex_proxy_new (void *x, long c)`
Creates a new proxy inlet.
- `void max_jit_obex_dumpout_set (void *x, void *outlet)`
Sets the Max wrapper object's dump outlet's outlet pointer.
- `void * max_jit_obex_dumpout_get (void *x)`
Retrieves the Max wrapper object's dump outlet's outlet pointer.
- `void max_jit_obex_dumpout (void *x, t_symbol *s, short argc, t_atom *argv)`
Sends a message and arguments out the dump outlet.
- `void * max_jit_obex_adornment_get (void *x, t_symbol *classname)`
Retrieves Max wrapper object adornment specified by class name.
- `void max_jit_obex_gimmeback (void *x, t_symbol *s, long ac, t_atom *av)`
Calls gimmeback methods and frees any return value.
- `void max_jit_obex_gimmeback_dumpout (void *x, t_symbol *s, long ac, t_atom *av)`
Calls gimmeback methods and outputs any return value through the Max wrapper class' dump outlet.

33.77.1 Function Documentation

33.77.1.1 void max_addmethod_defer (method m, char * s)

Adds method to Max class that calls defer rather than the method directly.

To prevent sequencing problems which arize through the use of defer, rather than defer_low, you should instead use the `max_addmethod_defer_low` function.

Parameters

m method (function pointer)

s method name

33.77.1.2 void max_addmethod_defer_low (method *m*, char * *s*)

Adds method to Max class that calls defer_low rather than the method directly.

Parameters

m method (function pointer)

s method name

33.77.1.3 void max_addmethod_usurp (method *m*, char * *s*)

Adds method to Max class that uses the usurp mechanism to execute method at low priority without backlog.

Equivalent to max_addmethod_usurp_low function.

Parameters

m method (function pointer)

s method name

33.77.1.4 void max_addmethod_usurp_low (method *m*, char * *s*)

Adds method to Max class that uses the usurp mechanism to execute method at low priority without backlog.

Parameters

m method (function pointer)

s method name

33.77.1.5 void max_jit_attr_args (void * *x*, short *ac*, t_atom * *av*)

Processes attribtue arguments.

Parameters

x Max wrapper object pointer

ac argument count

av argument vector

33.77.1.6 long max_jit_attr_args_offset (short *ac*, t_atom * *av*)

Determines argument offset to first attribute argument.

Parameters

- ac* argument count
- av* argument vector

Returns

- argument offset

33.77.1.7 t_jit_err max_jit_attr_get (void * *x*, t_symbol * *s*, long * *ac*, t_atom ** *av*)

Retrieves attribute value.

Parameters

- x* Max wrapper object pointer
- s* attribute name
- ac* pointer atom count
- av* pointer atom vector

Returns

- t_jit_err error code

33.77.1.8 void max_jit_attr_getdump (void * *x*, t_symbol * *s*, short *argc*, t_atom * *argv*)

Retrieves attribute value and sends out dump outlet.

Parameters

- x* Max wrapper object pointer
- s* attribute name
- argc* argument count (ignored)
- argv* argument vector (ignored)

33.77.1.9 void max_jit_attr_set (void * *x*, t_symbol * *s*, short *ac*, t_atom * *av*)

Sets attribute value.

Parameters

- x* Max wrapper object pointer
- s* attribute name
- ac* atom count
- av* atom vector

33.77.1.10 long max_jit_classex_addattr (void * *x*, void * *attr*)

Adds an attribute to the Max wrapper class.

Parameters

x pointer to t_max_jit_classex data (opaque)
attr attribute object pointer

Returns

t_jit_err error code

33.77.1.11 void * max_jit_classex_setup (long *oboffset*)

Allocates and initializes special t_max_jit_classex data, used by the Max wrapper class.

Parameters

oboffset object struct byte offset to obex pointer

Returns

pointer to t_max_jit_classex data (opaque)

33.77.1.12 void max_jit_classex_standard_wrap (void * *mclass*, void * *jclass*, long *flags*)

Adds standard Jitter methods, as well as public methods and attributes of the specified Jitter class.

This includes the following public methods: getattributes, getstate, summary, importattrs, exportattrs; and the following private methods: dumpout, quickref, attr_getnames, attr_get, attr_gettarget, and attrindex.

Parameters

mclass Max wrapper class pointer
jclass jitter class pointer
flags reserved for future use (currently ignored)

33.77.1.13 void* max_jit_obex_adornment_get (void * *x*, t_symbol * *classname*)

Retrieves Max wrapper object adornment specified by class name.

Typically used for accessing the jit_mop adornment for MOP Max wrapper objects.

Parameters

x Max wrapper object pointer
classname adornment classname

Returns

adornment pointer

33.77.1.14 t_jit_err max_jit_obex_attr_get (void *x, t_symbol *s, long *ac, t_atom **av)

Retrievnes an attribute of the Max wrapper or the wrapped Jitter object.

Parameters

- x* Max wrapper object pointer
- s* attribute name
- ac* pointer to atom count
- av* pointer to atom vector

Returns

- t_jit_error error code

33.77.1.15 t_jit_err max_jit_obex_attr_set (void *x, t_symbol *s, long ac, t_atom *av)

Sets an attribute of the Max wrapper or the wrapped Jitter object.

Parameters

- x* Max wrapper object pointer
- s* attribute name
- ac* atom count
- av* atom vector

Returns

- t_jit_error error code

33.77.1.16 void max_jit_obex_dumpout (void *x, t_symbol *s, short argc, t_atom *argv)

Sends a message and arguments out the dump outlet.

This message is equivalent to calling outlet_anything with the outlet returned by max_jit_obex_dumpout_get.

Parameters

- x* Max wrapper object pointer
- s* message symbol
- argc* argument count
- argv* argument vector

33.77.1.17 void* max_jit_obex_dumpout_get (void *x)

Retrieves the Max wrapper object's dump outlet's outlet pointer.

Parameters

x Max wrapper object pointer

Returns

dump outlet pointer

33.77.1.18 void max_jit_obex_dumpout_set (void * *x*, void * *outlet*)

Sets the Max wrapper object's dump outlet's outlet pointer.

Parameters

x Max wrapper object pointer

outlet dump outlet pointer

33.77.1.19 void max_jit_obex_free (void * *x*)

Frees additional resources for the Max wrapper object instance.

Parameters

x Max wrapper object pointer

33.77.1.20 void max_jit_obex_gimmeback (void * *x*, t_symbol * *s*, long *ac*, t_atom * *av*)

Calls gimmeback methods and frees any return value.

Parameters

x Max wrapper object pointer

s method name

ac argument count

av argument vector

33.77.1.21 void max_jit_obex_gimmeback_dumpout (void * *x*, t_symbol * *s*, long *ac*, t_atom * *av*)

Calls gimmeback methods and outputs any return value through the Max wrapper class' dump outlet.

Parameters

x Max wrapper object pointer

s method name

ac argument count

av argument vector

33.77.1.22 long max_jit_obex_inletnumber_get (void **x*)

Retrieves the current inlet number used by inlet proxies.

Parameters

x Max wrapper object pointer

Returns

current inlet index

33.77.1.23 void max_jit_obex_inletnumber_set (void **x*, long *inletnumber*)

Sets the current inlet number used by inlet proxies.

Warning

Typically not used outside jitlib.

Parameters

x Max wrapper object pointer

inletnumber inlet index

33.77.1.24 void* max_jit_obex_jitob_get (void **x*)

Retrieves the wrapped Jitter object from a Max wrapper object.

Parameters

x Max wrapper object pointer

Returns

Jitter object pointer

33.77.1.25 void max_jit_obex_jitob_set (void **x*, void **jitob*)

Sets the wrapped Jitter object for a Max wrapper object.

Parameters

x Max wrapper object pointer

jitob Jitter object pointer

33.77.1.26 void * max_jit_obex_new (void * *mc*, t_symbol * *classname*)

Allocates and initializes a new Max wrapper object instance.

This is used in place of the newobject function.

Parameters

mc Max class pointer

classname Jitter class name to wrap

Returns

pointer to new Max wrapper object instance

33.77.1.27 t_jit_err max_jit_obex_proxy_new (void * *x*, long *c*)

Creates a new proxy inlet.

Parameters

x Max wrapper object pointer

c inlet index

Returns

t_jit_err error code

33.78 Memory Module

Collaboration diagram for Memory Module:



Functions

- `void * jit_getbytes (long size)`
Allocates a pointer to memory.
- `void jit_freebytes (void *ptr, long size)`
Frees a pointer to memory.
- `void ** jit_handle_new (long size)`
Allocates a memory handle.
- `void jit_handle_free (void **handle)`
Frees a memory handle.
- `long jit_handle_size_get (void **handle)`
Retrieves a memory handle's size in bytes.
- `t_jit_err jit_handle_size_set (void **handle, long size)`
Sets a memory handle's size in bytes.
- `long jit_handle_lock (void **handle, long lock)`
Sets a memory handle's lock state.
- `void jit_copy_bytes (void *dest, const void *src, long bytes)`
Copy bytes from source to destination pointer.
- `long jit_freemem (void)`
Reports free memory.
- `char * jit_newptr (long size)`
Allocates a pointer to memory.
- `void jit_disposeptr (char *ptr)`
Frees a pointer to memory.

33.78.1 Function Documentation

33.78.1.1 `void jit_copy_bytes (void * dest, const void * src, long bytes)`

Copy bytes from source to destination pointer.

Parameters

dest destination pointer
src source pointer
bytes byte count to copy

33.78.1.2 void jit_disposeptr (char *ptr)

Frees a pointer to memory.

Warning

It is important to avoid mixing memory pools, and therefore to match calls to jit_newptr and jit_-disposeptr.

Parameters

ptr pointer to memory

33.78.1.3 void jit_freebytes (void *ptr, long size)

Frees a pointer to memory.

Depending on the size of the pointer, jit_freebytes will free from either the faster memory pool or the system memory pool.

Warning

It is important to avoid mixing memory pools, and therefore to match calls to jit_getbytes and jit_-freebytes.

Parameters

ptr pointer to memory
size size in bytes allocated

33.78.1.4 long jit_freemem (void)

Reports free memory.

Warning

Obsolete. OS 9 only.

Returns

free bytes

33.78.1.5 void* jit_getbytes (long size)

Allocates a pointer to memory.

Depending on the size requested, jit_getbytes will allocate from either the faster memory pool or the system memory pool.

Warning

It is important to avoid mixing memory pools, and therefore to match calls to jit_getbytes and jit_freebytes.

Parameters

size size in bytes to allocate

Returns

pointer to memory

33.78.1.6 void jit_handle_free (void ** handle)

Frees a memory handle.

Warning

It is important to avoid mixing memory pools, and therefore to match calls to jit_handle_new and jit_handle_free.

Parameters

handle memory handle

33.78.1.7 long jit_handle_lock (void ** handle, long lock)

Sets a memory handle's lock state.

Parameters

handle memory handle

lock state (1=locked, 0=unlocked)

Returns

lock state.

33.78.1.8 void jit_handle_new (long size)**

Allocates a memory handle.

Handles are relocatable sections of memory which should be locked before dereferencing, and unlocked when not in use so that they may be relocated as necessary.

Warning

It is important to avoid mixing memory pools, and therefore to match calls to `jit_handle_new` and `jit_handle_free`.

Parameters

`size` size in bytes to allocate

Returns

memory handle

33.78.1.9 long jit_handle_size_get (void ** *handle*)

Retrieves a memory handle's size in bytes.

Parameters

`handle` memory handle

Returns

size in bytes

33.78.1.10 t_jit_err jit_handle_size_set (void ** *handle*, long *size*)

Sets a memory handle's size in bytes.

Parameters

`handle` memory handle

`size` new size in bytes

Returns

`t_jit_err` error code.

33.78.1.11 char* jit_newptr (long *size*)

Allocates a pointer to memory.

Always allocates from the system memory pool.

Warning

It is important to avoid mixing memory pools, and therefore to match calls to `jit_newptr` and `jit_disposeptr`.

Parameters

`size` size in bytes to allocate

Returns

pointer to memory

33.79 MOP Module

Collaboration diagram for MOP Module:



Data Structures

- struct `t_jit_mop_io`
t_jit_mop_io object struct.
- struct `t_jit_mop`
t_jit_mop object struct.

Functions

- `t_jit_object * jit_mop_io_new (void)`
Constructs instance of t_jit_mop_io.
- `t_jit_object * jit_mop_io_newcopy (t_jit_mop_io *x)`
Constructs instance of t_jit_mop_io, copying settings of input.
- `t_jit_err jit_mop_io_free (t_jit_mop *x)`
Frees instance of t_jit_mop_io.
- `t_jit_err jit_mop_io_restrict_type (t_jit_mop_io *x, t_jit_matrix_info *info)`
Restricts the type specified in t_jit_matrix_info struct to those permitted by t_jit_mop_io instance, overwriting value in t_jit_matrix_info struct.
- `t_jit_err jit_mop_io_restrict_planecount (t_jit_mop_io *x, t_jit_matrix_info *info)`
Restricts the planecount specified in t_jit_matrix_info struct to those permitted by t_jit_mop_io instance, overwriting value in t_jit_matrix_info struct.
- `t_jit_err jit_mop_io_restrict_dim (t_jit_mop_io *x, t_jit_matrix_info *info)`
Restricts the dimension sizes specified in t_jit_matrix_info struct to those permitted by t_jit_mop_io instance, overwriting value in t_jit_matrix_info struct.
- `t_jit_err jit_mop_io_matrix (t_jit_mop_io *x, void *m)`
Sets the internal matrix reference.
- `void * jit_mop_io_getmatrix (t_jit_mop_io *x)`
Retrieves the internal matrix reference.
- `t_jit_err jit_mop_io_ioproc (t_jit_mop_io *x, method ioproc)`
Sets the I/O procedure used when handling incoming matrices.
- `method jit_mop_io_getioproc (t_jit_mop_io *x)`

Retrieves the I/O procedure used when handling incoming matrices.

- **t_jit_object * jit_mop_new** (long inputcount, long outputcount)
Constructs instance of t_jit_mop.
- **t_jit_object * jit_mop_newcopy (t_jit_mop *x)**
Constructs instance of t_jit_mop, copying settings of input.
- **void * jit_mop_getinput (t_jit_mop *x, long i)**
Retrieves input at input list index specified.
- **void * jit_mop_getoutput (t_jit_mop *x, long i)**
Retrieves output at output list index specified.
- **void * jit_mop_getinputlist (t_jit_mop *x)**
Retrieves input list.
- **void * jit_mop_getoutputlist (t_jit_mop *x)**
Retrieves output list.
- **t_jit_err jit_mop_free (t_jit_mop *x)**
Frees instance of t_jit_mop.
- **t_jit_err jit_mop_single_type (void *mop, t_symbol *s)**
Utility function to set the type attribute for all MOP inputs and outputs.
- **t_jit_err jit_mop_single_planecount (void *mop, long c)**
Utility function to set the planecount attribute for all MOP inputs and outputs.
- **t_jit_err jit_mop_methodall (void *mop, t_symbol *s,...)**
Utility function to send the same method to all MOP inputs and outputs.
- **t_jit_err jit_mop_input_nolink (void *mop, long c)**
Utility function to disable all linking attributes for a MOP input.
- **t_jit_err jit_mop_output_nolink (void *mop, long c)**
Utility function to disable all linking attributes for a MOP output.
- **t_jit_err jit_mop_ioproc_copy_adapt (void *mop, void *mop_io, void *matrix)**
MOP I/O procedure to copy and adapt to input.
- **t_jit_err jit_mop_ioproc_copy_trunc (void *mop, void *mop_io, void *matrix)**
MOP I/O procedure to copy, but truncate input.
- **t_jit_err jit_mop_ioproc_copy_trunc_zero (void *mop, void *mop_io, void *matrix)**
MOP I/O procedure to copy, but truncate input.
- **t_symbol * jit_mop_ioproc_tosym (void *ioproc)**
Utility to convert MOP I/O procedure function to a human-readable type name.

33.79.1 Function Documentation

33.79.1.1 `t_jit_err jit_mop_free (t_jit_mop *x)`

Frees instance of `t_jit_mop`.

Parameters

x `t_jit_mop` object pointer

Returns

`t_jit_err` error code

Warning

Use `jit_object_free` instead.

33.79.1.2 `void * jit_mop_getinput (t_jit_mop *x, long i)`

Retrieves input at input list index specified.

Parameters

x `t_jit_mop` object pointer

i index

Returns

`t_jit_mop_io` object pointer

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_mop`.

33.79.1.3 `void * jit_mop_getinputlist (t_jit_mop *x)`

Retrieves input list.

Parameters

x `t_jit_mop` object pointer

Returns

`t_jit_linklist` object pointer

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_mop`.

33.79.1.4 void * jit_mop_getoutput (t_jit_mop * x, long i)

Retrieves output at output list index specified.

Parameters

x [t_jit_mop](#) object pointer
i index

Returns

[t_jit_mop_io](#) object pointer

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of [t_jit_mop](#).

33.79.1.5 void * jit_mop_getoutputlist (t_jit_mop * x)

Retrieves output list.

Parameters

x [t_jit_mop](#) object pointer

Returns

[t_jit_linklist](#) object pointer

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of [t_jit_mop](#).

33.79.1.6 t_jit_err jit_mop_input_nolink (void * mop, long c)

Utility function to disable all linking attributes for a MOP input.

Parameters

mop [t_jit_mop](#) object pointer
c input index

Returns

[t_jit_err](#) error code

33.79.1.7 t_jit_err jit_mop_io_free (t_jit_mop *x)

Frees instance of [t_jit_mop_io](#).

Parameters

x [t_jit_mop_io](#) object pointer

Returns

t_jit_err error code

Warning

Use `jit_object_free` instead.

33.79.1.8 method jit_mop_io_getioproc (t_jit_mop_io *x)

Retrieves the I/O procedure used when handling incoming matrices.

Parameters

x [t_jit_mop_io](#) object pointer

Returns

I/O procedure

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of [t_jit_mop_io](#).

33.79.1.9 void * jit_mop_io_getmatrix (t_jit_mop_io *x)

Retrieves the internal matrix reference.

Parameters

x [t_jit_mop_io](#) object pointer

Returns

[t_jit_matrix](#) object pointer

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of [t_jit_mop_io](#).

33.79.1.10 t_jit_err jit_mop_io_ioproc (t_jit_mop_io *x, method ioproc)

Sets the I/O procedure used when handling incoming matrices.

Parameters

x [t_jit_mop_io](#) object pointer

ioproc I/O procedure

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of [t_jit_mop_io](#).

33.79.1.11 t_jit_err jit_mop_io_matrix (t_jit_mop_io *x, void *m)

Sets the internal matrix reference.

Parameters

x [t_jit_mop_io](#) object pointer

m [t_jit_matrix](#) object pointer

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of [t_jit_mop_io](#).

33.79.1.12 t_jit_object * jit_mop_io_new (void)

Constructs instance of [t_jit_mop_io](#).

Returns

[t_jit_mop_io](#) object pointer

Warning

This function is not exported, but is provided for reference when calling via jit_object_new.

33.79.1.13 `t_jit_object * jit_mop_io_newcopy (t_jit_mop_io * x)`

Constructs instance of `t_jit_mop_io`, copying settings of input.

Parameters

x `t_jit_mop_io` object pointer

Returns

`t_jit_mop_io` object pointer

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_mop_io`.

33.79.1.14 `t_jit_err jit_mop_io_restrict_dim (t_jit_mop_io * x, t_jit_matrix_info * info)`

Restricts the dimension sizes specified in `t_jit_matrix_info` struct to those permitted by `t_jit_mop_io` instance, overwriting value in `t_jit_matrix_info` struct.

Parameters

x `t_jit_mop_io` object pointer

info `t_jit_matrix_info` pointer

Returns

`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_mop_io`.

33.79.1.15 `t_jit_err jit_mop_io_restrict_planecount (t_jit_mop_io * x, t_jit_matrix_info * info)`

Restricts the planecount specified in `t_jit_matrix_info` struct to those permitted by `t_jit_mop_io` instance, overwriting value in `t_jit_matrix_info` struct.

Parameters

x `t_jit_mop_io` object pointer

info `t_jit_matrix_info` pointer

Returns

`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_mop_io`.

33.79.1.16 `t_jit_err jit_mop_io_restrict_type (t_jit_mop_io *x, t_jit_matrix_info *info)`

Restricts the type specified in `t_jit_matrix_info` struct to those permitted by `t_jit_mop_io` instance, over-writing value in `t_jit_matrix_info` struct.

Parameters

x `t_jit_mop_io` object pointer
info `t_jit_matrix_info` pointer

Returns

`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_mop_io`.

33.79.1.17 `t_jit_err jit_mop_ioproc_copy_adapt (void *mop, void *mop_io, void *matrix)`

MOP I/O procedure to copy and adapt to input.

Parameters

mop `t_jit_mop` object pointer
mop_io `t_jit_mop_io` object pointer
matrix `t_jit_matrix` object pointer

Returns

`t_jit_err` error code

```
void *m;
t_jit_matrix_info info;

if (matrix && (m= jit_object_method(mop_io, _jit_sym_getmatrix))) {
    jit_object_method(matrix, _jit_sym_getinfo, &info);
    jit_object_method(mop_io, _jit_sym_restrict_type, &info);
    jit_object_method(mop_io, _jit_sym_restrict_dim, &info);
    jit_object_method(mop_io, _jit_sym_restrict_planecount, &info);
    jit_object_method(m, _jit_sym_setinfo, &info);
    jit_object_method(m, _jit_sym_frommatrix, matrix, NULL);
}

return JIT_ERR_NONE;
```

33.79.1.18 `t_jit_err jit_mop_ioproc_copy_trunc (void *mop, void *mop_io, void *matrix)`

MOP I/O procedure to copy, but truncate input.

Parameters

mop `t_jit_mop` object pointer

mop *t_jit_mop_io* object pointer
matrix *t_jit_matrix* object pointer

Returns

t_jit_err error code

```
void *m;
t_jit_matrix_info info;

if (matrix&&(m= jit_object_method(mop_io,_jit_sym_getmatrix))) {
    jit_object_method(m,_jit_sym_frommatrix_trunc,matrix);
}

return JIT_ERR_NONE;
```

33.79.1.19 *t_jit_err jit_mop_ioproc_copy_trunc_zero (void * *mop*, void * *mop_io*, void * *matrix*)*

MOP I/O procedure to copy, but truncate input.

Zero elsewhere.

Parameters

mop *t_jit_mop* object pointer
mop *t_jit_mop_io* object pointer
matrix *t_jit_matrix* object pointer

Returns

t_jit_err error code

```
void *m;
t_jit_matrix_info info;

if (matrix&&(m= jit_object_method(mop_io,_jit_sym_getmatrix))) {
    jit_object_method(m,_jit_sym_clear);
    jit_object_method(m,_jit_sym_frommatrix_trunc,matrix);
}

return JIT_ERR_NONE;
```

33.79.1.20 *t_symbol* jit_mop_ioproc_tosym (void * *ioproc*)*

Utility to convert MOP I/O procedure function to a human-readable type name.

Parameters

ioproc *t_jit_mop_io* procedure pointer

Returns

t_symbol pointer

```

if (ioproc==NULL) {
    return ps_resamp;
} else if (ioproc==jit_mop_ioproc_copy_adapt) {
    return ps_adapt;
} else if (ioproc==jit_mop_ioproc_copy_trunc) {
    return ps_trunc;
} else if (ioproc==jit_mop_ioproc_copy_trunc_zero) {
    return ps_trunc_zero;
} else {
    return ps_custom;
}
return ps_resamp;

```

33.79.1.21 **t_jit_err jit_mop_methodall (void * *mop*, t_symbol * *s*, ...)**

Utility function to send the same method to all MOP inputs and outputs.

Parameters

mop [t_jit_mop](#) object pointer
s method symbol
... untyped arguments

Returns

[t_jit_err](#) error code

33.79.1.22 **t_jit_object * jit_mop_new (long *inputcount*, long *outputcount*)**

Constructs instance of [t_jit_mop](#).

Returns

[t_jit_mop](#) object pointer

Warning

This function is not exported, but is provided for reference when calling via `jit_object_new`.

33.79.1.23 **t_jit_object * jit_mop_newcopy (t_jit_mop * *x*)**

Constructs instance of [t_jit_mop](#), copying settings of input.

Parameters

x [t_jit_mop](#) object pointer

Returns

[t_jit_mop](#) object pointer

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of [t_jit_mop](#).

33.79.1.24 t_jit_err jit_mop_output_nolink (void * *mop*, long *c*)

Utility function to disable all linking attributes for a MOP output.

Parameters

mop t_jit_mop object pointer
c output index

Returns

t_jit_err error code

33.79.1.25 t_jit_err jit_mop_single_planeCount (void * *mop*, long *c*)

Utility function to set the planeCount attribute for all MOP inputs and outputs.

Parameters

mop t_jit_mop object pointer
c planeCount

Returns

t_jit_err error code

33.79.1.26 t_jit_err jit_mop_single_type (void * *mop*, t_symbol * *s*)

Utility function to set the type attribute for all MOP inputs and outputs.

Parameters

mop t_jit_mop object pointer
s type symbol

Returns

t_jit_err error code

33.80 Parallel Utility Module

Collaboration diagram for Parallel Utility Module:



Functions

- void `jit_parallel_ndim_calc` (`t_jit_parallel_ndim *p`)
Tasks N-dimensional matrix calcuations to multiple threads if appropriate.
- void `jit_parallel_ndim_simplecalc1` (`method fn, void *data, long dimcount, long *dim, long planeCount, t_jit_matrix_info *mInfo1, char *bp1, long flags1`)
Tasks one input/output N-dimensional matrix calcuations to multiple threads if appropriate.
- void `jit_parallel_ndim_simplecalc2` (`method fn, void *data, long dimcount, long *dim, long planeCount, t_jit_matrix_info *mInfo1, char *bp1, t_jit_matrix_info *mInfo2, char *bp2, long flags1, long flags2`)
Tasks two input/output N-dimensional matrix calcuations to multiple threads if appropriate.
- void `jit_parallel_ndim_simplecalc3` (`method fn, void *data, long dimcount, long *dim, long planeCount, t_jit_matrix_info *mInfo1, char *bp1, t_jit_matrix_info *mInfo2, char *bp2, t_jit_matrix_info *mInfo3, char *bp3, long flags1, long flags2, long flags3`)
Tasks three input/output N-dimensional matrix calcuations to multiple threads if appropriate.
- void `jit_parallel_ndim_simplecalc4` (`method fn, void *data, long dimcount, long *dim, long planeCount, t_jit_matrix_info *mInfo1, char *bp1, t_jit_matrix_info *mInfo2, char *bp2, t_jit_matrix_info *mInfo3, char *bp3, t_jit_matrix_info *mInfo4, char *bp4, long flags1, long flags2, long flags3, long flags4`)
Tasks four input/output N-dimensional matrix calcuations to multiple threads if appropriate.

33.80.1 Function Documentation

33.80.1.1 void `jit_parallel_ndim_calc` (`t_jit_parallel_ndim *p`)

Tasks N-dimensional matrix calcuations to multiple threads if appropriate.

This function is ultimately what the other parallel utility functions call after having set up the `t_jit_parallel_ndim` struct. The operation is tasked to multiple threads if all of the following conditions are met:

- multiple processors or cores are present
- parallel processing is enabled
- the size of the matrix data is larger then the parallel threshold

Parameters

`p` parallel ndim calc data

33.80.1.2 void jit_parallel_ndim_simplecalc1 (method *fn*, void * *data*, long *dimcount*, long * *dim*, long *planeCount*, t_jit_matrix_info * *mInfo1*, char * *bp1*, long *flags1*)

Tasks one input/output N-dimensional matrix calculations to multiple threads if appropriate.

This function fills out the t_jit_parallel_ndim struct for a one input/output N-dimensional matrix calc method, and calls jit_parallel_ndim_calc. This function does not distinguish between what is an input or output.

Parameters

fn N-dimensional matrix calc method
data user defined pointer (typically object)
dimcount master number of dimensions to iterate
dim master pointer to dimension sizes
planeCount master number of planes
mInfo1 matrix info for first input/output
bp1 matrix data pointer for first input/output
flags1 parallel flags for first input/output

33.80.1.3 void jit_parallel_ndim_simplecalc2 (method *fn*, void * *data*, long *dimcount*, long * *dim*, long *planeCount*, t_jit_matrix_info * *mInfo1*, char * *bp1*, t_jit_matrix_info * *mInfo2*, char * *bp2*, long *flags1*, long *flags2*)

Tasks two input/output N-dimensional matrix calculations to multiple threads if appropriate.

This function fills out the t_jit_parallel_ndim struct for a two input/output N-dimensional matrix calc method, and calls jit_parallel_ndim_calc. This function does not distinguish between what is an input or output.

Parameters

fn N-dimensional matrix calc method
data user defined pointer (typically object)
dimcount master number of dimensions to iterate
dim master pointer to dimension sizes
planeCount master number of planes
mInfo1 matrix info for first input/output
bp1 matrix data pointer for first input/output
flags1 parallel flags for first input/output
mInfo2 matrix info for second input/output
bp2 matrix data pointer for second input/output
flags2 parallel flags for second input/output

33.80.1.4 void jit_parallel_ndim_simplecalc3 (method *fn*, void * *data*, long *dimcount*, long * *dim*, long *planeCount*, t_jit_matrix_info * *mInfo1*, char * *bp1*, t_jit_matrix_info * *mInfo2*, char * *bp2*, t_jit_matrix_info * *mInfo3*, char * *bp3*, long *flags1*, long *flags2*, long *flags3*)

Tasks three input/output N-dimensional matrix calcuations to multiple threads if appropriate.

This function fills out the t_jit_parallel_ndim struct for a three input/output N-dimensional matrix calc method, and calls jit_parallel_ndim_calc. This function does not distinguish between what is an input or output.

Parameters

fn N-dimensional matrix calc method
data user defined pointer (typically object)
dimcount master number of dimensions to iterate
dim master pointer to dimension sizes
planeCount master number of planes
mInfo1 matrix info for first input/output
bp1 matrix data pointer for first input/output
flags1 parallel flags for first input/output
mInfo2 matrix info for second input/output
bp2 matrix data pointer for second input/output
flags2 parallel flags for second input/output
mInfo3 matrix info for third input/output
bp3 matrix data pointer for third input/output
flags3 parallel flags for third input/output

33.80.1.5 void jit_parallel_ndim_simplecalc4 (method *fn*, void * *data*, long *dimcount*, long * *dim*, long *planeCount*, t_jit_matrix_info * *mInfo1*, char * *bp1*, t_jit_matrix_info * *mInfo2*, char * *bp2*, t_jit_matrix_info * *mInfo3*, char * *bp3*, t_jit_matrix_info * *mInfo4*, char * *bp4*, long *flags1*, long *flags2*, long *flags3*, long *flags4*)

Tasks four input/output N-dimensional matrix calcuations to multiple threads if appropriate.

This function fills out the t_jit_parallel_ndim struct for a three input/output N-dimensional matrix calc method, and calls jit_parallel_ndim_calc. This function does not distinguish between what is an input or output.

Parameters

fn N-dimensional matrix calc method
data user defined pointer (typically object)
dimcount master number of dimensions to iterate
dim master pointer to dimension sizes
planeCount master number of planes
mInfo1 matrix info for first input/output
bp1 matrix data pointer for first input/output
flags1 parallel flags for first input/output

minfo2 matrix info for second input/output
bp2 matrix data pointer for second input/output
flags2 parallel flags for second input/output
minfo3 matrix info for third input/output
bp3 matrix data pointer for third input/output
flags3 parallel flags for third input/output
minfo4 matrix info for fourth input/output
bp4 matrix data pointer for fourth input/output
flags4 parallel flags for fourth input/output

33.81 MOP Max Wrapper Module

Collaboration diagram for MOP Max Wrapper Module:



Functions

- t_jit_err [max_jit_classex_mop_wrap](#) (void *mclass, void *jclass, long flags)

Adds default methods and attributes to the MOP Max wrapper class.
- t_jit_err [max_jit_classex_mop_mproc](#) (void *mclass, void *jclass, void *mproc)

Sets a custom matrix procedure for the MOP Max wrapper class.
- t_jit_err [max_jit_mop_setup](#) (void *x)

Sets up necessary resources for MOP Max wrapper object.
- t_jit_err [max_jit_mop_variable_addinputs](#) (void *x, long c)

Sets the number of inputs for a variable input MOP Max wrapper object.
- t_jit_err [max_jit_mop_variable_addoutputs](#) (void *x, long c)

Sets the number of outputs for a variable input MOP Max wrapper object.
- t_jit_err [max_jit_mop_inputs](#) (void *x)

Creates input resources for a MOP Max wrapper object.
- t_jit_err [max_jit_mop_outputs](#) (void *x)

Creates output resources for a MOP Max wrapper object.
- t_jit_err [max_jit_mop_matrixout_new](#) (void *x, long c)

Creates matrix outlet for a MOP Max wrapper object.
- t_jit_err [max_jit_mop_matrix_args](#) (void *x, long argc, [t_atom](#) *argv)

Process matrix arguments for a MOP Max wrapper object.
- t_jit_err [max_jit_mop_jit_matrix](#) (void *x, [t_symbol](#) *s, long argc, [t_atom](#) *argv)

Default jit_matrix method for a MOP Max wrapper object.
- t_jit_err [max_jit_mop_assist](#) (void *x, void *b, long m, long a, char *s)

Default assist method for a MOP Max wrapper object.
- t_jit_err [max_jit_mop_bang](#) (void *x)

Default bang method for a MOP Max wrapper object.
- t_jit_err [max_jit_mop_outputmatrix](#) (void *x)

Default outputmatrix method for a MOP Max wrapper object.
- void [max_jit_mop_clear](#) (void *x)

Default clear method for a MOP Max wrapper object.

- `t_jit_err max_jit_mop_notify (void *x, t_symbol *s, t_symbol *msg)`

Default notify method for a MOP Max wrapper object.

- `void max_jit_mop_free (void *x)`

Frees additional resources used by a MOP Max wrapper object.

- `t_jit_err max_jit_mop_adapt_matrix_all (void *x, void *y)`

Adapts all input and output matrices to matrix specified.

- `void * max_jit_mop_get_io_by_name (void *x, t_symbol *s)`

Retrieves `t_jit_mop_io` object pointer by name.

- `void * max_jit_mop_getinput (void *x, long c)`

Retrieves input `t_jit_mop_io` object pointer index.

- `void * max_jit_mop_getoutput (void *x, long c)`

Retrieves output `t_jit_mop_io` object pointer index.

- `long max_jit_mop_getoutputmode (void *x)`

Retrieves current MOP Max wrapper class output mode.

- `t_jit_err max_jit_mop_setup_simple (void *x, void *o, long argc, t_atom *argv)`

Initializes default state and resources for MOP Max wrapper class.

33.81.1 Function Documentation

33.81.1.1 `t_jit_err max_jit_classex_mop_mproc (void * mclass, void * jclass, void * mproc)`

Sets a custom matrix procedure for the MOP Max wrapper class.

Parameters

`mclass` max jit classex pointer returned from `max_jit_classex_setup`

`jclass` `t_jit_class` pointer, typically returned from `jit_class_findbyname`

`mproc` matrix procedure

Returns

`t_jit_err` error code

33.81.1.2 `t_jit_err max_jit_classex_mop_wrap (void * mclass, void * jclass, long flags)`

Adds default methods and attributes to the MOP Max wrapper class.

Parameters

`mclass` max jit classex pointer returned from `max_jit_classex_setup`

jclass t_jit_class pointer, typically returned from jit_class_findbyname
flags flags to override default MOP Max wrapper behavior

Returns

t_jit_err error code

33.81.1.3 t_jit_err max_jit_mop_adapt_matrix_all (void **x*, void **y*)

Adapts all input and output matrices to matrix specified.

Typically used within the MOP Max Wrapper jit_matrix method for left most input.

Parameters

x Max object pointer
y matrix to adapt to

Returns

t_jit_err error code

33.81.1.4 t_jit_err max_jit_mop_assist (void **x*, void **b*, long *m*, long *a*, char **s*)

Default assist method for a MOP Max wrapper object.

Parameters

x Max object pointer
b ignored
m inlet or outlet type
a index
s output string

Returns

t_jit_err error code

33.81.1.5 t_jit_err max_jit_mop_bang (void **x*)

Default bang method for a MOP Max wrapper object.

Simply calls the default outputmatrix method.

Parameters

x Max object pointer

Returns

t_jit_err error code

33.81.1.6 void max_jit_mop_clear (void * *x*)

Default clear method for a MOP Max wrapper object.

Calls the clear method on all input and output matrices.

Parameters

x Max object pointer

Returns

t_jit_err error code

33.81.1.7 void max_jit_mop_free (void * *x*)

Frees additional resources used by a MOP Max wrapper object.

Parameters

x Max object pointer

33.81.1.8 void* max_jit_mop_get_io_by_name (void * *x*, t_symbol * *s*)

Retrieves [t_jit_mop_io](#) object pointer by name.

Parameters

x Max object pointer

s input/output name (e.g. in, in2 , out, out2, etc.)

Returns

t_jit_err error code

33.81.1.9 void* max_jit_mop_getinput (void * *x*, long *c*)

Retrieves input [t_jit_mop_io](#) object pointer index.

Parameters

x Max object pointer

c input index

Returns

t_jit_err error code

33.81.1.10 void* max_jit_mop_getoutput (void **x*, long *c*)

Retrieves output [t_jit_mop_io](#) object pointer index.

Parameters

- x* Max object pointer
- c* output index

Returns

[t_jit_err](#) error code

33.81.1.11 long max_jit_mop_getoutputmode (void **x*)

Retrieves current MOP Max wrapper class output mode.

Parameters

- x* Max object pointer

Returns

[t_jit_err](#) error code

33.81.1.12 t_jit_err max_jit_mop_inputs (void **x*)

Creates input resources for a MOP Max wrapper object.

Parameters

- x* Max object pointer

Returns

[t_jit_err](#) error code

33.81.1.13 t_jit_err max_jit_mop jit_matrix (void **x*, t_symbol **s*, long *argc*, t_atom **argv*)

Default jit_matrix method for a MOP Max wrapper object.

Parameters

- x* Max object pointer
- s* message symbol ("jit_matrix")
- argc* argument count
- argv* argument vector

Returns

[t_jit_err](#) error code

33.81.1.14 t_jit_err max_jit_mop_matrix_args (void *x, long argc, t_atom *argv)

Process matrix arguments for a MOP Max wrapper object.

Parameters

- x* Max object pointer
- argc* argument count
- argv* argument vector

Returns

t_jit_err error code

33.81.1.15 t_jit_err max_jit_mop_matrixout_new (void *x, long c)

Creates matrix outlet for a MOP Max wrapper object.

Parameters

- x* Max object pointer
- c* output index (zero based)

Returns

t_jit_err error code

33.81.1.16 t_jit_err max_jit_mop_notify (void *x, t_symbol *s, t_symbol *msg)

Default notify method for a MOP Max wrapper object.

Handles any notification methods from any input and output matrix.

Parameters

- x* Max object pointer
- s* notifier name
- msg* notification message

Returns

t_jit_err error code

33.81.1.17 t_jit_err max_jit_mop_outputmatrix (void *x)

Default outputmatrix method for a MOP Max wrapper object.

Calculates and outputs according to the MOP outputmode attribute.

Parameters

- x* Max object pointer

Returns

t_jit_err error code

33.81.1.18 t_jit_err max_jit_mop_outputs (void *x)

Creates output resources for a MOP Max wrapper object.

Parameters

x Max object pointer

Returns

t_jit_err error code

33.81.1.19 t_jit_err max_jit_mop_setup (void *x)

Sets up necessary resources for MOP Max wrapper object.

Parameters

x Max object pointer

Returns

t_jit_err error code

33.81.1.20 t_jit_err max_jit_mop_setup_simple (void *x, void *o, long argc, t_atom *argv)

Initializes default state and resources for MOP Max wrapper class.

Parameters

x Max object pointer

o Jitter object pointer

argc argument count

argv argument vector

Returns

t_jit_err error code

```
max_jit_obex_jitob_set(x,o);
max_jit_obex_dumpout_set(x,outlet_new(x,NULL));
max_jit_mop_setup(x);
max_jit_mop_inputs(x);
max_jit_mop_outputs(x);
max_jit_mop_matrix_args(x,argc,argv);

return JIT_ERR_NONE;
```

33.81.1.21 t_jit_err max_jit_mop_variable_addinputs (void *x, long c)

Sets the number of inputs for a variable input MOP Max wrapper object.

Parameters

- x* Max object pointer
- c* inlet count

Returns

t_jit_err error code

33.81.1.22 t_jit_err max_jit_mop_variable_addoutputs (void *x, long c)

Sets the number of outputs for a variable input MOP Max wrapper object.

Parameters

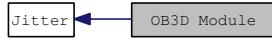
- x* Max object pointer
- c* inlet count

Returns

t_jit_err error code

33.82 OB3D Module

Collaboration diagram for OB3D Module:



Functions

- long [jit_gl_report_error](#) (char *prefix)
Tests for OpenGL error and reports to Max window.
- const char * [jit_gl_get_vendor](#) ()
Retrieves OpenGL vendor string.
- const char * [jit_gl_get_renderer](#) ()
Retrieves OpenGL renderer string.
- const char * [jit_gl_get_version](#) ()
Retrieves OpenGL version string.
- const char * [jit_gl_get_glu_version](#) ()
Retrieves OpenGL GL Utilities version string.
- const char * [jit_gl_get_extensions](#) ()
Retrieves OpenGL extensions string.
- char [jit_gl_is_min_version](#) (int major, int minor, int release)
Tests current OpenGL version to be greater than or equal to the version provided.
- char [jit_gl_is_extension_supported](#) (t_jit_gl_context ctx, const char *ext)
Given a t_jit_gl_context pointer, checks to see if it supports the provided extension.
- t_jit_glchunk * [jit_glchunk_new](#) (t_symbol *prim, int planes, int vertices, int indices)
Allocates and initializes a t_jit_glchunk struct.
- t_jit_glchunk * [jit_glchunk_grid_new](#) (t_symbol *prim, int planes, int width, int height)
Allocates and initializes a t_jit_glchunk struct with 2D grid matrix.
- void [jit_glchunk_delete](#) (t_jit_glchunk *x)
Disposes t_jit_glchunk struct.
- t_jit_err [jit_glchunk_copy](#) (t_jit_glchunk **new, t_jit_glchunk *orig)
Allocates t_jit_glchunk struct, and copies from t_jit_gl_struct provided.
- t_jit_err [jit_gl_drawinfo_setup](#) (void *x, t_jit_gl_drawinfo *drawinfo)
Initializes t_jit_gl_drawinfo struct with the current context and ob3d.
- long [jit_gl_drawinfo_active_textures](#) (t_jit_gl_drawinfo *drawinfo)

Determine the number of active texture units to use.

- void **jit_gl_texcoord1f** (**t_jit_gl_drawinfo** *drawinfo, float s)
Set texture coordinate for all active texture units.
- void **jit_gl_texcoord2f** (**t_jit_gl_drawinfo** *drawinfo, float s, float t)
Set texture coordinate for all active texture units.
- void **jit_gl_texcoord3f** (**t_jit_gl_drawinfo** *drawinfo, float s, float t, float r)
Set texture coordinate for all active texture units.
- void **jit_gl_texcoord1fv** (**t_jit_gl_drawinfo** *drawinfo, float *v)
Set texture coordinate for all active texture units.
- void **jit_gl_texcoord2fv** (**t_jit_gl_drawinfo** *drawinfo, float *v)
Set texture coordinate for all active texture units.
- void **jit_gl_texcoord3fv** (**t_jit_gl_drawinfo** *drawinfo, float *v)
Set texture coordinate for all active texture units.
- void **jit_gl_bindtexture** (**t_jit_gl_drawinfo** *drawinfo, **t_symbol** *s, long i)
Bind texture for specified texture unit.
- void **jit_gl_unbindtexture** (**t_jit_gl_drawinfo** *drawinfo, **t_symbol** *s, long i)
Unbind texture for specified texture unit.
- void **jit_gl_begincapture** (**t_jit_gl_drawinfo** *drawinfo, **t_symbol** *s, long i)
Begin texture capture.
- void **jit_gl_endcapture** (**t_jit_gl_drawinfo** *drawinfo, **t_symbol** *s, long i)
End texture capture.
- void * **jit_ob3d_setup** (void *jit_class, long oboffset, long flags)
Adds default methods and attributes to the OB3D class.
- void * **jit_ob3d_new** (void *x, **t_symbol** *dest_name)
Allocates and initializes OB3D resources.
- void **jit_ob3d_free** (void *jit_ob)
Disposes OB3D resources.
- **t_jit_err jit_ob3d_set_context** (void *jit_ob)
Sets the current Open GL context to the context referenced by the OB3D drawto attribute.
- void * **ob3d_jitob_get** (void *v)
Retrieves parent Jitter object from opaque t_jit_ob3d struct.
- long **ob3d_auto_get** (void *v)
Retrieves automatic flag from opaque t_jit_ob3d struct.

- `long ob3d_enable_get (void *v)`
Retrieves enable flag from opaque t_jit_ob3d struct.
- `long ob3d_ui_get (void *v)`
Retrieves UI flag from opaque t_jit_ob3d struct.
- `void * ob3d_outlet_get (void *v)`
Retrieves matrix outlet from opaque t_jit_ob3d struct.
- `long ob3d_dirty_get (void *v)`
Retrieves dirty flag from opaque t_jit_ob3d struct.
- `void ob3d_dirty_set (void *v, long c)`
Sets dirty flag from opaque t_jit_ob3d struct.
- `void ob3d_dest_dim_set (void *v, long width, long height)`
Sets destination dimensions in opaque t_jit_ob3d struct.
- `void ob3d_dest_dim_get (void *v, long *width, long *height)`
Gets destination dimensions from opaque t_jit_ob3d struct.
- `void ob3d_render_ptr_set (void *v, void *render_ptr)`
Sets renderer pointer in opaque t_jit_ob3d struct.
- `void * ob3d_render_ptr_get (void *v)`
Gets renderer pointer from opaque t_jit_ob3d struct.
- `void max_ob3d_setup (void)`
Adds default methods and OB3D Max wrapper class.
- `void max_jit_ob3d_attach (void *x, t_jit_object *jit_ob, void *outlet)`
Allocates and initializes OB3D Max wrapper related resources.
- `void max_jit_ob3d_detach (void *x)`
Disposes OB3D Max wrapper related resources.
- `t_jit_err max_jit_ob3d_assist (void *x, void *b, long m, long a, char *s)`
Default OB3D Max wrapper assistance method.
- `void max_ob3d_bang (t_max_object *x)`
Default OB3D Max wrapper bang method.
- `void max_ob3d_notify (t_max_object *x, t_symbol *sender_name, t_symbol *msg, void *p_- sender)`
Default OB3D Max wrapper notification method.
- `t_jit_err jit_ob3d_draw_chunk (void *v, t_jit_glchunk *chunk)`
Draws one t_jit_glchunk If the OB3D is not in matrixoutput mode, the drawing call is made directly to the renderer.

33.82.1 Function Documentation

33.82.1.1 void jit_gl_begincapture (*t_jit_gl_drawinfo * drawinfo, t_symbol * s, long i*)

Begin texture capture.

Parameters

- drawinfo* [t_jit_gl_drawinfo](#) pointer
- s* texture name
- i* ignored

33.82.1.2 void jit_gl_bindtexture (*t_jit_gl_drawinfo * drawinfo, t_symbol * s, long i*)

Bind texture for specified texture unit.

Parameters

- drawinfo* [t_jit_gl_drawinfo](#) pointer
- s* texture name
- i* texture unit

33.82.1.3 long jit_gl_drawinfo_active_textures (*t_jit_gl_drawinfo * drawinfo*)

Determine the number of active texture units to use.

Parameters

- drawinfo* [t_jit_gl_drawinfo](#) pointer

Returns

number of active texture units

33.82.1.4 t_jit_err jit_gl_drawinfo_setup (*void * x, t_jit_gl_drawinfo * drawinfo*)

Initializes [t_jit_gl_drawinfo](#) struct with the current context and ob3d.

Parameters

- x* Jitter object pointer
- drawinfo* [t_jit_gl_drawinfo](#) pointer

Returns

[t_jit_err](#) error code

33.82.1.5 void jit_gl_endcapture (*t_jit_gl_drawinfo* * *drawinfo*, *t_symbol* * *s*, long *i*)

End texture capture.

Returns

drawinfo *t_jit_gl_drawinfo* pointer

s texture name

i ignored

33.82.1.6 const char* jit_gl_get_extensions ()

Retrieves OpenGL extensions string.

Equivalent to glGetString(GL_EXTENSIONS). Assumes a valid context has been set.

Returns

OpenGL GL extensions string

33.82.1.7 const char* jit_gl_get_glu_version ()

Retrieves OpenGL GL Utilities version string.

Equivalent to glGetString(GL_GLU_VERSION). Assumes a valid context has been set.

Returns

OpenGL GL Utilities version string

33.82.1.8 const char* jit_gl_get_renderer ()

Retrieves OpenGL renderer string.

Equivalent to glGetString(GL_RENDERER). Assumes a valid context has been set.

Returns

OpenGL renderer string

33.82.1.9 const char* jit_gl_get_vendor ()

Retrieves OpenGL vendor string.

Equivalent to glGetString(GL_VENDOR). Assumes a valid context has been set.

Returns

OpenGL vendor string

33.82.1.10 const char* jit_gl_get_version ()

Retrieves OpenGL version string.

Equivalent to glGetString(GL_VERSION). Assumes a valid context has been set.

Returns

OpenGL version string

33.82.1.11 char jit_gl_is_extension_supported (t_jit_gl_context *ctx*, const char * *ext*)

Given a t_jit_gl_context pointer, checks to see if it supports the provided extension.

Equivalent to testing for the substring within the string returned by glGetString(GL_EXTENSIONS).

Parameters

ctx t_jit_gl_context pointer

ext extension string

Returns

1 if true, 0 if false.

33.82.1.12 char jit_gl_is_min_version (int *major*, int *minor*, int *release*)

Tests current OpenGL version to be greater than or equal to the version provided.

Assumes a valid context has been set.

Parameters

major major version number

minor minor version number

release release version number

Returns

1 if true, 0 if false.

33.82.1.13 long jit_gl_report_error (char * *prefix*)

Tests for OpenGL error and reports to Max window.

Parameters

prefix prefix string

Returns

OpenGL error code

33.82.1.14 void jit_gl_texcoord1f (t_jit_gl_drawinfo * drawinfo, float s)

Set texture coordinate for all active texture units.

Equivalent to glMultiTexCoord1fARB for each active texture unit.

Parameters

drawinfo t_jit_gl_drawinfo pointer

s s texture coordinate

33.82.1.15 void jit_gl_texcoord1fv (t_jit_gl_drawinfo * drawinfo, float * v)

Set texture coordinate for all active texture units.

Equivalent to glMultiTexCoord1fvARB for each active texture unit.

Parameters

drawinfo t_jit_gl_drawinfo pointer

v texture coordinate vector

33.82.1.16 void jit_gl_texcoord2f (t_jit_gl_drawinfo * drawinfo, float s, float t)

Set texture coordinate for all active texture units.

Equivalent to glMultiTexCoord2fARB for each active texture unit.

Parameters

drawinfo t_jit_gl_drawinfo pointer

s s texture coordinate

t t texture coordinate

33.82.1.17 void jit_gl_texcoord2fv (t_jit_gl_drawinfo * drawinfo, float * v)

Set texture coordinate for all active texture units.

Equivalent to glMultiTexCoord2fvARB for each active texture unit.

Parameters

drawinfo t_jit_gl_drawinfo pointer

v texture coordinate vector

33.82.1.18 void jit_gl_texcoord3f (t_jit_gl_drawinfo * drawinfo, float s, float t, float r)

Set texture coordinate for all active texture units.

Equivalent to glMultiTexCoord3fARB for each active texture unit.

Parameters

drawinfo `t_jit_gl_drawinfo` pointer

s s texture coordinate

t t texture coordinate

r r texture coordinate

33.82.1.19 void jit_gl_texcoord3fv (t_jit_gl_drawinfo * *drawinfo*, float * *v*)

Set texture coordinate for all active texture units.

Equivalent to glMultiTexCoord3fvARB for each active texture unit.

Parameters

drawinfo `t_jit_gl_drawinfo` pointer

v texture coordinate vector

33.82.1.20 void jit_gl_unbindtexture (t_jit_gl_drawinfo * *drawinfo*, t_symbol * *s*, long *i*)

Unbind texture for specified texture unit.

Parameters

drawinfo `t_jit_gl_drawinfo` pointer

s texture name

i texture unit

33.82.1.21 t_jit_err jit_glchunk_copy (t_jit_glchunk ** *new*, t_jit_glchunk * *orig*)

Allocates `t_jit_glchunk` struct, and copies from `t_jit_gl_struct` provided.

Parameters

new pointer to new `t_jit_glchunk` pointer

orig priginal `t_jit_glchunk` pointer

Returns

`t_jit_err` error code

33.82.1.22 void jit_glchunk_delete (t_jit_glchunk * *x*)

Disposes `t_jit_glchunk` struct.

Parameters

x `t_jit_glchunk` pointer

33.82.1.23 `t_jit_glchunk* jit_glchunk_grid_new (t_symbol * prim, int planes, int width, int height)`

Allocates and initializes a `t_jit_glchunk` struct with 2D grid matrix.

Parameters

- `prim` drawing primitive name
- `planes` number of planes to allocate in vertex matrix
- `width` width of vertex matrix to allocate
- `height` height of vertex matrix to allocate

Returns

`t_jit_glchunk` pointer

33.82.1.24 `t_jit_glchunk* jit_glchunk_new (t_symbol * prim, int planes, int vertices, int indices)`

Allocates and initializes a `t_jit_glchunk` struct.

Parameters

- `prim` drawing primitive name
- `planes` number of planes to allocate in vertex matrix
- `vertices` number of vertices to allocate in vertex matrix
- `indices` number of indices to allocate in index matrix, if used

Returns

`t_jit_glchunk` pointer

33.82.1.25 `t_jit_err jit_ob3d_draw_chunk (void * v, t_jit_glchunk * chunk)`

Draws one `t_jit_glchunk`. If the OB3D is not in matrixoutput mode, the drawing call is made directly to the renderer.

Otherwise, the chunk is sent out the OB3D's outlet as a message compatible with `jit.gl.render`.

Parameters

- `v` `t_jit_ob3d` pointer
- `chunk` `t_jit_glchunk` pointer

Returns

`t_jit_err` error code

33.82.1.26 `void jit_ob3d_free (void * jit_ob)`

Disposes OB3D resources.

Parameters

- `jit_ob` Jitter object pointer

33.82.1.27 void* jit_ob3d_new (void *x, t_symbol *dest_name)

Allocates and initializes OB3D resources.

Parameters

- x* Jitter object pointer
- dest_name* drawing destination name

Returns

t_jit_ob3d pointer (opaque)

33.82.1.28 t_jit_err jit_ob3d_set_context (void *jit_ob)

Sets the current Open GL context to the context referenced by the OB3D drawto attribute.

Warning

Not recommended for use within the draw method, as it can have adverse effects when rendering to alternate contexts as is the case when capturing to a texture.

Parameters

- jit_ob* Jitter object pointer

Returns

t_jit_err error code

33.82.1.29 void* jit_ob3d_setup (void *jit_class, long oboffset, long flags)

Adds default methods and attributes to the OB3D class.

Parameters

- jit_class* Jitter class pointer
- oboffset* object struct byte offset for t_jit_ob3d pointer
- flags* flags to override default OB3D behavior

Returns

t_jit_class3d pointer (opaque)

33.82.1.30 t_jit_err max_jit_ob3d_assist (void *x, void *b, long m, long a, char *s)

Default OB3D Max wrapper assistance method.

Parameters

- x* Max object pointer

b ignored
m inlet or outlet type
a index
s output string

Returns

`t_jit_err` error code

33.82.1.31 `void max_jit_ob3d_attach (void *x, t_jit_object *jit_ob, void *outlet)`

Allocates and initializes OB3D Max wrapper related resources.

Parameters

x Max wrapper object pointer
jit_ob Jitter object pointer
outlet matrix outlet pointer

33.82.1.32 `void max_jit_ob3d_detach (void *x)`

Disposes OB3D Max wrapper related resources.

Parameters

x Max wrapper object pointer

33.82.1.33 `void max_ob3d_bang (t_max_object *x)`

Default OB3D Max wrapper bang method.

Parameters

x Max object pointer

33.82.1.34 `void max_ob3d_notify (t_max_object *x, t_symbol *sender_name, t_symbol *msg, void *p_sender)`

Default OB3D Max wrapper notification method.

Parameters

x Max object pointer
sender_name sender's object name
msg notification message
p_sender sender's object pointer

33.82.1.35 long ob3d_auto_get (void * *v*)

Retrieves automatic flag from opaque t_jit_ob3d struct.

Parameters

v t_jit_ob3d pointer

Returns

automatic flag

33.82.1.36 void ob3d_dest_dim_get (void * *v*, long * *width*, long * *height*)

Gets destination dimensions from opaque t_jit_ob3d struct.

Parameters

v t_jit_ob3d pointer

width destination dimensions width pointer

height destination dimensions height pointer

33.82.1.37 void ob3d_dest_dim_set (void * *v*, long *width*, long *height*)

Sets destination dimensions in opaque t_jit_ob3d struct.

Parameters

v t_jit_ob3d pointer

width destination dimensions width

height destination dimensions height

33.82.1.38 long ob3d_dirty_get (void * *v*)

Retrieves dirty flag from opaque t_jit_ob3d struct.

Parameters

v t_jit_ob3d pointer

Returns

dirty flag

33.82.1.39 void ob3d_dirty_set (void * *v*, long *c*)

Sets dirty flag from opaque t_jit_ob3d struct.

Parameters

v t_jit_ob3d pointer

c dirty flag state

33.82.1.40 long ob3d_enable_get (void * v)

Retrieves enable flag from opaque t_jit_ob3d struct.

Parameters

v t_jit_ob3d pointer

Returns

enable flag

33.82.1.41 void* ob3d jitob_get (void * v)

Retrieves parent Jitter object from opaque t_jit_ob3d struct.

Parameters

v t_jit_ob3d pointer

Returns

parent Jitter object pointer

33.82.1.42 void* ob3d_outlet_get (void * v)

Retrieves matrix outlet from opaque t_jit_ob3d struct.

Parameters

v t_jit_ob3d pointer

Returns

matrix outlet

33.82.1.43 void* ob3d_render_ptr_get (void * v)

Gets renderer pointer from opaque t_jit_ob3d struct.

Parameters

v t_jit_ob3d pointer

Returns

renderer pointer

33.82.1.44 void ob3d_render_ptr_set (void * *v*, void * *render_ptr*)

Sets renderer pointer in opaque t_jit_ob3d struct.

Parameters

v t_jit_ob3d pointer

render_ptr renderer pointer

33.82.1.45 long ob3d_ui_get (void * *v*)

Retrieves UI flag from opaque t_jit_ob3d struct.

Parameters

v t_jit_ob3d pointer

Returns

UI flag

33.83 Operator Vector Module

Collaboration diagram for Operator Vector Module:



Functions

- void `jit_op_vector_pass_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: pass (char).
- void `jit_op_vector_mult_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: multiplication (char).
- void `jit_op_vector_div_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: division (char).
- void `jit_op_vector_mod_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: modulo (char).
- void `jit_op_vector_add_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: addition (char).
- void `jit_op_vector_adds_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: saturated addition (char).
- void `jit_op_vector_sub_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: subtraction (char).
- void `jit_op_vector_subs_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: saturated subtraction (char).
- void `jit_op_vector_min_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: minimum (char).
- void `jit_op_vector_max_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: maximum (char).

- void `jit_op_vector_avg_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: average (char).
- void `jit_op_vector_absdiff_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: absolute difference (char).
- void `jit_op_vector_pass_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: pass (float32).
- void `jit_op_vector_mult_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: multiplication (float32).
- void `jit_op_vector_div_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: division (float32).
- void `jit_op_vector_add_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: addition (float32).
- void `jit_op_vector_sub_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: subtraction (float32).
- void `jit_op_vector_min_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: minimum (float32).
- void `jit_op_vector_max_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: maximum (float32).
- void `jit_op_vector_abs_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: absolute value (float32).
- void `jit_op_vector_avg_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: average (float32).
- void `jit_op_vector_absdiff_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: absolute difference (float32).
- void `jit_op_vector_mod_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: modulo (float32).

- void `jit_op_vector_fold_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: fold (float32).
- void `jit_op_vector_wrap_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: wrap (float32).
- void `jit_op_vector_pass_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: pass (float64).
- void `jit_op_vector_mult_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: multiplication (float64).
- void `jit_op_vector_div_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: division (float64).
- void `jit_op_vector_add_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: addition (float64).
- void `jit_op_vector_sub_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: subtraction (float64).
- void `jit_op_vector_min_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: minimum (float64).
- void `jit_op_vector_max_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: maximum (float64).
- void `jit_op_vector_abs_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: absolute value (float64).
- void `jit_op_vector_avg_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: average (float64).
- void `jit_op_vector_absdiff_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: absolute difference (float64).
- void `jit_op_vector_mod_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: mod (float64).

Binary operator: modulo (float64).

- void `jit_op_vector_fold_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: fold (float64).

- void `jit_op_vector_wrap_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: wrap (float64).

- void `jit_op_vector_pass_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: pass (long).

- void `jit_op_vector_mult_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: multiplication (long).

- void `jit_op_vector_div_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: division (long).

- void `jit_op_vector_mod_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: modulo (long).

- void `jit_op_vector_add_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: addition (long).

- void `jit_op_vector_sub_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: subtraction (long).

- void `jit_op_vector_min_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: minimum (long).

- void `jit_op_vector_max_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: maximum (long).

- void `jit_op_vector_abs_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: absolute value (long).

- void `jit_op_vector_avg_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: average (long).

- void [jit_op_vector_absdiff_long](#) (long n, void *vecdata, [t_jit_op_info](#) *in0, [t_jit_op_info](#) *in1, [t_jit_op_info](#) *out)
Binary operator: absolute difference (long).
- void [jit_op_vector_bitand_char](#) (long n, void *vecdata, [t_jit_op_info](#) *in0, [t_jit_op_info](#) *in1, [t_jit_op_info](#) *out)
Binary operator: bitwise and (char).
- void [jit_op_vector_bitor_char](#) (long n, void *vecdata, [t_jit_op_info](#) *in0, [t_jit_op_info](#) *in1, [t_jit_op_info](#) *out)
Binary operator: bitwise or (char).
- void [jit_op_vector_bitxor_char](#) (long n, void *vecdata, [t_jit_op_info](#) *in0, [t_jit_op_info](#) *in1, [t_jit_op_info](#) *out)
Binary operator: bitwise exclusive or (char).
- void [jit_op_vector_bitnot_char](#) (long n, void *vecdata, [t_jit_op_info](#) *in0, [t_jit_op_info](#) *in1, [t_jit_op_info](#) *out)
Unary operator: bitwise not (char).
- void [jit_op_vector_rshift_char](#) (long n, void *vecdata, [t_jit_op_info](#) *in0, [t_jit_op_info](#) *in1, [t_jit_op_info](#) *out)
Binary operator: bitwise right shift (char).
- void [jit_op_vector_lshift_char](#) (long n, void *vecdata, [t_jit_op_info](#) *in0, [t_jit_op_info](#) *in1, [t_jit_op_info](#) *out)
Binary operator: bitwise left shift (char).
- void [jit_op_vector_bitand_long](#) (long n, void *vecdata, [t_jit_op_info](#) *in0, [t_jit_op_info](#) *in1, [t_jit_op_info](#) *out)
Binary operator: bitwise and (long).
- void [jit_op_vector_bitor_long](#) (long n, void *vecdata, [t_jit_op_info](#) *in0, [t_jit_op_info](#) *in1, [t_jit_op_info](#) *out)
Binary operator: bitwise or (long).
- void [jit_op_vector_bitxor_long](#) (long n, void *vecdata, [t_jit_op_info](#) *in0, [t_jit_op_info](#) *in1, [t_jit_op_info](#) *out)
Binary operator: bitwise exclusive or (long).
- void [jit_op_vector_bitnot_long](#) (long n, void *vecdata, [t_jit_op_info](#) *in0, [t_jit_op_info](#) *in1, [t_jit_op_info](#) *out)
Unary operator: bitwise not (long).
- void [jit_op_vector_rshift_long](#) (long n, void *vecdata, [t_jit_op_info](#) *in0, [t_jit_op_info](#) *in1, [t_jit_op_info](#) *out)
Binary operator: bitwise right shift (long).
- void [jit_op_vector_lshift_long](#) (long n, void *vecdata, [t_jit_op_info](#) *in0, [t_jit_op_info](#) *in1, [t_jit_op_info](#) *out)
Binary operator: bitwise left shift (long).

- void `jit_op_vector_flippass_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: flipped pass (char).
- void `jit_op_vector_flipdiv_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: flipped division (char).
- void `jit_op_vector_flipmod_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: flipped modulo (char).
- void `jit_op_vector_flipsub_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: flipped subtraction (char).
- void `jit_op_vector_flippass_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: flipped pass (float32).
- void `jit_op_vector_flipdiv_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: flipped division (float32).
- void `jit_op_vector_flipmod_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: flipped modulo (float32).
- void `jit_op_vector_flipsub_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: flipped subtraction (float32).
- void `jit_op_vector_flippass_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: flipped pass (float64).
- void `jit_op_vector_flipdiv_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: flipped division (float64).
- void `jit_op_vector_flipmod_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: flipped modulo (float64).
- void `jit_op_vector_flippass_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: flipped pass (long).
- void `jit_op_vector_flipdiv_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: flipped division (long).

- void `jit_op_vector_flipmod_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: flipped modulo (long).

- void `jit_op_vector_flipsub_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: flipped subtraction (long).

- void `jit_op_vector_and_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: logical and (char).

- void `jit_op_vector_or_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: logical or (char).

- void `jit_op_vector_not_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: logical not (char).

- void `jit_op_vector_gt_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: greater than (char).

- void `jit_op_vector_gte_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: greater than or equals (char).

- void `jit_op_vector_lt_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: less than (char).

- void `jit_op_vector_lte_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: less than or equals (char).

- void `jit_op_vector_eq_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: equals (char).

- void `jit_op_vector_neq_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: not equals (char).

- void `jit_op_vector_and_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: logical and (float32).

- void `jit_op_vector_or_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: logical or (float32).
- void `jit_op_vector_not_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: logical not (float32).
- void `jit_op_vector_gt_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: greater than (float32).
- void `jit_op_vector_gte_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: greater than or equals (float32).
- void `jit_op_vector_lt_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: less than (float32).
- void `jit_op_vector_lte_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: less than or equals (float32).
- void `jit_op_vector_eq_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: equals (float32).
- void `jit_op_vector_neq_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: not equals (float32).
- void `jit_op_vector_and_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: logical and (float64).
- void `jit_op_vector_or_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: logical or (float64).
- void `jit_op_vector_not_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: logical not (float64).
- void `jit_op_vector_gt_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: greater than (float64).
- void `jit_op_vector_gte_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: greater than or equals (float64).

- void `jit_op_vector_lt_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: less than (float64).
- void `jit_op_vector_lte_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: less than or equals (float64).
- void `jit_op_vector_eq_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: equals (float64).
- void `jit_op_vector_neq_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: not equals (float64).
- void `jit_op_vector_and_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: logical and (long).
- void `jit_op_vector_or_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: logical or (long).
- void `jit_op_vector_not_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: logical not (long).
- void `jit_op_vector_gt_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: greater than (long).
- void `jit_op_vector_gte_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: greater than or equals (long).
- void `jit_op_vector_lt_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: less than (long).
- void `jit_op_vector_lte_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: less than or equals (long).
- void `jit_op_vector_eq_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: equals (long).
- void `jit_op_vector_neq_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: not equals (long).

Binary operator: not equals (long).

- void `jit_op_vector_gtp_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: greater than pass (char).

- void `jit_op_vector_gtep_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: greater than or equals pass (char).

- void `jit_op_vector_ltp_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: less than pass (char).

- void `jit_op_vector_ltep_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: less than or equals pass (char).

- void `jit_op_vector_eqp_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: equals pass (char).

- void `jit_op_vector_neqp_char` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: not equals pass (char).

- void `jit_op_vector_gtp_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: greater than pass (float32).

- void `jit_op_vector_gtep_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: greater than or equals pass (float32).

- void `jit_op_vector_ltp_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: less than pass (float32).

- void `jit_op_vector_ltep_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: less than or equals pass (float32).

- void `jit_op_vector_eqp_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: equals pass (float32).

- void `jit_op_vector_neqp_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: not equals pass (float32).

- void `jit_op_vector_gtp_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: greater than pass (float64).
- void `jit_op_vector_gtep_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: greater than or equals pass (float64).
- void `jit_op_vector_ltp_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: less than pass (float64).
- void `jit_op_vector_ltep_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: less than or equals pass (float64).
- void `jit_op_vector_eqp_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: equals pass (float64).
- void `jit_op_vector_neqp_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: not equals pass (float64).
- void `jit_op_vector_gtp_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: greater than pass (long).
- void `jit_op_vector_gtep_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: greater than or equals pass (long).
- void `jit_op_vector_ltp_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: less than pass (long).
- void `jit_op_vector_ltep_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: less than or equals pass (long).
- void `jit_op_vector_eqp_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: equals pass (long).
- void `jit_op_vector_neqp_long` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: not equals pass (long).
- void `jit_op_vector_sin_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: sine (float32).

- void `jit_op_vector_cos_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: cosine (float32).
- void `jit_op_vector_tan_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: tangent (float32).
- void `jit_op_vector_asin_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: arcsine (float32).
- void `jit_op_vector_acos_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: arccosine (float32).
- void `jit_op_vector_atan_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: arctangent (float32).
- void `jit_op_vector_atan2_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: arctangent (float32).
- void `jit_op_vector_sinh_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: hyperbolic sine (float32).
- void `jit_op_vector_cosh_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: hyperbolic cosine (float32).
- void `jit_op_vector_tanh_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: hyperbolic tangent (float32).
- void `jit_op_vector_asinh_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: hyperbolic arcsine (float32).
- void `jit_op_vector_acosh_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: hyperbolic arccosine (float32).
- void `jit_op_vector_atanh_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: hyperbolic arctangent (float32).
- void `jit_op_vector_exp_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: exponent (float32).

- void `jit_op_vector_exp2_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: exponent base 10 (float32).

- void `jit_op_vector_log_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: logarithm (float32).

- void `jit_op_vector_log2_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: logarithm base 2 (float32).

- void `jit_op_vector_log10_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: logarithm base 10 (float32).

- void `jit_op_vector_hypot_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: hypotenuse (float32).

- void `jit_op_vector_pow_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: power (float32).

- void `jit_op_vector_sqrt_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: square root (float32).

- void `jit_op_vector_ceil_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: ceiling (float32).

- void `jit_op_vector_floor_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: floor (float32).

- void `jit_op_vector_round_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: round (float32).

- void `jit_op_vector_trunc_float32` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: truncate (float32).

- void `jit_op_vector_sin_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: sine (float64).

- void `jit_op_vector_cos_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: cosine (float64).
- void `jit_op_vector_tan_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: tangent (float64).
- void `jit_op_vector_asin_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: arcsine (float64).
- void `jit_op_vector_acos_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: arccosine (float64).
- void `jit_op_vector_atan_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: arctangent (float64).
- void `jit_op_vector_atan2_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Binary operator: arctangent (float64).
- void `jit_op_vector_sinh_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: hyperbolic sine (float64).
- void `jit_op_vector_cosh_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: hyperbolic cosine (float64).
- void `jit_op_vector_tanh_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: hyperbolic tangent (float64).
- void `jit_op_vector_asinh_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: hyperbolic arcsine (float64).
- void `jit_op_vector_acosh_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: hyperbolic arccosine (float64).
- void `jit_op_vector_atanh_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: hyperbolic arctangent (float64).
- void `jit_op_vector_exp_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)
Unary operator: exponent (float64).

- void `jit_op_vector_exp2_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: exponent base 2(float64).
- void `jit_op_vector_log_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: logarithm (float64).
- void `jit_op_vector_log2_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: logarithm base 2 (float64).
- void `jit_op_vector_log10_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: logarithm base 10 (float64).
- void `jit_op_vector_hypot_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: hypotenuse (float64).
- void `jit_op_vector_pow_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Binary operator: power (float64).
- void `jit_op_vector_sqrt_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: square root (float64).
- void `jit_op_vector_ceil_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: ceiling (float64).
- void `jit_op_vector_floor_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: floor (float64).
- void `jit_op_vector_round_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: round (float64).
- void `jit_op_vector_trunc_float64` (long n, void *vecdata, `t_jit_op_info` *in0, `t_jit_op_info` *in1, `t_jit_op_info` *out)

Unary operator: truncate (float64).

33.83.1 Function Documentation

33.83.1.1 void `jit_op_vector_abs_float32` (long *n*, void * *vecdata*, `t_jit_op_info` * *in0*, `t_jit_op_info` * *in1*, `t_jit_op_info` * *out*)

Unary operator: absolute value (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.2 void jit_op_vector_abs_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: absolute value (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.3 void jit_op_vector_abs_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: absolute value (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.4 void jit_op_vector_absdiff_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: absolute difference (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.5 void jit_op_vector_absdiff_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: absolute difference (float32).

Parameters

- n* length of vectors
- vecdata* ignored
- in0* left input pointer and stride
- in1* right input pointer and stride
- out* output pointer and stride

33.83.1.6 void jit_op_vector_absdiff_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: absolute difference (float64).

Parameters

- n* length of vectors
- vecdata* ignored
- in0* left input pointer and stride
- in1* right input pointer and stride
- out* output pointer and stride

33.83.1.7 void jit_op_vector_absdiff_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: absolute difference (long).

Parameters

- n* length of vectors
- vecdata* ignored
- in0* left input pointer and stride
- in1* right input pointer and stride
- out* output pointer and stride

33.83.1.8 void jit_op_vector_acos_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: arccosine (float32).

Parameters

- n* length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.9 void jit_op_vector_acos_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: arccosine (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.10 void jit_op_vector_acosh_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: hyperbolic arccosine (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.11 void jit_op_vector_acosh_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: hyperbolic arccosine (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.12 void jit_op_vector_add_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: addition (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.13 void jit_op_vector_add_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: addition (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.14 void jit_op_vector_add_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: addition (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.15 void jit_op_vector_add_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: addition (long).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.16 void jit_op_vector_adds_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: saturated addition (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.17 void jit_op_vector_and_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: logical and (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.18 void jit_op_vector_and_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: logical and (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.19 void jit_op_vector_and_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: logical and (float64).

Parameters

n length of vectors

vecdata ignored

in0 left input pointer and stride

in1 right input pointer and stride

out output pointer and stride

33.83.1.20 void jit_op_vector_and_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: logical and (long).

Parameters

n length of vectors

vecdata ignored

in0 left input pointer and stride

in1 right input pointer and stride

out output pointer and stride

33.83.1.21 void jit_op_vector_asin_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: arcsine (float32).

Parameters

n length of vectors

vecdata ignored

in0 left input pointer and stride

in1 right input pointer and stride

out output pointer and stride

33.83.1.22 void jit_op_vector_asin_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: arcsine (float64).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.23 void jit_op_vector_asinh_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: hyperbolic arcsine (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.24 void jit_op_vector_asinh_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: hyperbolic arcsine (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.25 void jit_op_vector_atan2_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: arctangent (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.26 void jit_op_vector_atan2_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: arctangent (float64).

Parameters

- n* length of vectors
- vecdata* ignored
- in0* left input pointer and stride
- in1* right input pointer and stride
- out* output pointer and stride

33.83.1.27 void jit_op_vector_atan_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: arctangent (float32).

Parameters

- n* length of vectors
- vecdata* ignored
- in0* left input pointer and stride
- in1* right input pointer and stride
- out* output pointer and stride

33.83.1.28 void jit_op_vector_atan_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: arctangetn (float64).

Parameters

- n* length of vectors
- vecdata* ignored
- in0* left input pointer and stride
- in1* right input pointer and stride
- out* output pointer and stride

33.83.1.29 void jit_op_vector_atanh_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: hyperbolic arctangent (float32).

Parameters

- n* length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.30 void jit_op_vector_atanh_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: hyperbolic arctangent (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.31 void jit_op_vector_avg_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: average (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.32 void jit_op_vector_avg_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: average (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.33 void jit_op_vector_avg_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: average (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.34 void jit_op_vector_avg_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: average (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.35 void jit_op_vector_bitand_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: bitwise and (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.36 void jit_op_vector_bitand_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: bitwise and (long).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.37 void jit_op_vector_bitnot_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: bitwise not (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.38 void jit_op_vector_bitnot_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: bitwise not (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.39 void jit_op_vector_bitor_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: bitwise or (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.40 void jit_op_vector_bitor_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: bitwise or (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.41 void jit_op_vector_bitxor_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: bitwise exclusive or (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.42 void jit_op_vector_bitxor_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: bitwise exclusive or (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.43 void jit_op_vector_ceil_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: ceiling (float32).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.44 void jit_op_vector_ceil_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: ceiling (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.45 void jit_op_vector_cos_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: cosine (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.46 void jit_op_vector_cos_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: cosine (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.47 void jit_op_vector_cosh_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: hyperbolic cosine (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.48 void jit_op_vector_cosh_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: hyperbolic cosine (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.49 void jit_op_vector_div_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: division (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.50 void jit_op_vector_div_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: division (float32).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.51 void jit_op_vector_div_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: division (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.52 void jit_op_vector_div_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: division (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.53 void jit_op_vector_eq_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: equals (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.54 void jit_op_vector_eq_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: equals (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.55 void jit_op_vector_eq_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: equals (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.56 void jit_op_vector_eq_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: equals (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.57 void jit_op_vector_eqp_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: equals pass (char).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.58 void jit_op_vector_eqp_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: equals pass (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.59 void jit_op_vector_eqp_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: equals pass (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.60 void jit_op_vector_eqp_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: equals pass (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.61 void jit_op_vector_exp2_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: exponent base 10 (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.62 void jit_op_vector_exp2_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: exponent base 2(float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.63 void jit_op_vector_exp_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: exponent (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.64 void jit_op_vector_exp_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: exponent (float64).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.65 void jit_op_vector_flipdiv_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: flipped division (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.66 void jit_op_vector_flipdiv_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: flipped division (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.67 void jit_op_vector_flipdiv_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: flipped division (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.68 void jit_op_vector_flipdiv_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: flipped division (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.69 void jit_op_vector_flipmod_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: flipped modulo (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.70 void jit_op_vector_flipmod_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: flipped modulo (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.71 void jit_op_vector_flipmod_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: flipped modulo (float64).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.72 void jit_op_vector_flipmod_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: flipped modulo (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.73 void jit_op_vector_flippass_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: flipped pass (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.74 void jit_op_vector_flippass_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: flipped pass (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.75 void jit_op_vector_flippass_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: flipped pass (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.76 void jit_op_vector_flippass_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: flipped pass (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.77 void jit_op_vector_flipsub_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: flipped subtraction (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.78 void jit_op_vector_flipsub_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: flipped subtraction (float32).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.79 void jit_op_vector_flipsub_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: flipped subtraction (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.80 void jit_op_vector_floor_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: floor (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.81 void jit_op_vector_floor_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: floor (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.82 void jit_op_vector_fold_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: fold (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.83 void jit_op_vector_fold_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: fold (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.84 void jit_op_vector_gt_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.85 void jit_op_vector_gt_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than (float32).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.86 void jit_op_vector_gt_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.87 void jit_op_vector_gt_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.88 void jit_op_vector_gte_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than or equals (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.89 void jit_op_vector_gte_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than or equals (float32).

Parameters

- n* length of vectors
- vecdata* ignored
- in0* left input pointer and stride
- in1* right input pointer and stride
- out* output pointer and stride

33.83.1.90 void jit_op_vector_gte_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than or equals (float64).

Parameters

- n* length of vectors
- vecdata* ignored
- in0* left input pointer and stride
- in1* right input pointer and stride
- out* output pointer and stride

33.83.1.91 void jit_op_vector_gte_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than or equals (long).

Parameters

- n* length of vectors
- vecdata* ignored
- in0* left input pointer and stride
- in1* right input pointer and stride
- out* output pointer and stride

33.83.1.92 void jit_op_vector_gtep_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than or equals pass (char).

Parameters

- n* length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.93 void jit_op_vector_gtep_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than or equals pass (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.94 void jit_op_vector_gtep_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than or equals pass (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.95 void jit_op_vector_gtep_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than or equals pass (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.96 void jit_op_vector_gtp_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than pass (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.97 void jit_op_vector_gtp_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than pass (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.98 void jit_op_vector_gtp_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than pass (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.99 void jit_op_vector_gtp_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: greater than pass (long).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.100 void jit_op_vector_hypot_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: hypotenuse (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.101 void jit_op_vector_hypot_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: hypotenuse (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.102 void jit_op_vector_log10_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: logarithm base 10 (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.103 void jit_op_vector_log10_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: logarithm base 10 (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.104 void jit_op_vector_log2_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: logarithm base 2(float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.105 void jit_op_vector_log2_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: logarithm base 2 (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.106 void jit_op_vector_log_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: logarithm (float32).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.107 void jit_op_vector_log_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: logarithm (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.108 void jit_op_vector_lshift_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: bitwise left shift (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.109 void jit_op_vector_lshift_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: bitwise left shift (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.110 void jit_op_vector_lt_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.111 void jit_op_vector_lt_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.112 void jit_op_vector_lt_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.113 void jit_op_vector_lt_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than (long).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.114 void jit_op_vector_lte_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than or equals (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.115 void jit_op_vector_lte_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than or equals (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.116 void jit_op_vector_lte_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than or equals (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.117 void jit_op_vector_lte_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than or equals (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.118 void jit_op_vector_ltep_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than or equals pass (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.119 void jit_op_vector_ltep_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than or equals pass (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.120 void jit_op_vector_ltep_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than or equals pass (float64).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.121 void jit_op_vector_ltep_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than or equals pass (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.122 void jit_op_vector_ltp_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than pass (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.123 void jit_op_vector_ltp_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than pass (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.124 void jit_op_vector_ltp_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than pass (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.125 void jit_op_vector_ltp_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: less than pass (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.126 void jit_op_vector_max_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: maximum (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.127 void jit_op_vector_max_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: maximum (float32).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.128 void jit_op_vector_max_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: maximum (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.129 void jit_op_vector_max_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: maximum (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.130 void jit_op_vector_min_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: minimum (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.131 void jit_op_vector_min_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: minimum (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.132 void jit_op_vector_min_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: minimum (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.133 void jit_op_vector_min_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: minimum (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.134 void jit_op_vector_mod_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: modulo (char).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.135 void jit_op_vector_mod_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: modulo (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.136 void jit_op_vector_mod_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: modulo (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.137 void jit_op_vector_mod_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: modulo (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.138 void jit_op_vector_mult_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: multiplication (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.139 void jit_op_vector_mult_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: multiplication (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.140 void jit_op_vector_mult_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: multiplication (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.141 void jit_op_vector_mult_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: multiplication (long).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.142 void jit_op_vector_neq_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: not equals (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.143 void jit_op_vector_neq_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: not equals (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.144 void jit_op_vector_neq_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: not equals (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.145 void jit_op_vector_neq_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: not equals (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.146 void jit_op_vector_neqp_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: not equals pass (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.147 void jit_op_vector_neqp_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: not equals pass (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.148 void jit_op_vector_neqp_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: not equals pass (float64).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.149 void jit_op_vector_neqp_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: not equals pass (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.150 void jit_op_vector_not_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: logical not (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.151 void jit_op_vector_not_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: logical not (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.152 void jit_op_vector_not_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: logical not (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.153 void jit_op_vector_not_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: logical not (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.154 void jit_op_vector_or_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: logical or (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.155 void jit_op_vector_or_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: logical or (float32).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.156 void jit_op_vector_or_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: logical or (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.157 void jit_op_vector_or_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: logical or (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.158 void jit_op_vector_pass_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: pass (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.159 void jit_op_vector_pass_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: pass (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.160 void jit_op_vector_pass_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: pass (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.161 void jit_op_vector_pass_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: pass (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.162 void jit_op_vector_pow_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: power (float32).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.163 void jit_op_vector_pow_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: power (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.164 void jit_op_vector_round_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: round (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.165 void jit_op_vector_round_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: round (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.166 void jit_op_vector_rshift_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: bitwise right shift (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.167 void jit_op_vector_rshift_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: bitwise right shift (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.168 void jit_op_vector_sin_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: sine (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.169 void jit_op_vector_sin_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: sine (float64).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.170 void jit_op_vector_sinh_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: hyperbolic sine (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.171 void jit_op_vector_sinh_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: hyperbolic sine (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.172 void jit_op_vector_sqrt_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: square root (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.173 void jit_op_vector_sqrt_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: square root (float64).

Parameters

n length of vectors

vecdata ignored

in0 left input pointer and stride

in1 right input pointer and stride

out output pointer and stride

33.83.1.174 void jit_op_vector_sub_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: subtraction (char).

Parameters

n length of vectors

vecdata ignored

in0 left input pointer and stride

in1 right input pointer and stride

out output pointer and stride

33.83.1.175 void jit_op_vector_sub_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: subtraction (float32).

Parameters

n length of vectors

vecdata ignored

in0 left input pointer and stride

in1 right input pointer and stride

out output pointer and stride

33.83.1.176 void jit_op_vector_sub_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: subtraction (float64).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.177 void jit_op_vector_sub_long (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: subtraction (long).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.178 void jit_op_vector_subs_char (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: saturated subtraction (char).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.179 void jit_op_vector_tan_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: tangent (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.180 void jit_op_vector_tan_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: tangent (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.181 void jit_op_vector_tanh_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: hyperbolic tangent (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.182 void jit_op_vector_tanh_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: hyperbolic tangent (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.183 void jit_op_vector_trunc_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: truncate (float32).

Parameters

n length of vectors

vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.184 void jit_op_vector_trunc_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Unary operator: truncate (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.185 void jit_op_vector_wrap_float32 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: wrap (float32).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.83.1.186 void jit_op_vector_wrap_float64 (long *n*, void * *vecdata*, t_jit_op_info * *in0*, t_jit_op_info * *in1*, t_jit_op_info * *out*)

Binary operator: wrap (float64).

Parameters

n length of vectors
vecdata ignored
in0 left input pointer and stride
in1 right input pointer and stride
out output pointer and stride

33.84 QuickTime Codec Module

Collaboration diagram for QuickTime Codec Module:



Functions

- `t_jit_err jit_bt_codec_getcodeclist_video (void *x, void *attr, long *ac, t_atom **av)`
Drop-in getter function for "codeclist"-type attribute, returns the list of valid video codecs installed on the system.
- `t_jit_err jit_bt_codec_getcodeclist_audio (void *x, void *attr, long *ac, t_atom **av)`
Drop-in getter function for "codeclist"-type attribute, returns the list of valid sound compressor codecs installed on the system.
- `t_jit_err jit_bt_codec_getcodeclist_gfx (void *x, void *attr, long *ac, t_atom **av)`
Drop-in getter function for "codeclist"-type attribute, returns the list of valid graphic exporter codecs installed on the system.
- `void jit_bt_codec_getcodeclist_video_raw (long *count, t_symbol ***names)`
Returns the list of valid video codecs installed on the system.
- `void jit_bt_codec_getcodeclist_audio_raw (long *count, t_symbol ***names)`
Returns the list of valid sound compressor codecs installed on the system.
- `void jit_bt_codec_getcodeclist_gfx_raw (long *count, t_symbol ***names)`
Returns the list of valid graphic exporter codecs installed on the system.
- `t_symbol * jit_bt_codec_qual2sym (long qual)`
Convert a QuickTime Codec Quality value to a human-readable symbol.
- `long jit_bt_codec_sym2qual (t_symbol *s)`
Convert a codec quality symbol to a valid QuickTime Codec Quality value.
- `t_symbol * jit_bt_codec_type2sym_valid (long type)`
Given the four-char type code of a QuickTime video codec, returns a human-readable name, assuming that the codec is installed _and_ is supported by Jitter.
- `t_symbol * jit_bt_codec_type2sym (long type)`
Given the four-char type code of a QuickTime video codec, returns a human-readable name, assuming that the codec is installed.
- `long jit_bt_codec_sym2type_valid (t_symbol *s)`
Given the human-readable name of a QuickTime video codec, returns a four-char code, assuming that the codec is installed _and_ is supported by Jitter.
- `long jit_bt_codec_sym2type (t_symbol *s)`

Given the human-readable name of a QuickTime video codec, returns a four-char code, assuming that the codec is installed.

- **t_symbol * jit_qt_codec_acodec2sym (long type)**

Given the four-char type code of a QuickTime audio codec, returns a human-readable name, assuming that the codec is installed.

- **long jit_qt_codec_sym2acodec (t_symbol *s)**

Given the human-readable name of a QuickTime audio codec, returns a four-char code, assuming that the codec is installed.

33.84.1 Function Documentation

33.84.1.1 t_symbol* jit_qt_codec_acodec2sym (long type)

Given the four-char type code of a QuickTime audio codec, returns a human-readable name, assuming that the codec is installed.

Parameters

type four-char type code of a QuickTime Codec

Returns

t_symbol pointer containing a human-readable codec name

33.84.1.2 t_jit_err jit_qt_codec_getcodeclist_audio (void *x, void *attr, long *ac, t_atom **av)

Drop-in getter function for "codeclist"-type attribute, returns the list of valid sound compressor codecs installed on the system.

Parameters

x undocumented

attr undocumented

ac undocumented

av undocumented

Returns

t_jit_err error code

33.84.1.3 void jit_qt_codec_getcodeclist_audio_raw (long *count, t_symbol ***names)

Returns the list of valid sound compressor codecs installed on the system.

Parameters

count (on output) number of codec names returned

names (on output) **t_symbol** pointer array of (count) length, containing the names of valid installed codecs

33.84.1.4 `t_jit_err jit_qt_codec_getcodeclist_gfx (void *x, void *attr, long *ac, t_atom **av)`

Drop-in getter function for "codeclist"-type attribute, returns the list of valid graphic exporter codecs installed on the system.

Parameters

x undocumented
attr undocumented
ac undocumented
av undocumented

Returns

`t_jit_err` error code

33.84.1.5 `void jit_qt_codec_getcodeclist_gfx_raw (long *count, t_symbol *names)`**

Returns the list of valid graphic exporter codecs installed on the system.

Parameters

count (on output) number of codec names returned
names (on output) `t_symbol` pointer array of (count) length, containing the names of valid installed codecs

33.84.1.6 `t_jit_err jit_qt_codec_getcodeclist_video (void *x, void *attr, long *ac, t_atom **av)`

Drop-in getter function for "codeclist"-type attribute, returns the list of valid video codecs installed on the system.

Parameters

x undocumented
attr undocumented
ac undocumented
av undocumented

Returns

`t_jit_err` error code

33.84.1.7 `void jit_qt_codec_getcodeclist_video_raw (long *count, t_symbol *names)`**

Returns the list of valid video codecs installed on the system.

Parameters

count (on output) number of codec names returned
names (on output) `t_symbol` pointer array of (count) length, containing the names of valid installed codecs

33.84.1.8 `t_symbol* jit_qt_codec_qual2sym (long qual)`

Convert a QuickTime Codec Quality value to a human-readable symbol.

Parameters

qual QuickTime Codec Quality

Returns

`t_symbol` pointer containing a human-readable quality name (lossless, min, low, normal, high, max)

33.84.1.9 `long jit_qt_codec_sym2acodec (t_symbol * s)`

Given the human-readable name of a QuickTime audio codec, returns a four-char code, assuming that the codec is installed.

Parameters

s human-readable name of a QuickTime Codec (as returned by "jit_qt_codec_getcodeclist_video_-raw" or similar)

Returns

long four-char code

33.84.1.10 `long jit_qt_codec_sym2qual (t_symbol * s)`

Convert a codec quality symbol to a valid QuickTime Codec Quality value.

Parameters

s codec quality name (lossless, min, low, normal, high, max)

Returns

QuickTime Codec Quality

33.84.1.11 `long jit_qt_codec_sym2type (t_symbol * s)`

Given the human-readable name of a QuickTime video codec, returns a four-char code, assuming that the codec is installed.

Parameters

s human-readable name of a QuickTime Codec (as returned by "jit_qt_codec_getcodeclist_video_-raw" or similar)

Returns

long four-char code

33.84.1.12 long jit_qt_codec_sym2type_valid (t_symbol * s)

Given the human-readable name of a QuickTime video codec, returns a four-char code, assuming that the codec is installed _and_ is supported by Jitter.

Parameters

s human-readable name of a QuickTime Codec (as returned by "jit_qt_codec_getcodeclist_video_-raw" or similar)

Returns

long four-char code

33.84.1.13 t_symbol* jit_qt_codec_type2sym (long *type*)

Given the four-char type code of a QuickTime video codec, returns a human-readable name, assuming that the codec is installed.

Parameters

type four-char type code of a QuickTime Codec

Returns

[t_symbol](#) pointer containing a human-readable codec name

33.84.1.14 t_symbol* jit_qt_codec_type2sym_valid (long *type*)

Given the four-char type code of a QuickTime video codec, returns a human-readable name, assuming that the codec is installed _and_ is supported by Jitter.

Parameters

type four-char type code of a QuickTime Codec

Returns

[t_symbol](#) pointer containing a human-readable codec name

33.85 jit.qt.movie Module

Collaboration diagram for jit.qt.movie Module:



Functions

- `t_jit/qt_movie * jit_qt_movie_new (long width, long height)`
Constructs instance of t_jit/qt_movie.
- `long jit_qt_movie_matrix_calc (t_jit/qt_movie *x, void *inputs, void *outputs)`
matrix_calc method for the jit.qt.movie object
- `t_jit_err jit_qt_movie_matrix_to_image (t_jit/qt_movie *x, void *o, short ac, t_atom *av, t_graphic_exportsettings *gs)`
Export a Jitter matrix to a QuickTime-compatible image file.
- `t_jit_err jit_qt_movie_read_typed (t_jit/qt_movie *x, t_symbol *s, long ac, t_atom *av, t_atom *rv)`
Read a QuickTime Movie.

33.85.1 Function Documentation

33.85.1.1 long jit_qt_movie_matrix_calc (t_jit/qt_movie * *x*, void * *inputs*, void * *outputs*)

matrix_calc method for the jit.qt.movie object

Parameters

- x* t_jit/qt_movie object pointer
- inputs* input list (unused)
- outputs* output list (should be or contain 1 t_jit_matrix object)

Returns

t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of a t_jit/qt_movie object.

33.85.1.2 t_jit_err jit_qt_movie_matrix_to_image (t_jit/qt_movie * *x*, void * *o*, short *ac*, t_atom * *av*, t_graphic_exportsettings * *gs*)

Export a Jitter matrix to a QuickTime-compatible image file.

Parameters

x t_jit_qt_movie object pointer
o t_jit_matrix pointer
ac argument count
av argument vector
gs undocumented

Returns

t_jit_err error code

Optional arguments are file type, desired file name/file path for exported image file, and a show settings dialog flag (0/1). Format is essentially the same as that of "exportimage" method to jit.qt.movie, as documented in the Jitter Reference.

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of a t_jit_qt_movie object.

33.85.1.3 t_jit_qt_movie* jit_qt_movie_new (long *width*, long *height*)

Constructs instance of t_jit_qt_movie.

Parameters

width output matrix width
height output matrix height

Returns

t_jit_qt_movie object pointer

Warning

This function is not exported, but is provided for reference when calling via jit_object_new.

33.85.1.4 t_jit_err jit_qt_movie_read_typed (t_jit_qt_movie **x*, t_symbol **s*, long *ac*, t_atom **av*, t_atom **rv*)

Read a QuickTime Movie.

Parameters

x t_jit_qt_movie object pointer
s t_symbol pointer containing method name ("read" or "asyncread", "import", "importfile" or "asyncimport")
ac argument count
av argument vector

rv (optional) [t_atom](#) pointer, on output, will be of type A_OBJ and contain a [t_atomarray](#) object with any return values

Returns

[t_jit_err](#) error code

Optional argument is the file name/file path/URL or "scrap" to load.

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method_typed` on an instance of a [t_jit_qt_movie](#) object.

33.86 jit.qt.record Module

Collaboration diagram for jit.qt.record Module:



Functions

- `t_jit_err jit/qt_record_matrix_calc (t_jit/qt_record *x, void *inputs, void *outputs)`
matrix_calc method for the jit.qt.record object
- `t_jit/qt_record * jit/qt_record_new (long width, long height)`
Constructs instance of t_jit/qt_record.

33.86.1 Function Documentation

33.86.1.1 `t_jit_err jit/qt_record_matrix_calc (t_jit/qt_record *x, void *inputs, void *outputs)`

matrix_calc method for the jit.qt.record object

Parameters

`x` t_jit/qt_record object pointer
`inputs` input list (should be or contain 1 t_jit_matrix object)
`outputs` output list (unused)

Returns

`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of a `t_jit/qt_movie` object.

33.86.1.2 `t_jit/qt_record* jit/qt_record_new (long width, long height)`

Constructs instance of `t_jit/qt_record`.

Parameters

`width` output matrix width
`height` output matrix height

Returns

`t_jit/qt_record` object pointer

Warning

This function is not exported, but is provided for reference when calling via `jit_object_new`.

33.87 QuickTime Utilities Module

Collaboration diagram for QuickTime Utilities Module:



Functions

- void [jit_gworld_clear](#) (GWorldPtr gp, long c)
Set all pixels in a QuickDraw GWorld to a specified 32-bit value.
- long [jit_gworld_can_coerce_matrix](#) (t_gworld_conv_info *gc, void *m)
Determine whether a Jitter matrix can be wrapped in a QuickDraw GWorld (without a copy).
- long [jit_gworld_matrix_equal_dim](#) (GWorldPtr gp, void *m)
Test for equality of dimensions between a QuickDraw GWorld and a Jitter matrix Note: supports UYVY matrices.
- t_jit_err [jit_coerce_matrix_pixmap](#) (void *m, PixMap *pm)
Generate a QuickDraw PixMap for a given Jitter matrix.
- t_jit_err [jit_bt_utils_moviedataref_create](#) (t_symbol **sname, short *path, Handle *dataRef, OSType *dataRefType)
Creates a new Data Reference from a file path, returning it and the filename/path pair.
- Movie [jit_bt_utils_moviefile_create](#) (t_symbol **sname, short *path, long flags, DataHandler *dhandler)
Creates a new QuickTime Movie from a file path, optionally returning the Data Handler and/or filename/path pair.
- Boolean [jit_bt_utils_tempfile](#) (char *name, Handle *dataRef, OSType *dataRefType)
Returns a QuickTime-compatible Data Reference for a named file in the system's temporary files directory.
- Movie [jit_bt_utils_tempmoviefile_create](#) (t_symbol **sname, short *path, long flags, DataHandler *dhandler)
Creates a new QuickTime Movie in the system's temporary file directory, optionally returning the movie's data handler.
- long [jit_bt_utils_moviefile_close](#) (Movie movie, DataHandler dhandler)
Closes a QuickTime Movie previously created with jit_bt_utils_moviefile_create or jit_bt_utils_tempmoviefile_create, adding the necessary movie resources.
- Track [jit_bt_utils_trackmedia_add](#) (Movie movie, long type, Rect *trackframe, long vol, long timescale)
Adds a new Track, with associated Media, to a QuickTime Movie.
- Media [jit_bt_utils_trackmedia_get](#) (Track track)
Returns the Media for a specified Track.

- long [jit_qt_utils_trackmedia_dispose](#) (Track track)
Removes a Track, with associated Media, from a QuickTime Movie.
- void [jit_qt_utils_type2str](#) (OSType type, char *typestr)
Given a four-char type code, return a 0-terminated C string.
- OSType [jit_qt_utils_str2type](#) (char *typestr)
Given a C string, return a four-char code.
- void [jit_qt_utils_trackname_set](#) (Track track, t_symbol *s)
Set the name of a QuickTime Track.
- t_symbol * [jit_qt_utils_trackname_get](#) (Track track)
Get the name of a QuickTime Track.
- t_symbol * [jit_qt_utils_tracktype_get](#) (Track track)
Get the Media Type name from a QuickTime Track.
- t_symbol * [jit_qt_utils_tracktypecode_get](#) (Track track)
Get the four-char code for a Track's Media Type, formatted as a symbol.

33.87.1 Function Documentation

33.87.1.1 t_jit_err jit_coerce_matrix_pixmap (void * *m*, PixMap * *pm*)

Generate a QuickDraw PixMap for a given Jitter matrix.

Parameters

m input t_jit_matrix pointer
pm on output, a pointer to the generated PixMap

Returns

t_jit_err error code

Warning

The matrix should be locked previous to this call, and unlocked afterward.

33.87.1.2 long jit_gworld_can_coerce_matrix (t_gworld_conv_info * *gc*, void * *m*)

Determine whether a Jitter matrix can be wrapped in a QuickDraw GWorld (without a copy).

Parameters

gc optional pointer to a t_gworld_conv_info struct
m input t_jit_matrix pointer

Returns

long success code (1 = can coerce, 0 = cannot coerce)

33.87.1.3 void jit_gworld_clear (GWorldPtr *gp*, long *c*)

Set all pixels in a QuickDraw GWorld to a specified 32-bit value.

Parameters

gp QuickDraw GWorldPtr
c clear color

33.87.1.4 long jit_gworld_matrix_equal_dim (GWorldPtr *gp*, void * *m*)

Test for equality of dimensions between a QuickDraw GWorld and a Jitter matrix Note: supports UYVY matrices.

Parameters

gp input GWorldPtr
m input t_jit_matrix pointer

Returns

long success code (1 = dims are equal, 0 = dims are not equal)

33.87.1.5 t_jit_err jit_qt_utils_moviedataref_create (t_symbol ** *sname*, short * *path*, Handle * *dataRef*, OSType * *dataRefType*)

Creates a new Data Reference from a file path, returning it and the filename/path pair.

Parameters

sname (in/out) in: file name or fully qualified path in; out: file name of opened movie file
path (in/out) in: only necessary if sname is unqualified; out: path of opened movie file
dataRef (on output) QuickTime-compatible Data Reference for the specified file name, must be disposed by the caller
dataRefType (on output) Data Reference type

Returns

t_jit_err error

33.87.1.6 long jit_qt_utils_moviefile_close (Movie *movie*, DataHandler *dhandler*)

Closes a QuickTime Movie previously created with jit_qt_utils_moviefile_create or jit_qt_utils_tempmoviefile_create, adding the necessary movie resources.

Parameters

movie QuickTime Movie, as returned from one of the above-named functions
dhandler data handler for the Movie, as returned from one of the above-named functions

Returns

long QuickTime error code

33.87.1.7 Movie jit_qt_utils_moviefile_create (t_symbol ** *sname*, short * *path*, long *flags*, DataHandler * *dhandler*)

Creates a new QuickTime Movie from a file path, optionally returning the Data Handler and/or filename-/path pair.

Parameters

sname (in/out) in: file name or fully qualified path in; out: file name of opened movie file

path (in/out, optional) in: only necessary if *sname* is unqualified; out: path of opened movie file

flags movie file creation flags (see QuickTime Documentation for more information) if no flags are specified, the following flags are used: createMovieFileDeleteCurFile | createMovieFileDontCreateResFile

dhandler (on output, optional) data handler for the opened movie file

Returns

Movie QuickTime Movie

33.87.1.8 OSType jit_qt_utils_str2type (char * *typestr*)

Given a C string, return a four-char code.

Parameters

typestr C string

Returns

OSType four-char code

33.87.1.9 Boolean jit_qt_utils_tempfile (char * *name*, Handle * *dataRef*, OSType * *dataRefType*)

Returns a QuickTime-compatible Data Reference for a named file in the system's temporary files directory.

Parameters

name file name

dataRef (on output) QuickTime-compatible Data Reference for the specified file name, must be disposed by the caller

dataRefType (on output) Data Reference type

Returns

Boolean success (true) or failure (false)

33.87.1.10 Movie jit_qt_utils_tempmoviefile_create (*t_symbol ** fname*, *short * path*, *long flags*, *DataHandler * dhandler*)

Creates a new QuickTime Movie in the system's temporary file directory, optionally returning the movie's data handler.

Parameters

fname (in/out) in: file name or fully qualified path in; out: file name of opened movie file

path (in/out, optional) in: only necessary if fname is unqualified; out: path of opened movie file

flags movie file creation flags (see QuickTime Documentation for more information) if no flags are specified, the following flags are used: createMovieFileDeleteCurFile | createMovieFileDontCreateResFile

dhandler (on output, optional) data handler for the opened movie file

Returns

Movie QuickTime Movie

33.87.1.11 Track jit_qt_utils_trackmedia_add (*Movie movie*, *long type*, *Rect * trackframe*, *long vol*, *long timescale*)

Adds a new Track, with associated Media, to a QuickTime Movie.

Parameters

movie QuickTime Movie

type four-char code specifying the track/media type to be added (see QuickTime Documentation)

trackframe the new Track's Rect, relative to the Movie's Rect

vol initial value for the sound volume in the new Track

timescale the new Track's timescale

Returns

Track QuickTime Track

33.87.1.12 long jit_qt_utils_trackmedia_dispose (*Track track*)

Removes a Track, with associated Media, from a QuickTime Movie.

Parameters

track QuickTime Track

Returns

long QuickTime error code

33.87.1.13 Media jit_qt_utils_trackmedia_get (Track *track*)

Returns the Media for a specified Track.

Parameters

track QuickTime Track

Returns

Media QuickTime Media

33.87.1.14 t_symbol* jit_qt_utils_trackname_get (Track *track*)

Get the name of a QuickTime Track.

Parameters

track QuickTime Track

Returns

t_symbol pointer containing Track's name

33.87.1.15 void jit_qt_utils_trackname_set (Track *track*, t_symbol * *s*)

Set the name of a QuickTime Track.

Parameters

track QuickTime Track

s track name

33.87.1.16 t_symbol* jit_qt_utils_tracktype_get (Track *track*)

Get the Media Type name from a QuickTime Track.

Parameters

track QuickTime Track

Returns

t_symbol pointer containing the name of the Track's Media Type

33.87.1.17 t_symbol* jit_qt_utils_tracktypecode_get (Track *track*)

Get the four-char code for a Track's Media Type, formatted as a symbol.

Parameters

track QuickTime Track

Returns

`t_symbol` pointer containing C string representation of the Media Type

33.87.1.18 void jit_qt_utils_type2str (OSType *type*, char * *typestr*)

Given a four-char type code, return a 0-terminated C string.

Parameters

type four-char code

typestr (on output) 0-terminated C string

Chapter 34

Data Structure Documentation

34.1 Ex_ex Struct Reference

```
ex_ex.  
#include <ext_expr.h>
```

Data Fields

- union {
 } ex_cont

content
- long ex_type
type of the node

34.1.1 Detailed Description

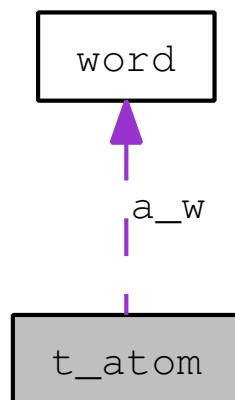
ex_ex.

34.2 t_atom Struct Reference

An atom is a typed datum.

```
#include <ext_mess.h>
```

Collaboration diagram for t_atom:



Data Fields

- short `a_type`
a value as defined in [e_max_atomtypes](#)
- union `word a_w`
the actual data

34.2.1 Detailed Description

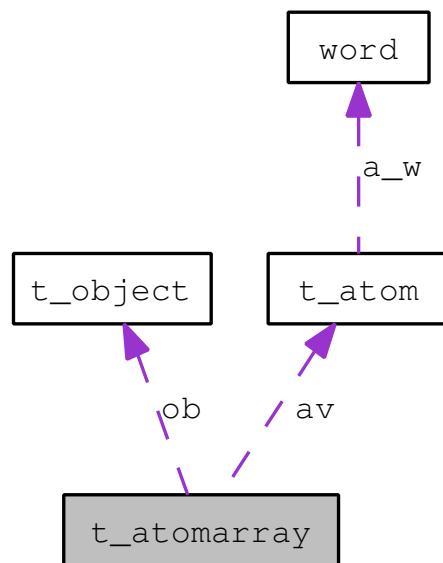
An atom is a typed datum.

34.3 t_atomarray Struct Reference

The atomarray object.

```
#include <ext_atomarray.h>
```

Collaboration diagram for t_atomarray:



34.3.1 Detailed Description

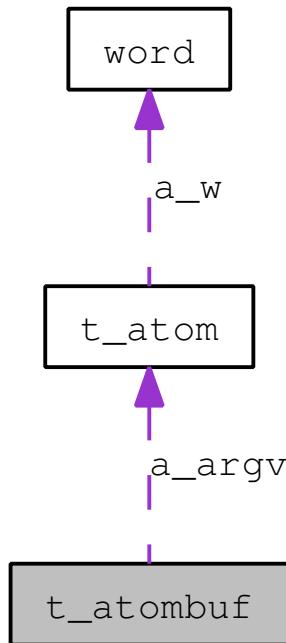
The atomarray object. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

34.4 t_atombuf Struct Reference

The atombuf struct provides a way to pass a collection of atoms.

```
#include <ext_maxtypes.h>
```

Collaboration diagram for t_atombuf:



Data Fields

- long [a_argc](#)
the number of atoms
- [t_atom a_argv](#) [1]
the first of the array of atoms

34.4.1 Detailed Description

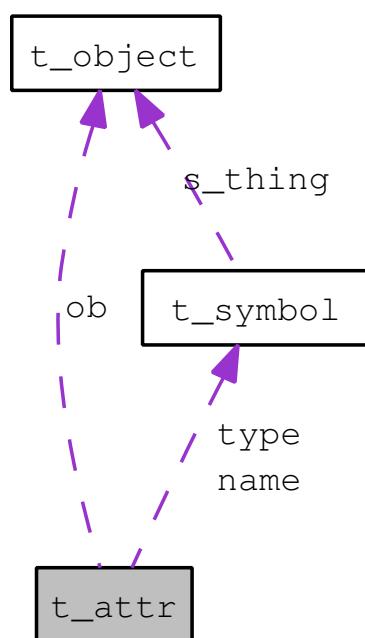
The atombuf struct provides a way to pass a collection of atoms.

34.5 t_attr Struct Reference

Common attr struct.

```
#include <ext_obex.h>
```

Collaboration diagram for t_attr:



34.5.1 Detailed Description

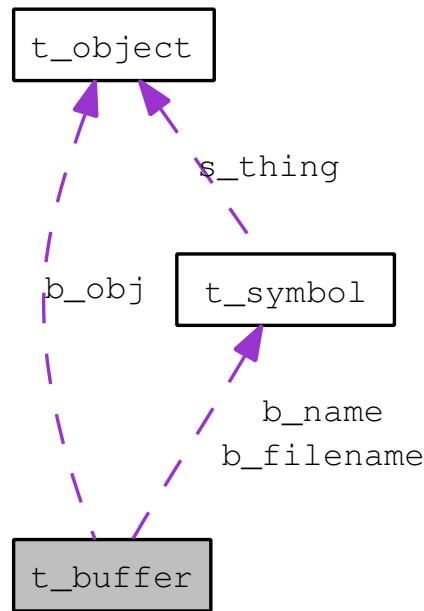
Common attr struct. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

34.6 t_buffer Struct Reference

Data structure for the buffer~ object.

```
#include <buffer.h>
```

Collaboration diagram for t_buffer:



Data Fields

- **t_object b_obj**
doesn't have any signals so it doesn't need to be pxobject
- long **b_valid**
flag is off during read replacement or editing operation
- float * **b_samples**
stored with interleaved channels if multi-channel
- long **b_frames**
*number of sample frames (each one is sizeof(float) * b_nchans bytes)*
- long **b_nchans**
number of channels
- long **b_size**
size of buffer in floats
- float **b_sr**
sampling rate of the buffer

- float **b_loversr**
l / sr
- float **b_msr**
*sr * .001*
- float * **b_memory**
pointer to where memory starts (initial padding for interp)
- **t_symbol * b_name**
name of the buffer
- long **b_susloopstart**
looping info (from AIFF file) in samples
- long **b_susloopend**
looping info (from AIFF file) in samples
- long **b_reloopstart**
looping info (from AIFF file) in samples
- long **b_reloopend**
looping info (from AIFF file) in samples
- long **b_format**
'AIFF' or 'Sd2f'
- **t_symbol * b_filename**
last file read (not written) for readagain message
- long **b_oldnchans**
used for resizing window in case of # of channels change
- long **b_outputbytes**
number of bytes used for output sample (1-4)
- long **b_modtime**
last modified time ("dirty" method)
- struct _buffer * **b_peer**
objects that share this symbol (used as a link in the peers)
- Boolean **b_owner**
b_memory/b_samples "owned" by this object
- long **b_outputfmt**
sample type (A_LONG, A_FLOAT, etc.)
- **t_int32_atomic b_inuse**

objects that use buffer should ATOMIC_INCREMENT / ATOMIC_DECREMENT this in their perform

- void * **b_dspchain**
dspchain used for this instance

34.6.1 Detailed Description

Data structure for the buffer~ object.

34.7 t_celldesc Struct Reference

A dataview cell description.

```
#include <jdataview.h>
```

34.7.1 Detailed Description

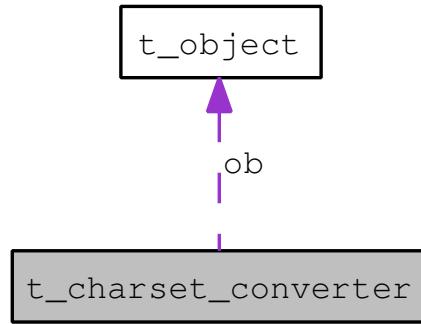
A dataview cell description.

34.8 t_charset_converter Struct Reference

The charset_converter object.

```
#include <ext_charset.h>
```

Collaboration diagram for t_charset_converter:



34.8.1 Detailed Description

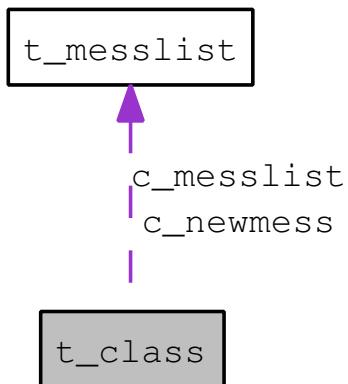
The charset_converter object. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

34.9 t_class Struct Reference

The data structure for a Max class.

```
#include <ext_mess.h>
```

Collaboration diagram for t_class:



Data Fields

- struct symbol * [c_sym](#)
symbol giving name of class
- struct symbol * [c_filename](#)
name of file associated with this class

34.9.1 Detailed Description

The data structure for a Max class. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

34.10 t_datetime Struct Reference

The Systime data structure.

```
#include <ext_systime.h>
```

Data Fields

- unsigned long **year**
year
- unsigned long **month**
month
- unsigned long **day**
day
- unsigned long **hour**
hour
- unsigned long **minute**
minute
- unsigned long **second**
second
- unsigned long **millisecond**
(reserved for future use)

34.10.1 Detailed Description

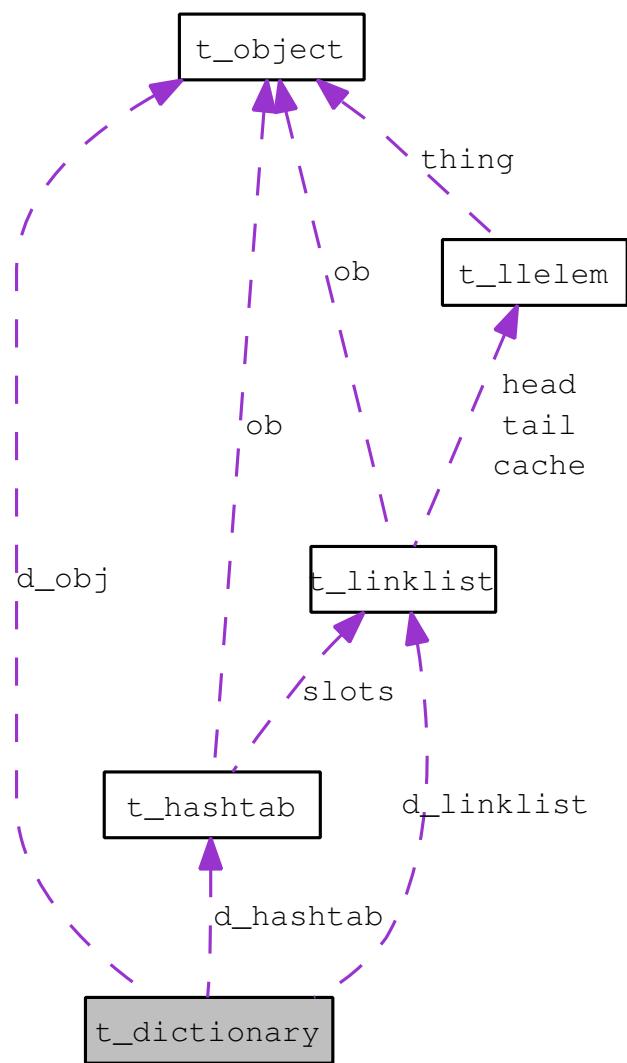
The Systime data structure.

34.11 t_dictionary Struct Reference

The dictionary object.

```
#include <ext_dictionary.h>
```

Collaboration diagram for t_dictionary:



34.11.1 Detailed Description

The dictionary object. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

See also

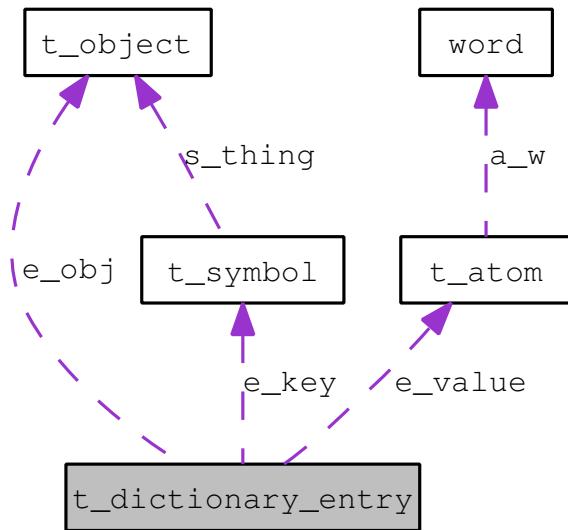
[t_dictionary](#)

34.12 t_dictionary_entry Struct Reference

A dictionary entry.

```
#include <ext_dictionary.h>
```

Collaboration diagram for t_dictionary_entry:



34.12.1 Detailed Description

A dictionary entry. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

See also

[t_dictionary](#)

34.13 t_expr Struct Reference

Struct for an instance of expr.

```
#include <ext_expr.h>
```

Data Fields

- struct ex_ex [exp_res](#)
the result of last evaluation

34.13.1 Detailed Description

Struct for an instance of expr.

34.14 t_fileinfo Struct Reference

Information about a file.

```
#include <ext_path.h>
```

Data Fields

- long **type**
type (four-char-code)
- long **creator**
Mac-only creator (four-char-code).
- long **date**
date
- long **flags**
One of the values defined in [e_max_fileinfo_flags](#).

34.14.1 Detailed Description

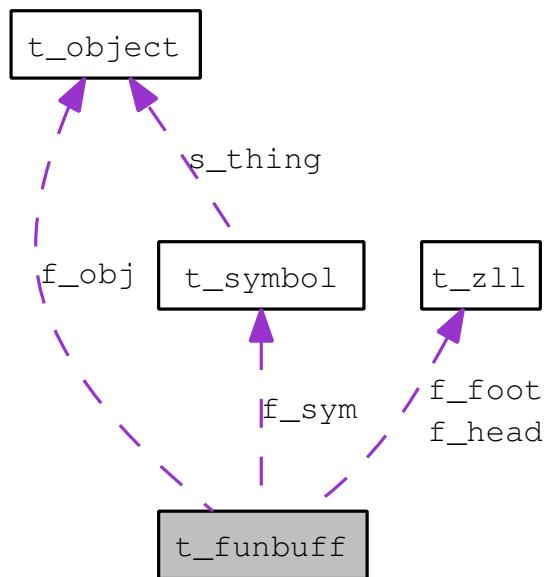
Information about a file.

34.15 t_funbuff Struct Reference

The structure of a funbuff object.

```
#include <ext_maxtypes.h>
```

Collaboration diagram for t_funbuff:



Data Fields

- `t_zll f_head`
head of double linked list of function elements
- `t_zll * f_foot`
foot in the door pointer for list
- long `f_gotoDelta`
used by goto and next
- long `f_selectX`
selected region start
- long `f_selectW`
selected region width
- `t_symbol * f_sym`
filename
- long `f_y`
y-value from inlet

- char [f_yvalid](#)
flag that y has been set since x has
- char [f_embed](#)
flag for embedding funbuff values in patcher

34.15.1 Detailed Description

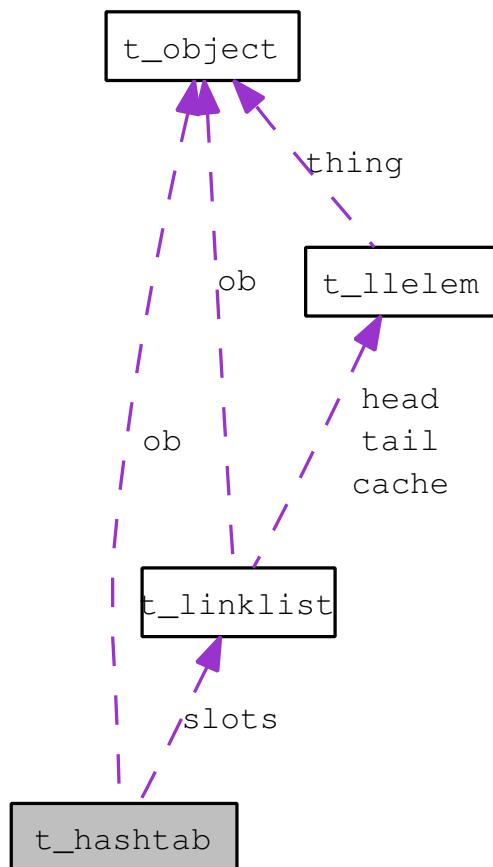
The structure of a funbuff object.

34.16 t_hashtab Struct Reference

The hashtab object.

```
#include <ext_hashtab.h>
```

Collaboration diagram for t_hashtab:



34.16.1 Detailed Description

The hashtab object. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

See also

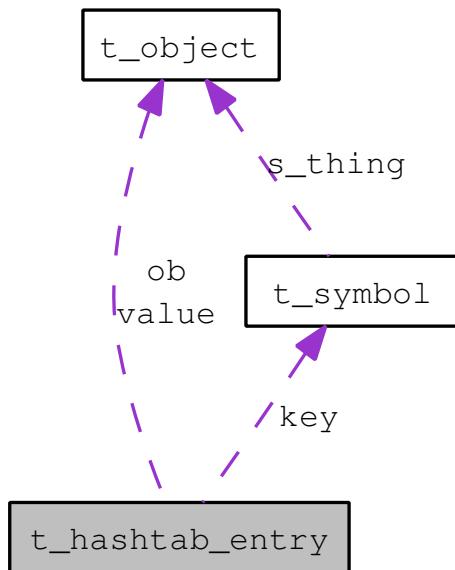
[t_hashtab](#)

34.17 t_hashtab_entry Struct Reference

A hashtab entry.

```
#include <ext_hashtab.h>
```

Collaboration diagram for t_hashtab_entry:



34.17.1 Detailed Description

A hashtab entry. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

See also

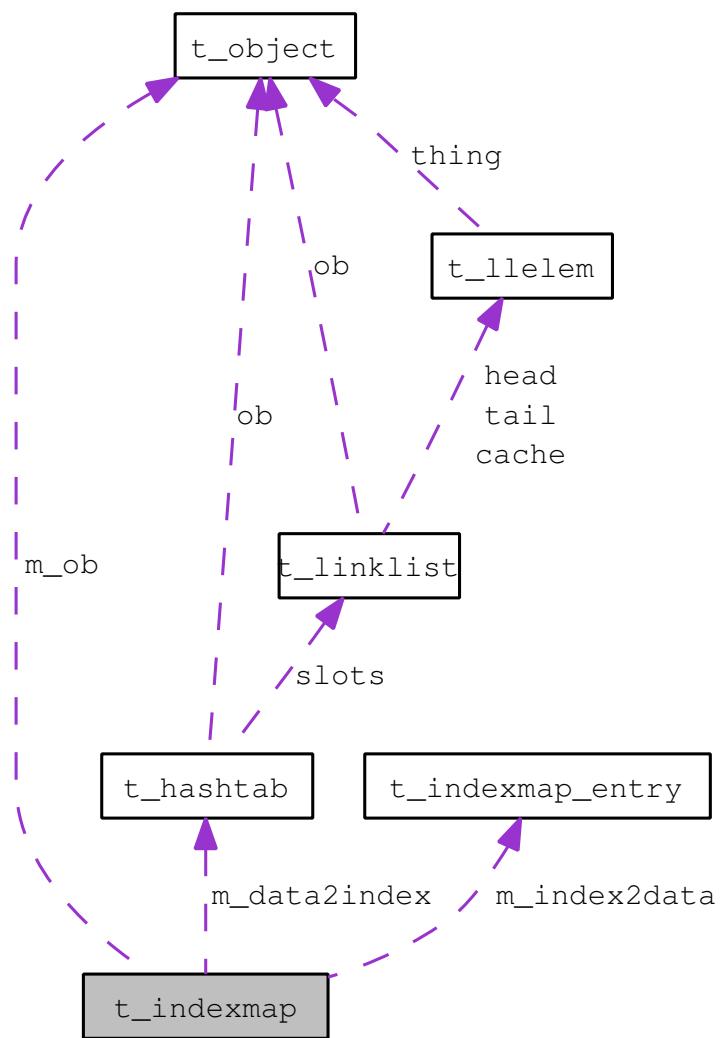
[t_hashtab](#)

34.18 t_indexmap Struct Reference

An indexmap object.

```
#include <indexmap.h>
```

Collaboration diagram for t_indexmap:



34.18.1 Detailed Description

An indexmap object. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

See also

[t_indexmap_entry](#)

34.19 t_indexmap_entry Struct Reference

An indexmap element.

```
#include <indexmap.h>
```

34.19.1 Detailed Description

An indexmap element. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

See also

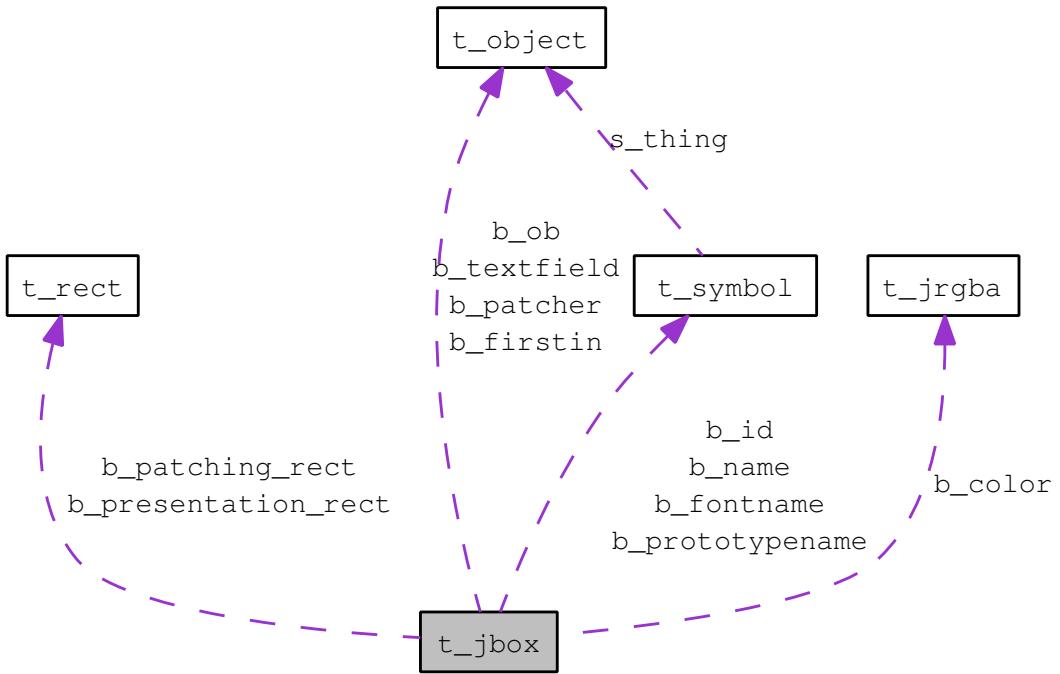
[t_indexmap](#)

34.20 t_jbox Struct Reference

The `t_jbox` struct provides the header for a Max user-interface object.

```
#include <jpatcher_api.h>
```

Collaboration diagram for `t_jbox`:



34.20.1 Detailed Description

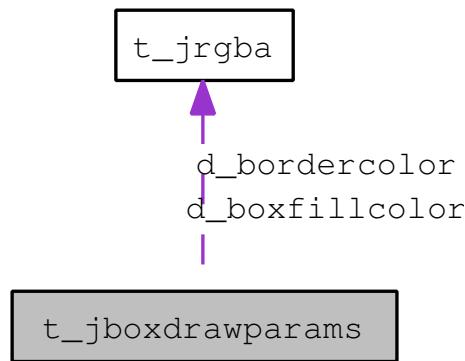
The `t_jbox` struct provides the header for a Max user-interface object. This struct should be considered opaque and is subject to change without notice. Do not access its members directly in any code.

34.21 t_jboxdrawparams Struct Reference

The [t_jboxdrawparams](#) structure.

```
#include <jpatcher_api.h>
```

Collaboration diagram for t_jboxdrawparams:



34.21.1 Detailed Description

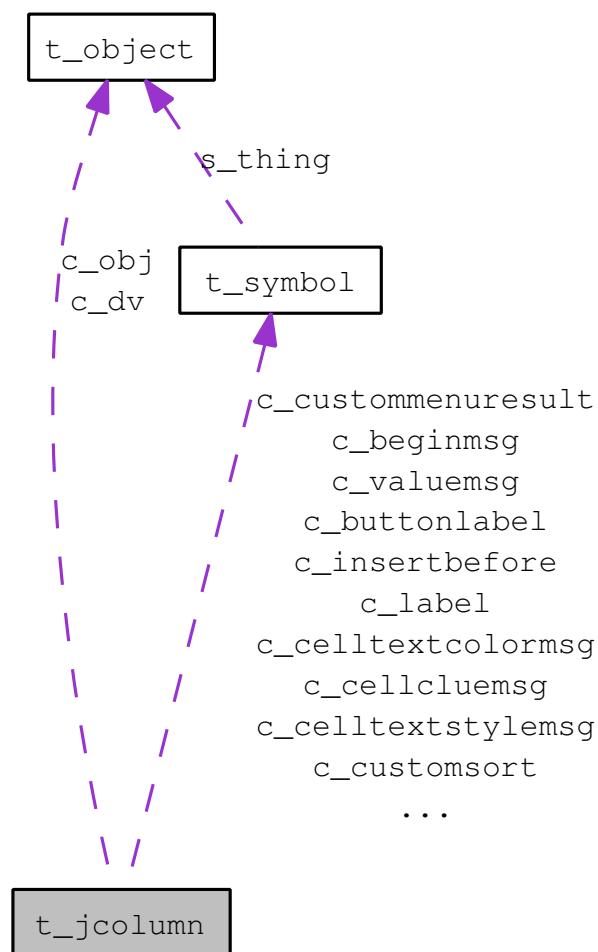
The [t_jboxdrawparams](#) structure. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

34.22 t_jcolumn Struct Reference

A dataview column.

```
#include <jdataview.h>
```

Collaboration diagram for t_jcolumn:



Data Fields

- `t_symbol * c_name`
column name (hash)
- `t_object * c_dv`
parent dataview
- `int c_id`
id in DataViewComponent
- `long c_width`

column width in pixels

- long **c_maxwidth**
max column width
- long **c_minwidth**
min column width
- char **c_autosize**
determine width of text column automatically (true/false)
- char **c_alignment**
display of text, left, right, center
- **t_symbol * c_font**
name of font
- long **c_fontsize**
font size (points?)
- **t_symbol * c_label**
heading of column
- char **c_separator**
separator mode
- char **c_button**
column has a button (true/false)
- **t_symbol * c_buttonlabel**
text in a button
- **t_symbol * c_customsort**
message sent to sort this column -- if none, default sorting is used based on value c_numeric
- char **c_overridesort**
if true only the sortdata method is called, not the sort method (true/false)
- **t_symbol * c_custompaint**
send this msg name to client to paint this column
- **t_symbol * c_valuemsg**
message sent when a component mode cell's value changes
- **t_symbol * c_beginmsg**
message sent when a component mode cell's value is about to start changing
- **t_symbol * c_endmsg**
message sent when a component mode cell's value is finished changing

- **t_symbol * c_rowcomponentmsg**
message sent to determine what kind of component should be created for each cell in a column
- **t_symbol * c_custommenuset**
message to set a menu (for a readonly or custompaint column)
- **t_symbol * c_custommenuresult**
message sent when an item is chosen from a custom menu
- **char c_editable**
can you edit the data in a cell in this column
- **char c_selectable**
can select the data in a cell in this column (possibly without being able to edit)
- **char c_multiselectable**
can you select more than one cell in this column
- **char c_sortable**
can you click on a column heading to sort the data
- **long c_initiallysorted**
if this is set to JCOLUMN_INITIALYSORTED_FORWARD the column is displayed with the sort triangle
- **long c_maxtextlen**
maximum text length: this is used to allocate a buffer to pass to gettext (but there is also a constant)
- **long c_sortdirection**
0 for ascending, 1 for descending
- **long c_component**
enum of components (check box etc.)
- **char c_canselect**
can select entire column
- **char c_cancut**
can cut/clear entire column
- **char c_cancopy**
can copy entire column
- **char c_cancutcells**
can cut a single cell (assumes "editable" or "selectable") (probably won't be implemented)
- **char c_cancopycells**
can copy a single cell
- **char c_canpastecells**
can paste into a single cell

- char **c_hideable**
can the column be hidden
- char **c_hidden**
is the column hidden (set/get)
- char **c_numeric**
is the data numeric (i.e., is getcellvalue implemented)
- char **c_draggable**
can drag the column to rearrange it
- char **c_casesensitive**
use case sensitive sorting (applies only to default text sorting)
- void * **c_reference**
reference for the use of the client
- double **c_indentspacing**
amount of space (in pixels) for one indent level
- **t_symbol * c_insertbefore**
name of column before which this one should have been inserted (used only once)
- **t_symbol * c_cellcluemsg**
message to send requesting clue text for a cell
- **t_symbol * c_celltextcolormsg**
message to get the cell's text color
- **t_symbol * c_celltextstylemsg**
message to get the cell's style and alignment

34.22.1 Detailed Description

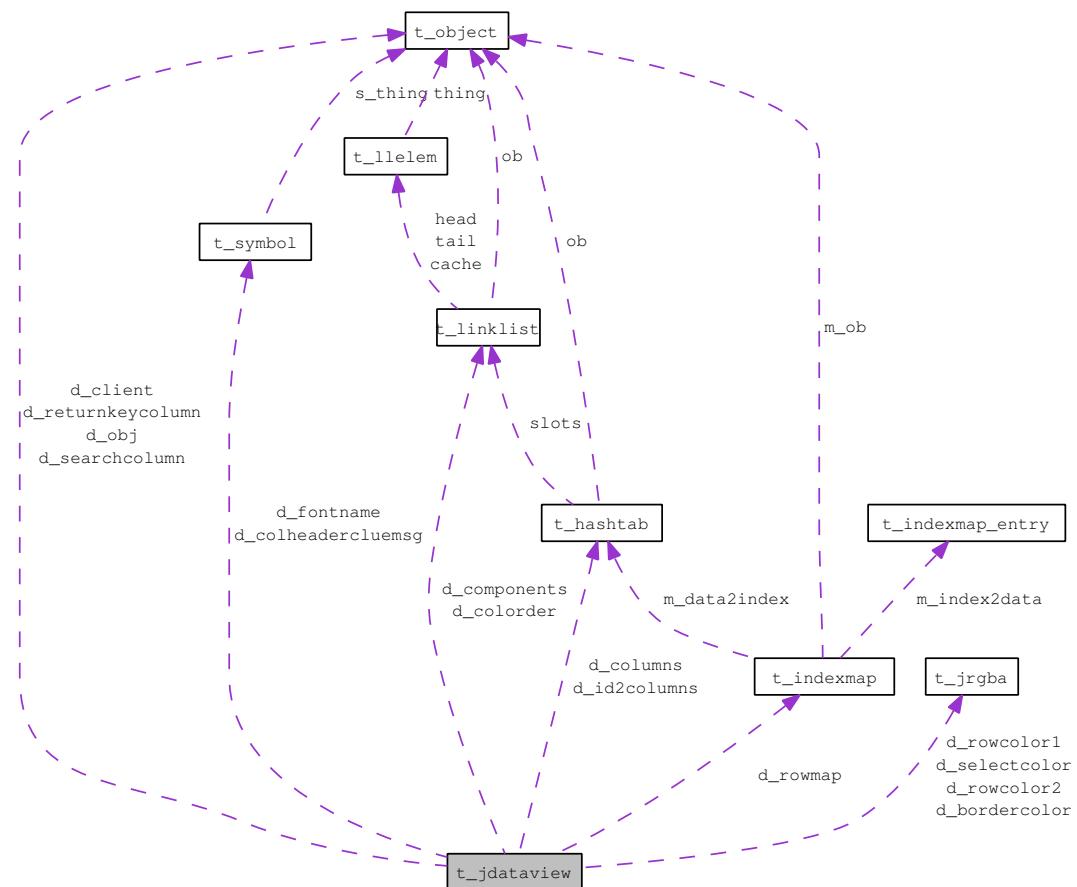
A dataview column. Columns for a given dataview are stored in a **t_hashtab** and accessed by name.

34.23 t_jdataview Struct Reference

The dataview object.

```
#include <jdataview.h>
```

Collaboration diagram for t_jdataview:



Data Fields

- `t_lllelem * d_components`
list of DataViewComponents showing this dataview
- `t_object * d_client`
object that will be sent messages to get data to display
- `t_hashtab * d_columns`
columns -- point to t_jcolumn objects
- `t_hashtab * d_id2columns`
columns from column IDs
- `t_linklist * d_colororder`

current order of columns

- **t_indexmap * d_rowmap**
collection of rows (including number of rows)
- long **d_numcols**
number of columns
- double **d_rowheight**
fixed height of a row in pixels
- char **d_autoheight**
height determined by font
- char **d_hierarchical**
does it allow hierarchical disclosure (true / false) -- not implemented yet
- **t_jrgba d_rowcolor1**
odd row color (striped)
- **t_jrgba d_rowcolor2**
even row color
- **t_jrgba d_selectcolor**
color when rows are selected
- **t_jrgba d_bordercolor**
border color
- char **d_bordercolorset**
was border color set? if not, use JUCE default
- char **d_canselectmultiple**
multiple rows are selectable
- char **d_cancopy**
copy enabled
- char **d_cancut**
cut / clear enabled
- char **d_canpaste**
paste enabled
- char **d_canrearrangerows**
rows can be dragged to rearrange -- may not be implemented yet
- char **d_canrearrangecolumns**
columns can be dragged to rearrange

- long **d_viscount**
number of visible views of this dataview
- long **d_inset**
inset for table inside containing component in pixels
- char **d_autosizeright**
right side autosizes when top-level component changes
- char **d_autosizebottom**
bottom autosizes when top-level component changes
- char **d_dragenabled**
enabled for dragging (as in drag and drop)
- **t_symbol * d_fontname**
font name
- double **d_fontsize**
font size
- **t_symbol * d_colheadercluemsg**
message to send requesting clue text for the column headers
- char **d_autosizerightcolumn**
right column should stretch to remaining width of the dataview, regardless of column width
- char **d_customselectcolor**
send getcellcolor message to draw selected cell, don't use select color
- void * **d_qelem**
defer updating
- long **d_top_inset**
vertical inset for row background (default 0)
- long **d_bottom_inset**
vertical inset for row background (default 0)
- long **d_borderthickness**
border line thickness default 0 for no border
- char **d_keyfocusable**
notify component to grab some keys
- char **d_enabledeletekey**
delete key will delete selected rows
- char **d_usegradient**
color rows with gradient between rowcolor1 (top) and rowcolor2 (bottom)

- char **d_inchange**
in change flag for inspector end-change protection system
- char **d_horizscrollvisible**
is horizontal scroll bar visible
- char **d_vertscrollvisible**
is vertical scroll bar visible
- char **d_scrollvisset**
has the scroll visibility ever been changed since the dv was created?
- char **d_overridedefocus**
override default focus behavior where ListBox is focused when assigning focus to the dataview
- char **d_usesystemfont**
use system font (true by default)
- **t_object * d_searchcolumn**
column we ask for celltext in order to navigate the selection via the keyboard
- **t_object * d_returnkeycolumn**
column that is sent the return key when a given row is selected
- void * **d_navcache**
sorted list of column strings for key navigation
- char **d_usecharheight**
use font specified in points rather than pixels (default is pixels)

34.23.1 Detailed Description

The dataview object.

34.24 t_jgraphics_font_extents Struct Reference

A structure for holding information related to how much space the rendering of a given font will use.

```
#include <jgraphics.h>
```

Data Fields

- double **ascent**
The ascent.
- double **descent**
The descent.
- double **height**
The hieght.
- double **max_x_advance**
Unused / Not valid.
- double **max_y_advance**
Unused / Not valid.

34.24.1 Detailed Description

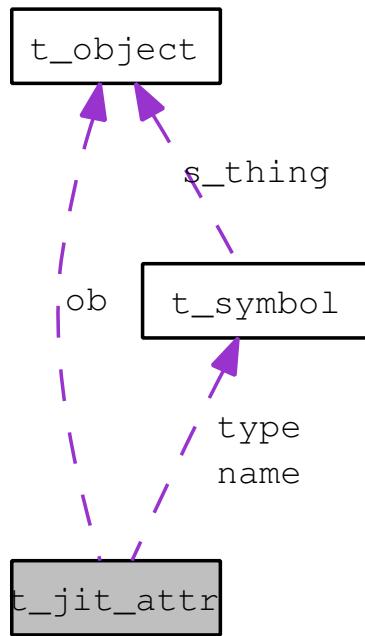
A structure for holding information related to how much space the rendering of a given font will use. The units for these measurements is in pixels.

34.25 t_jit_attr Struct Reference

Common attribute struct.

```
#include <jit.common.h>
```

Collaboration diagram for t_jit_attr:



Data Fields

- `t_jit_object ob`
common object header
- `t_symbol * name`
attribute name
- `t_symbol * type`
attribute type (char, long, float32, float64, symbol, atom, or obj)
- `long flags`
flags for public/private get/set methods
- `method get`
override default get method
- `method set`
override default set method
- `void * filterget`

filterobject for get method

- void * **filterset**

filterobject for set method

- void * **reserved**

for future use

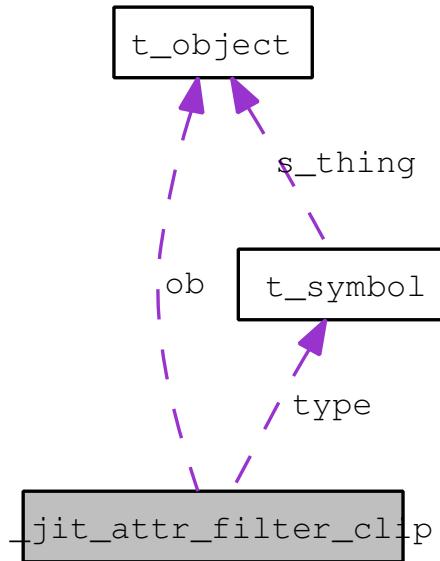
34.25.1 Detailed Description

Common attribute struct. Shared by all built in attribute classes.

34.26 t_jit_attr_filter_clip Struct Reference

[t_jit_attr_filter_clip](#) object struct.

Collaboration diagram for t_jit_attr_filter_clip:



Data Fields

- [t_jit_object](#) `ob`
common object header
- [t_symbol](#) * `type`
"type" attribute
- double `scale`
scaling factor; "scale" attribute
- double `min`
minimum value; "min" attribute
- double `max`
maximum value; "max" attribute
- char `usescale`
use scaling flag; "usescale" attribute
- char `usemin`
clip to minimum flag; "usemin" attribute
- char `usemax`
clip to maximum flag; "usemax" attribute

34.26.1 Detailed Description

[t_jit_attr_filter_clip](#) object struct.

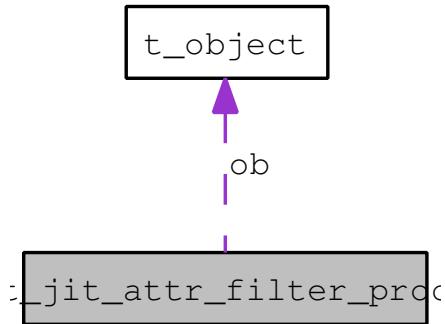
Warning

This struct should not be accessed directly, but is provided for reference when calling Jitter attribute functions.

34.27 t_jit_attr_filter_proc Struct Reference

[t_jit_attr_filter_proc](#) object struct.

Collaboration diagram for t_jit_attr_filter_proc:



Data Fields

- [t_jit_object ob](#)
common object header
- [method proc](#)
filter procedure

34.27.1 Detailed Description

[t_jit_attr_filter_proc](#) object struct.

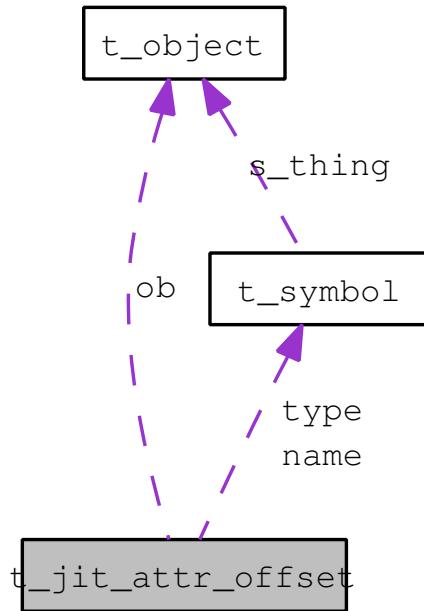
Warning

This struct should not be accessed directly, but is provided for reference when calling Jitter attribute functions.

34.28 t_jit_attr_offset Struct Reference

[t_jit_attr_offset](#) object struct.

Collaboration diagram for t_jit_attr_offset:



Data Fields

- `t_jit_object ob`
common object header
- `t_symbol * name`
attribute name
- `t_symbol * type`
attribute type (char, long, float32, float64, symbol, atom, or obj)
- `long flags`
flags for public/private get/set methods
- `method get`
override default get method
- `method set`
override default set method
- `void * filterget`
filterobject for get method
- `void * filterset`

filterobject for set method

- void * **reserved**
for future use
- long **offset**
byte offset to the attribute data

34.28.1 Detailed Description

[**t_jit_attr_offset**](#) object struct.

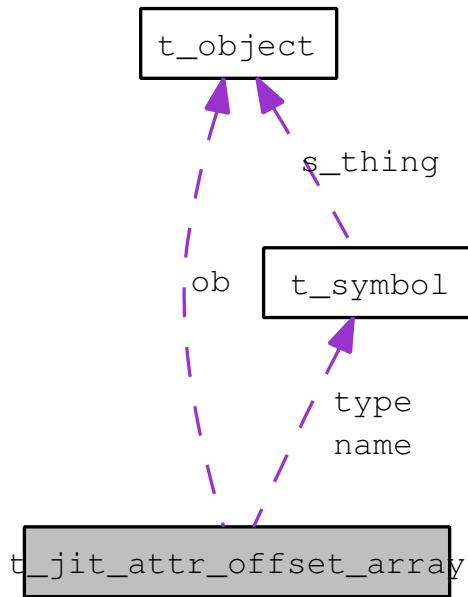
Warning

This struct should not be accessed directly, but is provided for reference. Attribute objects do not typically use attributes themselves to access members, but rather accessor methods--i.e. use `jit_object_-method` in place of the `jit_attr_*` functions to access attribute state.

34.29 t_jit_attr_offset_array Struct Reference

[t_jit_attr_offset_array](#) object struct.

Collaboration diagram for t_jit_attr_offset_array:



Data Fields

- `t_jit_object ob`
common object header
- `t_symbol * name`
attribute name
- `t_symbol * type`
attribute type (char, long, float32, float64, symbol, atom, or obj)
- `long flags`
flags for public/private get/set methods
- `method get`
override default get method
- `method set`
override default set method
- `void * filterget`
filterobject for get method
- `void * filterset`

filterobject for set method

- void * **reserved**
for future use
- long **offset**
byte offset to the attribute data
- long **size**
maximum size
- long **offsetcount**
byte offset to the attribute count

34.29.1 Detailed Description

[`t_jit_attr_offset_array`](#) object struct.

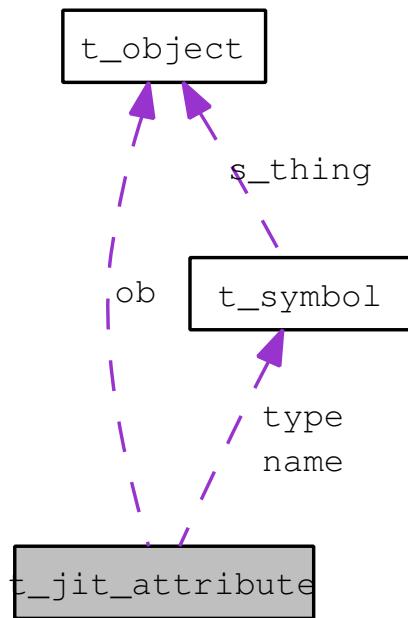
Warning

This struct should not be accessed directly, but is provided for reference. Attribute objects do not typically use attributes themselves to access members, but rather accessor methods--i.e. use `jit_object_-method` in place of the `jit_attr_*` functions to access attribute state.

34.30 t_jit_attribute Struct Reference

[t_jit_attribute](#) object struct.

Collaboration diagram for t_jit_attribute:



Data Fields

- `t_jit_object ob`
common object header
- `t_symbol * name`
attribute name
- `t_symbol * type`
attribute type (char, long, float32, float64, symbol, atom, or obj)
- `long flags`
flags for public/private get/set methods
- `method get`
override default get method
- `method set`
override default set method
- `void * filterget`
filterobject for get method

- void * **filterset**
filterobject for set method
- void * **reserved**
for future use
- void * **data**
internally stored data
- long **size**
data size

34.30.1 Detailed Description

[t_jit_attribute](#) object struct.

Warning

This struct should not be accessed directly, but is provided for reference. Attribute objects do not typically use attributes themselves to access members, but rather accessor methods--i.e. use `jit_object_-method` in place of the `jit_attr_*` functions to access attribute state.

34.31 t_jit_gl_drawinfo Struct Reference

[t_jit_gl_drawinfo](#) struct used for tasks such as multi texture unit binding.

```
#include <jit/gl/drawinfo.h>
```

Data Fields

- `t_jit_gl_context ctx`
current t_jit_gl_context
- `void * ob3d`
object's t_jit_ob3d pointer
- `void * rfu [6]`
reserved for future use

34.31.1 Detailed Description

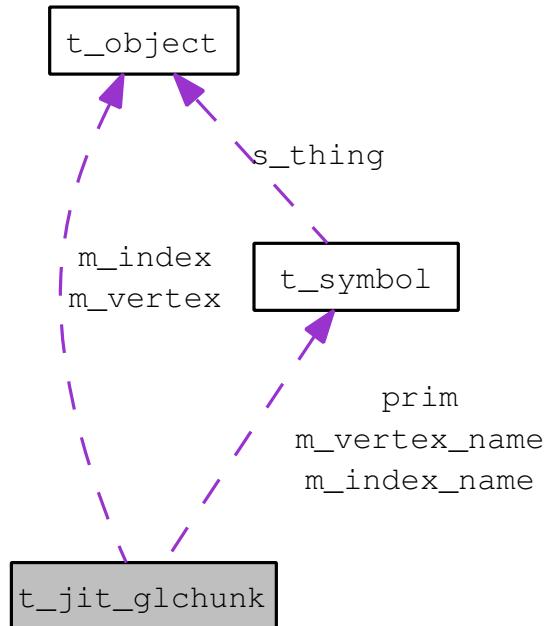
[t_jit_gl_drawinfo](#) struct used for tasks such as multi texture unit binding.

34.32 t_jit_glchunk Struct Reference

`t_jit_glchunk` is a public structure to store one gl-command's-worth of data, in a format which can be passed easily to `glDrawRangeElements`, and `matrixoutput`.

```
#include <jit.gl.chunk.h>
```

Collaboration diagram for `t_jit_glchunk`:



Data Fields

- `t_symbol * prim`
drawing primitive. "tri_strip", "tri", "quads", "quad_grid", etc.
- `t_jit_object * m_vertex`
vertex matrix containing xyzst... data
- `t_symbol * m_vertex_name`
vertex matrix name
- `t_jit_object * m_index`
optional 1d matrix of vertex indices to use with drawing primitive
- `t_symbol * m_index_name`
index matrix name
- `unsigned long m_flags`
chunk flags to ignore texture, normal, color, or edge planes when drawing
- `void * next_chunk`

pointer to next chunk for drawing a list of chunks together

34.32.1 Detailed Description

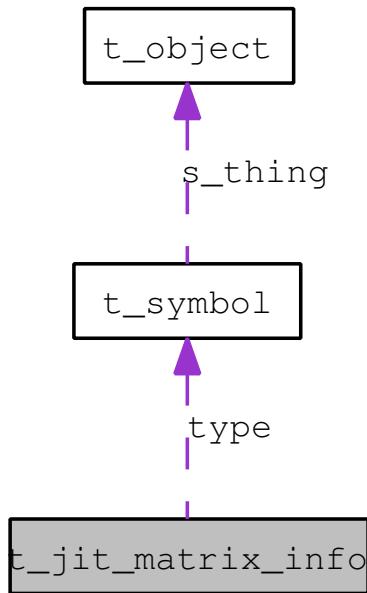
`t_jit_glchunk` is a public structure to store one gl-command's-worth of data, in a format which can be passed easily to `glDrawRangeElements`, and `matrixoutput`.

34.33 t_jit_matrix_info Struct Reference

Matrix information struct.

```
#include <jit.common.h>
```

Collaboration diagram for t_jit_matrix_info:



Data Fields

- long **size**
in bytes (0xFFFFFFFF=UNKNOWN)
- **t_symbol * type**
primitive type (char, long, float32, or float64)
- long **flags**
flags to specify data reference, handle, or tightly packed
- long **dimcount**
number of dimensions
- long **dim** [JIT_MATRIX_MAX_DIMCOUNT]
dimension sizes
- long **dimstride** [JIT_MATRIX_MAX_DIMCOUNT]
stride across dimensions in bytes
- long **planecount**
number of planes

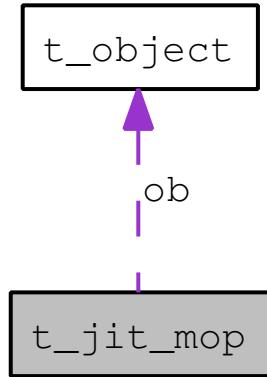
34.33.1 Detailed Description

Matrix information struct. Used to get/set multiple matrix attributes at once.

34.34 t_jit_mop Struct Reference

[t_jit_mop](#) object struct.

Collaboration diagram for t_jit_mop:



Data Fields

- `t_jit_object ob`
standard object header
- `void * special`
special data pointer for use by wrappers of various kinds (e.g. max wrapper)
- `long inputcount`
"inputcount" attribute
- `long outputcount`
"outputcount" attribute
- `t_jit_linklist * inputlist`
linked list of inputs, accessed via methods
- `t_jit_linklist * outputlist`
linked list of outputs, accessed via methods
- `char caninplace`
deprecated
- `char adapt`
"adapt" attribute
- `char outputmode`
"outputmode" attribute

34.34.1 Detailed Description

[t_jit_mop](#) object struct.

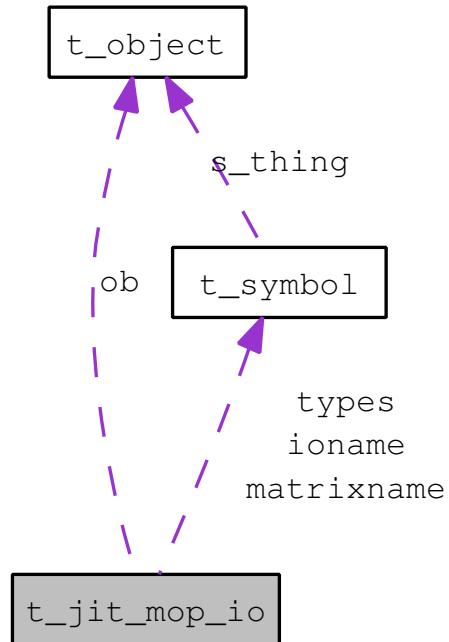
Warning

This struct should not be accessed directly, but is provided for reference when calling Jitter attribute functions.

34.35 t_jit_mop_io Struct Reference

[t_jit_mop_io](#) object struct.

Collaboration diagram for t_jit_mop_io:



Data Fields

- `t_jit_object ob`
standard object header
- `void * special`
special data pointer for use by wrappers of various kinds (e.g. max wrapper)
- `t_symbol * ioname`
"ioname" attribute
- `t_symbol * matrixname`
"matrixname" attribute
- `void * matrix`
internal matrix, accessed via methods (unused in class template MOP)
- `t_symbol * types [JIT_MATRIX_MAX_TYPES]`
"types" attribute
- `long mindim [JIT_MATRIX_MAX_DIMCOUNT]`
"mindim" attribute

- long [maxdim](#) [JIT_MATRIX_MAX_DIMCOUNT]
"maxdim" attribute
- long [typescount](#)
relevant to "types" attribute
- long [mindimcount](#)
"mindimcount" attribute
- long [maxdimcount](#)
"maxdimcount" attribute
- long [minplanecount](#)
"minplanecount" attribute
- long [maxplanecount](#)
"maxplanecount" attribute
- char [typelink](#)
"typelink" attribute
- char [dimlink](#)
"dimlink" attribute
- char [planelink](#)
"planelink" attribute
- method [ioproc](#)
I/O procedure, accessed via methods.

34.35.1 Detailed Description

[t_jit_mop_io](#) object struct.

Warning

This struct should not be accessed directly, but is provided for reference when calling Jitter attribute functions.

34.36 t_jit_op_info Struct Reference

Provides base pointer and stride for vector operator functions.

```
#include <jit.op.h>
```

Data Fields

- `void * p`
base pointer (coerced to appropriate type)
- `long stride`
stride between elements (in type, not bytes)

34.36.1 Detailed Description

Provides base pointer and stride for vector operator functions.

34.37 t_jmatrix Struct Reference

An affine transformation (such as scale, shear, etc).

```
#include <jgraphics.h>
```

Data Fields

- double **xx**
xx component
- double **yx**
yx component
- double **xy**
xy component
- double **yy**
yy component
- double **x0**
x translation
- double **y0**
y translation

34.37.1 Detailed Description

An affine transformation (such as scale, shear, etc).

34.38 t_jrgb Struct Reference

A color composed of red, green, and blue components.

```
#include <jpatcher_api.h>
```

Data Fields

- double [red](#)
Red component in the range [0.0, 1.0].
- double [green](#)
Green component in the range [0.0, 1.0].
- double [blue](#)
Blue component in the range [0.0, 1.0].

34.38.1 Detailed Description

A color composed of red, green, and blue components. Typically such a color is assumed to be completely opaque (with no transparency).

See also

[t_jrgba](#)

34.39 t_jrgba Struct Reference

A color composed of red, green, blue, and alpha components.

```
#include <jpatcher_api.h>
```

Data Fields

- double `red`
Red component in the range [0.0, 1.0].
- double `green`
Green component in the range [0.0, 1.0].
- double `alpha`
Alpha (transparency) component in the range [0.0, 1.0].

34.39.1 Detailed Description

A color composed of red, green, blue, and alpha components.

34.40 t_line_3d Struct Reference

Line or line segment in 3D space (GLfloat).

```
#include <jit/gl.h>
```

Data Fields

- GLfloat **u** [3]
starting point
- GLfloat **v** [3]
ending point

34.40.1 Detailed Description

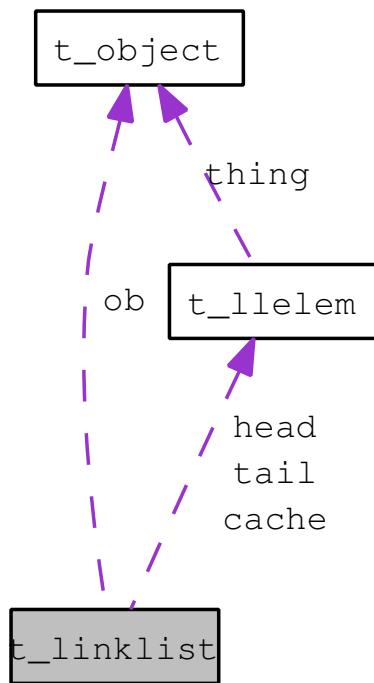
Line or line segment in 3D space (GLfloat).

34.41 t_linklist Struct Reference

The linklist object.

```
#include <ext_linklist.h>
```

Collaboration diagram for t_linklist:



34.41.1 Detailed Description

The linklist object. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

See also

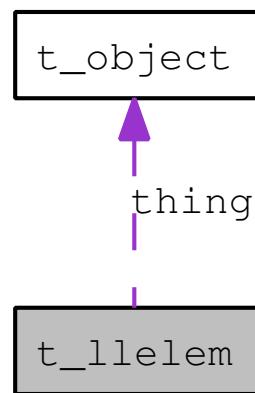
[t llelem](#)

34.42 t_ll elem Struct Reference

A linklist element.

```
#include <ext_linklist.h>
```

Collaboration diagram for t_ll elem:



34.42.1 Detailed Description

A linklist element. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

See also

[t_linklist](#)

34.43 t_matrix_conv_info Struct Reference

Matrix conversion struct.

```
#include <jit.common.h>
```

Data Fields

- long **flags**
flags for whether or not to use interpolation, or source/destination dimensions
- long **planemap** [JIT_MATRIX_MAX_PLANECOUNT]
plane mapping
- long **srcdimstart** [JIT_MATRIX_MAX_DIMCOUNT]
source dimension start
- long **srcdimend** [JIT_MATRIX_MAX_DIMCOUNT]
source dimension end
- long **dstdimstart** [JIT_MATRIX_MAX_DIMCOUNT]
destination dimension start
- long **dstdimend** [JIT_MATRIX_MAX_DIMCOUNT]
destination dimension end

34.43.1 Detailed Description

Matrix conversion struct. Used to copy data from one matrix to another with special characteristics.

34.44 t_messlist Struct Reference

A list of symbols and their corresponding methods, complete with typechecking information.

```
#include <ext_mess.h>
```

Data Fields

- struct symbol * [m_sym](#)
Name of the message.
- method [m_fun](#)
Method associated with the message.
- char [m_type](#) [MAXARG+1]
Argument type information.

34.44.1 Detailed Description

A list of symbols and their corresponding methods, complete with typechecking information.

34.45 t_object Struct Reference

The structure for the head of any object which wants to have inlets or outlets, or support attributes.

```
#include <ext_mess.h>
```

Data Fields

- struct messlist * [o_messlist](#)
list of messages and methods. The -I entry of the message list of an object contains a pointer to its [t_class](#) entry.
- struct inlet * [o_inlet](#)
list of inlets
- struct outlet * [o_outlet](#)
list of outlets

34.45.1 Detailed Description

The structure for the head of any object which wants to have inlets or outlets, or support attributes.

34.46 t_path Struct Reference

The path data structure.

```
#include <ext_path.h>
```

34.46.1 Detailed Description

The path data structure. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

34.47 t_pathlink Struct Reference

The pathlink data structure.

```
#include <ext_path.h>
```

34.47.1 Detailed Description

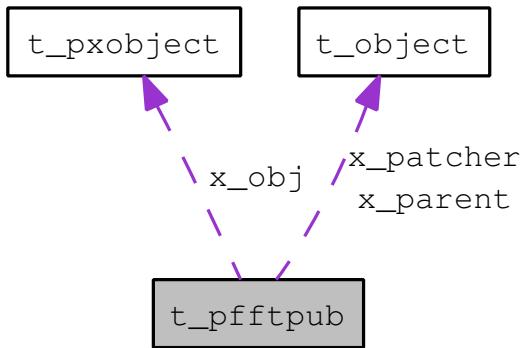
The pathlink data structure. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

34.48 t_pfftpub Struct Reference

Public FFT Patcher struct.

```
#include <r_pfft.h>
```

Collaboration diagram for t_pfftpub:



Data Fields

- `t_object * x_parent`
parent patcher
- `t_object * x_patch`
patcher loaded
- `struct _dspchain * x_chain`
dsp chain within pfft
- `long x_fftsize`
fft frame size
- `long x_ffthop`
hop between fft frames
- `long x_fftoffset`
n samples offset before fft is started
- `long x_fftindex`
current index into fft frame
- `short x_fullspect`
process half-spectrum (0) or full mirrored spectrum (1)?

34.48.1 Detailed Description

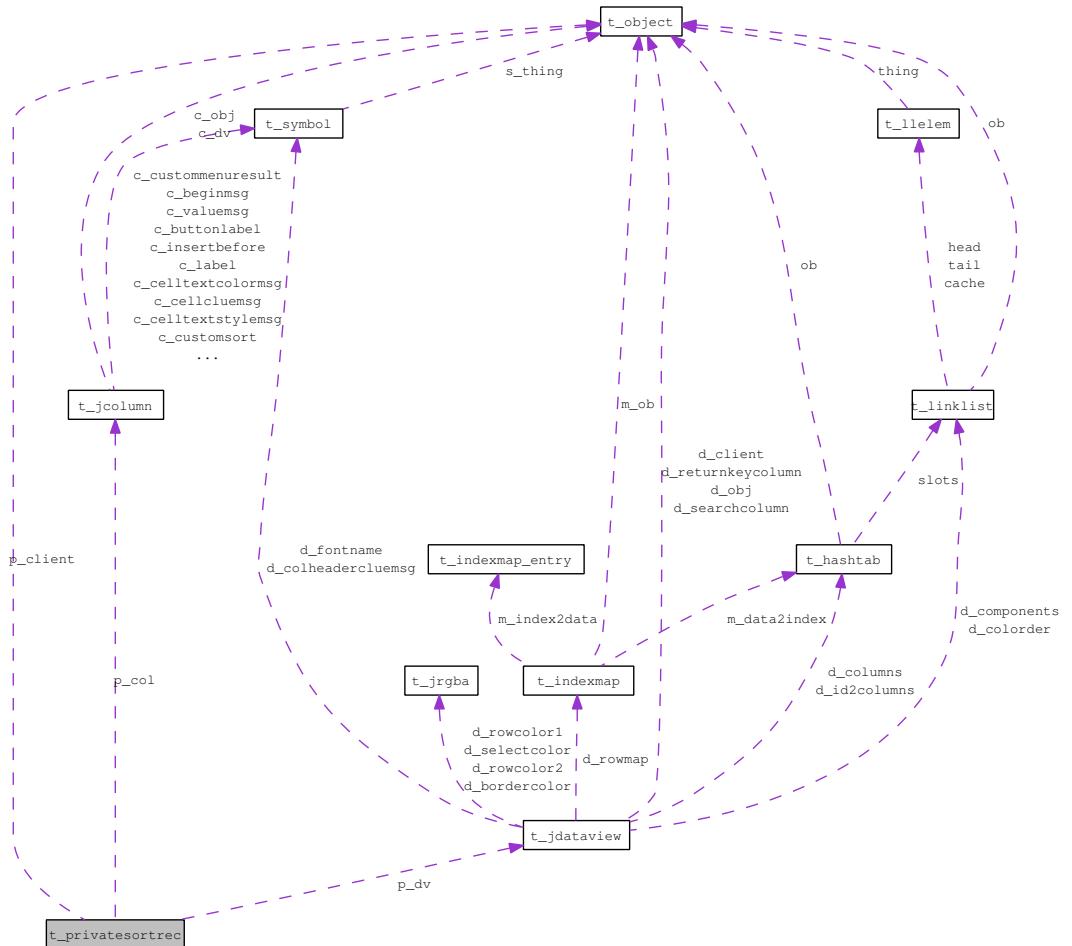
Public FFT Patcher struct.

34.49 t_privatesortrec Struct Reference

used to pass data to a client sort function

```
#include <jdataview.h>
```

Collaboration diagram for t_privatesortrec:



Data Fields

- `t_jcolumn * p_col`
column object to sort
- `char p_fwd`
I if sorting "forwards"
- `t_object * p_client`
pointer to the client object
- `t_jdataview * p_dx`
pointer to the dataview

34.49.1 Detailed Description

used to pass data to a client sort function

34.50 t_pt Struct Reference

Coordinates for specifying a point.

```
#include <jpatcher_api.h>
```

Data Fields

- double [x](#)
The horizontal coordinate.
- double [y](#)
The vertical coordinate.

34.50.1 Detailed Description

Coordinates for specifying a point.

See also

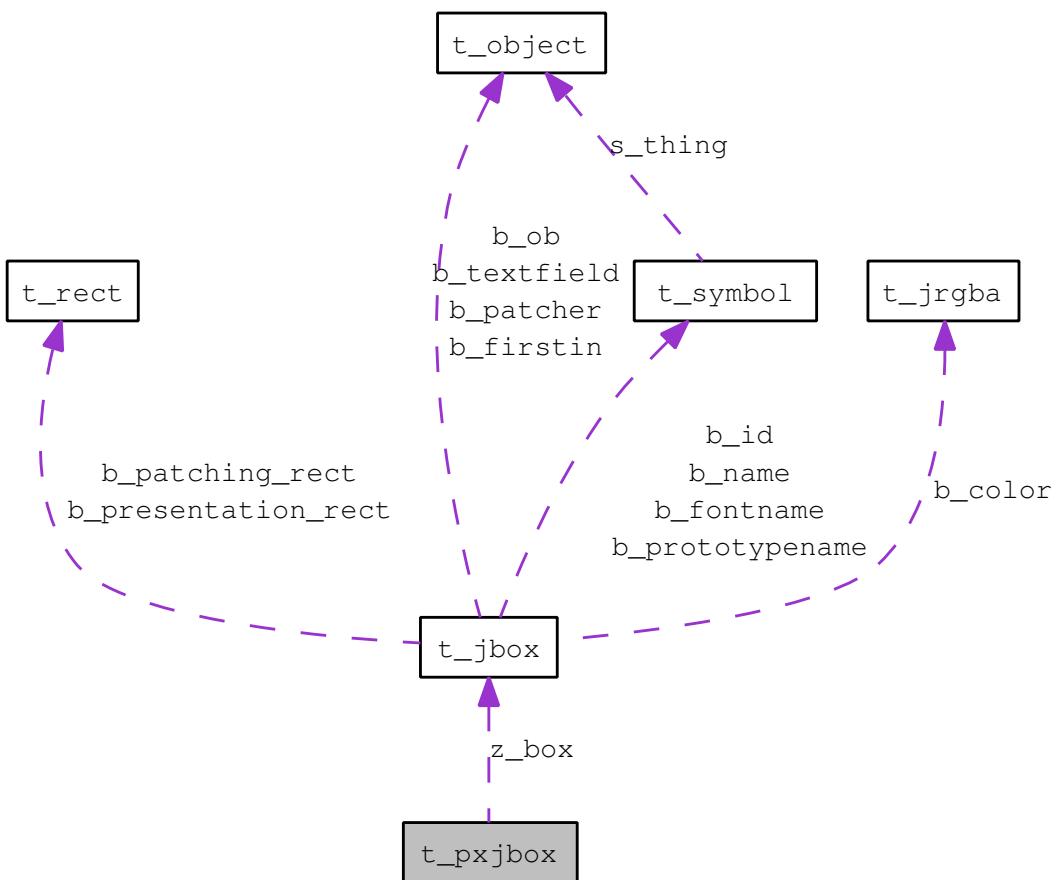
[t_rect](#)
[t_size](#)

34.51 t_pxjbox Struct Reference

Header for any ui signal processing object.

```
#include <z_dsp.h>
```

Collaboration diagram for t_pxjbox:



Data Fields

- **t_jbox z_box**
The box struct used by all ui objects.
 - long **z_disabled**
set to non-zero if this object is muted (using the pcontrol or mute~ objects)
 - short **z_count**
an array that indicates what inlets/outlets are connected with signals
 - short **z_misc**
flags (bitmask) determining object behaviour, such as Z_NO_INPLACE, Z_PUT_FIRST, or Z_PUT_LAST

34.51.1 Detailed Description

Header for any ui signal processing object. For non-ui objects use [t_pxobject](#).

34.52 t_pxobject Struct Reference

Header for any non-ui signal processing object.

```
#include <z_dsp.h>
```

Data Fields

- struct object [z_ob](#)
The standard [t_object](#) struct.
- long [z_disabled](#)
set to non-zero if this object is muted (using the pcontrol or mute~ objects)
- short [z_count](#)
an array that indicates what inlets/outlets are connected with signals
- short [z_misc](#)
flags (bitmask) determining object behaviour, such as [Z_NO_INPLACE](#), [Z_PUT_FIRST](#), or [Z_PUT_LAST](#)

34.52.1 Detailed Description

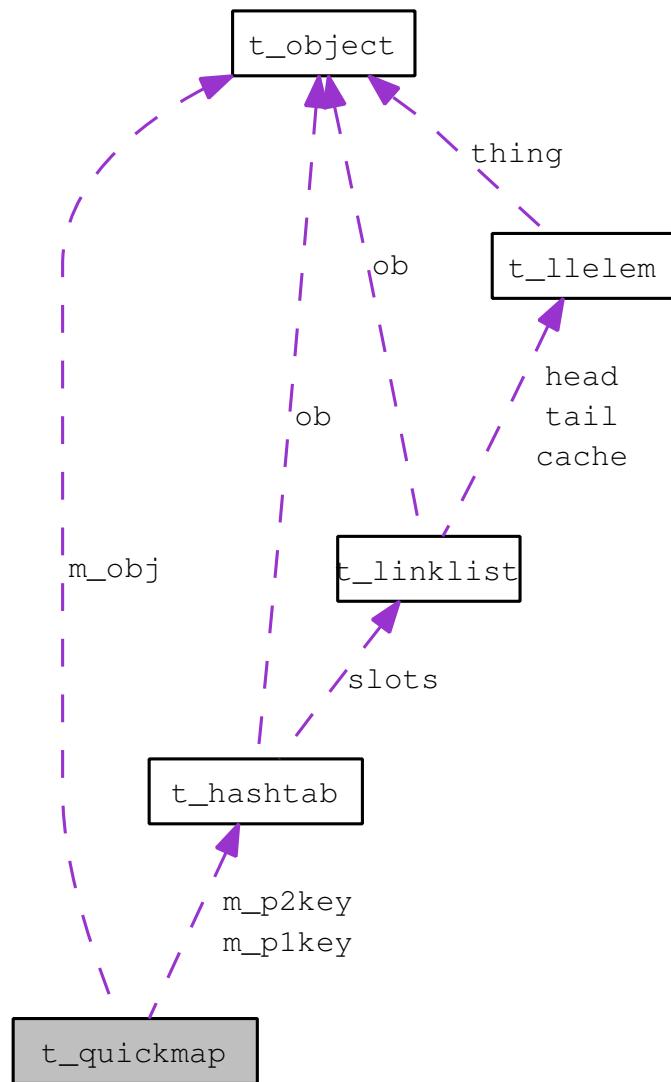
Header for any non-ui signal processing object. For ui objects use [t_pxjbox](#).

34.53 t_quickmap Struct Reference

The quickmap object.

```
#include <ext_quickmap.h>
```

Collaboration diagram for t_quickmap:



34.53.1 Detailed Description

The quickmap object. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

34.54 t_rect Struct Reference

Coordinates for specifying a rectangular region.

```
#include <jpatcher_api.h>
```

Data Fields

- double [x](#)
The horizontal origin.
- double [y](#)
The vertical origin.
- double [width](#)
The width.
- double [height](#)
The height.

34.54.1 Detailed Description

Coordinates for specifying a rectangular region.

See also

[t_pt](#)
[t_size](#)

34.55 t_signal Struct Reference

The signal data structure.

```
#include <z_dsp.h>
```

Data Fields

- long [s_n](#)
The vector size of the signal.
- [t_sample * s_vec](#)
An array of buffers holding the vectors of audio.
- float [s_sr](#)
The sample rate of the signal.

34.55.1 Detailed Description

The signal data structure.

34.56 t_size Struct Reference

Coordinates for specifying the size of a region.

```
#include <jpatcher_api.h>
```

Data Fields

- double [width](#)

The width.

- double [height](#)

The height.

34.56.1 Detailed Description

Coordinates for specifying the size of a region.

See also

[t_rect](#)

[t_pt](#)

34.57 t_stack_splat Struct Reference

for passing on the stack in method calls (no need for struct packing here, since flat array)

```
#include <jit.common.h>
```

Data Fields

- char **b** [64]
byte array to push onto stack

34.57.1 Detailed Description

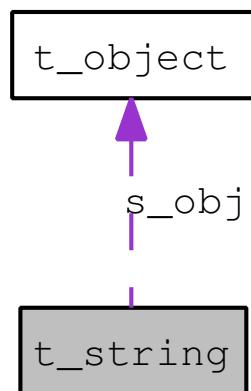
for passing on the stack in method calls (no need for struct packing here, since flat array)

34.58 t_string Struct Reference

The string object.

```
#include <ext_obstring.h>
```

Collaboration diagram for t_string:



34.58.1 Detailed Description

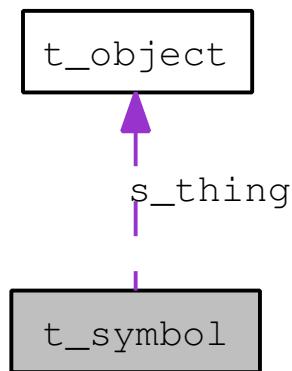
The string object. This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

34.59 t_symbol Struct Reference

The symbol.

```
#include <ext_mess.h>
```

Collaboration diagram for t_symbol:



Data Fields

- char * [s_name](#)
name: a c-string
- struct object * [s_thing](#)
possible binding to a [t_object](#)

34.59.1 Detailed Description

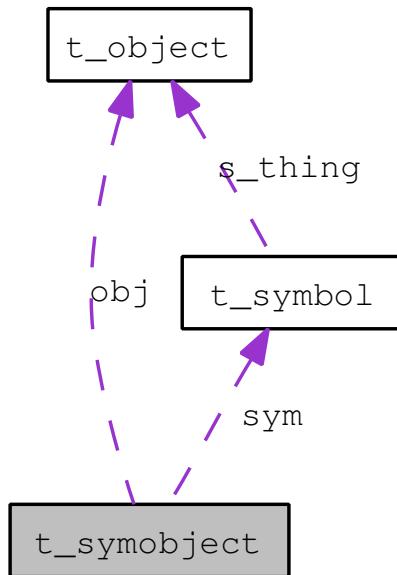
The symbol. Note: You should *never* manipulate the s_name field of the [t_symbol](#) directly! Doing so will corrupt Max's symbol table. Instead, *always* use [gensym\(\)](#) to get a symbol with the desired string contents for the s_name field.

34.60 t_symobject Struct Reference

The symobject data structure.

```
#include <ext_symobject.h>
```

Collaboration diagram for t_symobject:



Data Fields

- `t_object obj`
Max object header.
- `t_symbol * sym`
The symbol contained by the object.
- `long flags`
Any user-flags you wish to set or get.
- `void * thing`
A generic pointer for attaching additional data to the symobject.

34.60.1 Detailed Description

The symobject data structure.

34.61 t_tinyobject Struct Reference

The tiny object structure sits at the head of any object to which you may pass messages (and which you may feed to [freeobject\(\)](#)).

```
#include <ext_mess.h>
```

Data Fields

- struct messlist * [t_messlist](#)
list of messages and methods

34.61.1 Detailed Description

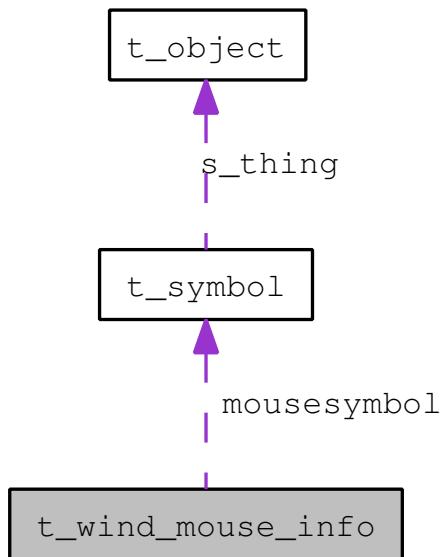
The tiny object structure sits at the head of any object to which you may pass messages (and which you may feed to [freeobject\(\)](#)). In general, you should use [t_object](#) instead.

34.62 t_wind_mouse_info Struct Reference

t_wind_mouse_info_struct provided by jit.window and jit.pwindow mouse events

```
#include <jit.gl.h>
```

Collaboration diagram for t_wind_mouse_info:



Data Fields

- Atom `mouseatoms` [8]
h, v, (up/down), cmdKey, shiftKey, alphaLock, option, control.
- int `argc`
argument count
- `t_symbol * mousesymbol`
mouse event type

34.62.1 Detailed Description

t_wind_mouse_info_struct provided by jit.window and jit.pwindow mouse events

34.63 t_zll Struct Reference

A simple doubly-linked list used by the [t_funbuff](#) object.

```
#include <ext_maxtypes.h>
```

34.63.1 Detailed Description

A simple doubly-linked list used by the [t_funbuff](#) object.

34.64 word Union Reference

Union for packing any of the datum defined in [e_max_atomtypes](#).

```
#include <ext_mess.h>
```

Data Fields

- long [w_long](#)
long integer
- t_atom_float [w_float](#)
32-bit float
- struct symbol * [w_sym](#)
pointer to a symbol in the Max symbol table
- struct object * [w_obj](#)
pointer to a [t_object](#) or other generic pointer

34.64.1 Detailed Description

Union for packing any of the datum defined in [e_max_atomtypes](#).

Index

A_CANT
 atom, [359](#)
A_COMMAT
 atom, [359](#)
A_DEFER
 atom, [359](#)
A_DEFER_LOW
 atom, [359](#)
A_DEFFLOAT
 atom, [359](#)
A_DEFLONG
 atom, [359](#)
A_DEFSYM
 atom, [359](#)
A_DOLLAR
 atom, [359](#)
A_DOLLSYM
 atom, [359](#)
A_FLOAT
 atom, [359](#)
A_GIMME
 atom, [359](#)
A_GIMMEBACK
 atom, [359](#)
A_LONG
 atom, [359](#)
A_NOTHING
 atom, [359](#)
A_OBJ
 atom, [359](#)
A_SEMI
 atom, [359](#)
A_SYM
 atom, [359](#)
A_USURP
 atom, [359](#)
A_USURP_LOW
 atom, [360](#)
aaCancel
 misc, [440](#)
aaNo
 misc, [440](#)
aaYes
 misc, [440](#)
addbang

 class_old, [260](#)
addfloat
 class_old, [260](#)
addftx
 class_old, [260](#)
addint
 class_old, [260](#)
addinx
 class_old, [261](#)
addmess
 class_old, [261](#)
alias
 class_old, [261](#)
atom
 A_CANT, [359](#)
 A_COMMAT, [359](#)
 A_DEFER, [359](#)
 A_DEFER_LOW, [359](#)
 A_DEFFLOAT, [359](#)
 A_DEFLONG, [359](#)
 A_DEFSYM, [359](#)
 A_DOLLAR, [359](#)
 A_DOLLSYM, [359](#)
 A_FLOAT, [359](#)
 A_GIMME, [359](#)
 A_GIMMEBACK, [359](#)
 A_LONG, [359](#)
 A_NOTHING, [359](#)
 A_OBJ, [359](#)
 A_SEMI, [359](#)
 A_SYM, [359](#)
 A_USURP, [359](#)
 A_USURP_LOW, [360](#)
atom_alloc
 atom_alloc, [360](#)
 atom_alloc_array, [360](#)
 atom_arg_getdouble, [360](#)
 atom_arg_getfloat, [361](#)
 atom_arg_getlong, [361](#)
 atom_arg_getobjclass, [362](#)
 atom_arg_getsym, [362](#)
 atom_copy, [363](#)
 atom_getatom_array, [363](#)
 atom_getchar_array, [363](#)
 atom_getcharfix, [364](#)
 atom_getdouble_array, [364](#)

atom_getfloat, 364
 atom_getfloat_array, 365
 atom_getformat, 365
 atom_getlong, 365
 atom_getlong_array, 366
 atom_getobj, 366
 atom_getobj_array, 366
 atom_getobjclass, 367
 atom_getsym, 367
 atom_getsym_array, 367
 atom_gettext, 368
 atom_gettype, 368
 atom_setatom_array, 368
 atom_setchar_array, 369
 atom_setdouble_array, 369
 atom_setfloat, 369
 atom_setfloat_array, 369
 atom_setformat, 370
 atom_setlong, 370
 atom_setlong_array, 371
 atom_setobj, 371
 atom_setobj_array, 371
 atom_setparse, 371
 atom_setsym, 372
 atom_setsym_array, 372
 atomisatomarray, 372
 atomisdictionary, 373
 atomisstring, 373
 e_max_atom_gettext_flags, 358
 e_max_atomtypes, 359
 OBEX_UTIL_ATOM_GETTEXT_-
 COMMA_DELIM, 359
 OBEX_UTIL_ATOM_GETTEXT_-
 DEFAULT, 358
 OBEX_UTIL_ATOM_GETTEXT_FORCE_-
 ZEROS, 359
 OBEX_UTIL_ATOM_GETTEXT_NUM_-
 HI_RES, 359
 OBEX_UTIL_ATOM_GETTEXT_SYM_-
 FORCE_QUOTE, 359
 OBEX_UTIL_ATOM_GETTEXT_SYM_-
 NO_QUOTE, 359
 OBEX_UTIL_ATOM_GETTEXT_-
 TRUNCATE_ZEROS, 358
 postargs, 373
 Atom Array, 276
 Atom Module, 679
 atom_alloc
 atom, 360
 atom_alloc_array
 atom, 360
 atom_arg_getdouble
 atom, 360
 atom_arg_getfloat
 atom, 361
 atom_arg_getlong
 atom, 361
 atom_arg_getobjcclass
 atom, 362
 atom_arg_getsym
 atom, 362
 atom_copy
 atom, 363
 atom_getatom_array
 atom, 363
 atom_getchar_array
 atom, 363
 atom_getcharfix
 atom, 364
 atom_getdouble_array
 atom, 364
 atom_getfloat
 atom, 364
 atom_getfloat_array
 atom, 365
 atom_getformat
 atom, 365
 atom_getlong
 atom, 365
 atom_getlong_array
 atom, 366
 atom_getobj
 atom, 366
 atom_getobj_array
 atom, 366
 atom_getobjcclass
 atom, 367
 atom_getsym
 atom, 367
 atom_getsym_array
 atom, 367
 atom_gettext
 atom, 368
 atom_gettype
 atom, 368
 atom_setatom_array
 atom, 368
 atom_setchar_array
 atom, 369
 atom_setdouble_array
 atom, 369
 atom_setfloat
 atom, 369
 atom_setfloat_array
 atom, 369
 atom_setformat
 atom, 370
 atom_setlong

atom, 370
atom_setlong_array
 atom, 371
atom_setobj
 atom, 371
atom_setobj_array
 atom, 371
atom_setparse
 atom, 371
atom_setsym
 atom, 372
atom_setsym_array
 atom, 372
atomarray
 atomarray_appendatom, 277
 atomarray_appendatoms, 277
 atomarray_chuckindex, 278
 atomarray_clear, 278
 atomarray_copyatoms, 278
 atomarray_duplicate, 279
 ATOMARRAY_FLAG_FREECHILDREN,
 277
 atomarray_flags, 279
 atomarray_funall, 279
 atomarray_getatoms, 280
 atomarray_getflags, 280
 atomarray_getindex, 280
 atomarray_getsize, 281
 atomarray_new, 281
 atomarray_setatoms, 282
atomarray_appendatom
 atomarray, 277
atomarray_appendatoms
 atomarray, 277
atomarray_chuckindex
 atomarray, 278
atomarray_clear
 atomarray, 278
atomarray_copyatoms
 atomarray, 278
atomarray_duplicate
 atomarray, 279
ATOMARRAY_FLAG_FREECHILDREN
 atomarray, 277
atomarray_flags
 atomarray, 279
atomarray_funall
 atomarray, 279
atomarray_getatoms
 atomarray, 280
atomarray_getflags
 atomarray, 280
atomarray_getindex
 atomarray, 280
atomarray_getsize
 atomarray, 281
atomarray_new
 atomarray, 281
atomarray_setatoms
 atomarray, 282
atomarray_setobj
 atomarray, 281
atomarray_setsize
 atomarray, 281
atomarray_text
 atomarray, 281
atombuf
 atombuf_free, 374
 atombuf_new, 374
 atombuf_text, 375
atombuf_free
 atombuf, 374
atombuf_new
 atombuf, 374
atombuf_text
 atombuf, 375
Atombufs, 374
ATOMIC_DECREMENT
 threading, 598
ATOMIC_INCREMENT
 threading, 598
atomisatomarray
 atom, 372
atomisdictionary
 atom, 373
atomisstring
 atom, 373
atommod
 jit_atom_arg_getdouble, 680
 jit_atom_arg_getfloat, 680
 jit_atom_arg_getlong, 680
 jit_atom_arg_getsym, 680
 jit_atom_getcharfix, 681
 jit_atom_getfloat, 681
 jit_atom_getlong, 681
 jit_atom_getobj, 681
 jit_atom_getsym, 682
 jit_atom_setfloat, 682
 jit_atom_setlong, 682
 jit_atom_setobj, 682
 jit_atom_setsym, 683
Atoms, 355
atoms_to_jrgba
 color, 650
attr
 ATTR_FLAGS_NONE, 231
 ATTR_GET_OPAQUE, 231
 ATTR_GET_OPAQUE_USER, 231
 ATTR_SET_OPAQUE, 231
 ATTR_SET_OPAQUE_USER, 231
 attr_addfilter_clip, 231
 attr_addfilter_clip_scale, 231
 attr_addfilterget_clip, 232
 attr_addfilterget_clip_scale, 232

attr_addfilterget_proc, 233
 attr_addfilterset_clip, 233
 attr_addfilterset_clip_scale, 234
 attr_addfilterset_proc, 234
 attr_args_dictionary, 235
 attr_args_offset, 235
 attr_args_process, 236
 attr_dictionary_process, 236
 attr_offset_array_new, 236
 attr_offset_new, 237
 attribute_new, 238
 CLASS_ATTR_ACCESSORS, 195
 CLASS_ATTR_ADD_FLAGS, 196
 CLASS_ATTR_ALIAS, 196
 CLASS_ATTR_ATOM, 196
 CLASS_ATTR_ATOM_ARRAY, 197
 CLASS_ATTR_ATOM_VARSIZE, 197
 CLASS_ATTR_CATEGORY, 197
 CLASS_ATTR_CHAR, 198
 CLASS_ATTR_CHAR_ARRAY, 198
 CLASS_ATTR_CHAR_VARSIZE, 198
 CLASS_ATTR_DEFAULT, 199
 CLASS_ATTR_DEFAULT_PAINT, 199
 CLASS_ATTR_DEFAULT_SAVE, 199
 CLASS_ATTR_DEFAULT_SAVE_PAINT,
 200
 CLASS_ATTR_DEFAULTNAME, 200
 CLASS_ATTR_DEFAULTNAME_PAINT,
 201
 CLASS_ATTR_DEFAULTNAME_SAVE,
 201
 CLASS_ATTR_DEFAULTNAME_SAVE_-
 PAINT, 201
 CLASS_ATTR_DOUBLE, 202
 CLASS_ATTR_DOUBLE_ARRAY, 202
 CLASS_ATTR_DOUBLE_VARSIZE, 202
 CLASS_ATTR_ENUM, 203
 CLASS_ATTR_ENUMINDEX, 203
 CLASS_ATTR_FILTER_CLIP, 204
 CLASS_ATTR_FILTER_MAX, 204
 CLASS_ATTR_FILTER_MIN, 205
 CLASS_ATTR_FLOAT, 205
 CLASS_ATTR_FLOAT_ARRAY, 205
 CLASS_ATTR_FLOAT_VARSIZE, 206
 CLASS_ATTR_INVISIBLE, 206
 CLASS_ATTR_LABEL, 206
 CLASS_ATTR_LONG, 206
 CLASS_ATTR_LONG_ARRAY, 207
 CLASS_ATTR_LONG_VARSIZE, 207
 CLASS_ATTR_MAX, 207
 CLASS_ATTR_MIN, 208
 CLASS_ATTR_OBJ, 208
 CLASS_ATTR_OBJ_ARRAY, 209
 CLASS_ATTR_OBJ_VARSIZE, 209
 CLASS_ATTR_ORDER, 209
 CLASS_ATTR_PAINT, 210
 CLASS_ATTR_REMOVE_FLAGS, 210
 CLASS_ATTR_RGBA, 210
 CLASS_ATTR_SAVE, 211
 CLASS_ATTR_STYLE, 211
 CLASS_ATTR_STYLE_LABEL, 211
 CLASS_ATTR_SYM, 212
 CLASS_ATTR_SYM_ARRAY, 212
 CLASS_ATTR_SYM_VARSIZE, 212
 CLASS_METHOD_ATTR_PARSE, 213
 CLASS_STICKY_ATTR, 213
 CLASS_STICKY_ATTR_CLEAR, 214
 CLASS_STICKY_METHOD, 214
 CLASS_STICKY_METHOD_CLEAR, 215
 e_max_attrflags, 231
 OBJ_ATTR_ATOM, 215
 OBJ_ATTR_ATOM_ARRAY, 215
 OBJ_ATTR_CHAR, 216
 OBJ_ATTR_CHAR_ARRAY, 216
 OBJ_ATTR_DEFAULT, 216
 OBJ_ATTR_DEFAULT_SAVE, 217
 OBJ_ATTR_DOUBLE, 217
 OBJ_ATTR_DOUBLE_ARRAY, 217
 OBJ_ATTR_FLOAT, 217
 OBJ_ATTR_FLOAT_ARRAY, 218
 OBJ_ATTR_LONG, 218
 OBJ_ATTR_LONG_ARRAY, 218
 OBJ_ATTR_OBJ, 218
 OBJ_ATTR_OBJ_ARRAY, 219
 OBJ_ATTR_SAVE, 219
 OBJ_ATTR_SYM, 219
 OBJ_ATTR_SYM_ARRAY, 219
 object_addattr, 239
 object_attr_get, 239
 object_attr_get_rect, 240
 object_attr_getchar_array, 240
 object_attr_getcolor, 240
 object_attr_getdouble_array, 241
 object_attr_getdump, 241
 object_attr_getfloat, 241
 object_attr_getfloat_array, 242
 object_attr_getjrgba, 242
 object_attr_getlong, 242
 object_attr_getlong_array, 243
 object_attr_getpt, 243
 object_attr_getsize, 243
 object_attr_getsym, 244
 object_attr_getsym_array, 244
 object_attr_method, 244
 object_attr_set_rect, 245
 object_attr_setchar_array, 245
 object_attr_setcolor, 245
 object_attr_setdouble_array, 246

object_attr_setfloat, 246
object_attr_setfloat_array, 246
object_attr_setjrgba, 247
object_attr_setlong, 247
object_attr_setlong_array, 247
object_attr_setparse, 248
object_attr_setpt, 248
object_attr_setsize, 248
object_attr_setsym, 249
object_attr_setsym_array, 249
object_attr_setvalueof, 249
object_attr_usercanget, 250
object_attr_usercanset, 250
object_chuckattr, 250
object_deleteattr, 251
object_new_parse, 251
STATIC_ATTR_ATOM, 220
STATIC_ATTR_ATOM_ARRAY, 220
STATIC_ATTR_CHAR, 220
STATIC_ATTR_CHAR_ARRAY, 220
STATIC_ATTR_DOUBLE, 221
STATIC_ATTR_DOUBLE_ARRAY, 221
STATIC_ATTR_FLOAT, 221
STATIC_ATTR_FLOAT_ARRAY, 221
STATIC_ATTR_LONG, 222
STATIC_ATTR_LONG_ARRAY, 222
STATIC_ATTR_OBJ, 222
STATIC_ATTR_OBJ_ARRAY, 222
STATIC_ATTR_SYM, 223
STATIC_ATTR_SYM_ARRAY, 223
STRUCT_ATTR_ATOM, 223
STRUCT_ATTR_ATOM_ARRAY, 224
STRUCT_ATTR_ATOM_VARSIZE, 224
STRUCT_ATTR_CHAR, 224
STRUCT_ATTR_CHAR_ARRAY, 225
STRUCT_ATTR_CHAR_VARSIZE, 225
STRUCT_ATTR_DOUBLE, 225
STRUCT_ATTR_DOUBLE_ARRAY, 226
STRUCT_ATTR_DOUBLE_VARSIZE, 226
STRUCT_ATTR_FLOAT, 226
STRUCT_ATTR_FLOAT_ARRAY, 227
STRUCT_ATTR_FLOAT_VARSIZE, 227
STRUCT_ATTR_LONG, 227
STRUCT_ATTR_LONG_ARRAY, 228
STRUCT_ATTR_LONG_VARSIZE, 228
STRUCT_ATTR_OBJ, 228
STRUCT_ATTR_OBJ_ARRAY, 229
STRUCT_ATTR_OBJ_VARSIZE, 229
STRUCT_ATTR_SYM, 229
STRUCT_ATTR_SYM_ARRAY, 230
STRUCT_ATTR_SYM_VARSIZE, 230
ATTR_FLAGS_NONE
 attr, 231
ATTR_GET_OPAQUE

attr, 231
ATTR_GET_OPAQUE_USER
 attr, 231
ATTR_SET_OPAQUE
 attr, 231
ATTR_SET_OPAQUE_USER
 attr, 231
attr_addfilter_clip
 attr, 231
attr_addfilter_clip_scale
 attr, 231
attr_addfilterget_clip
 attr, 232
attr_addfilterget_clip_scale
 attr, 232
attr_addfilterget_proc
 attr, 233
attr_addfilterset_clip
 attr, 233
attr_addfilterset_clip_scale
 attr, 234
attr_addfilterset_proc
 attr, 234
attr_args_dictionary
 attr, 235
attr_args_offset
 attr, 235
attr_args_process
 attr, 236
attr_dictionary_process
 attr, 236
attr_offset_array_new
 attr, 236
attr_offset_new
 attr, 237
Attribute Module, 684
attribute_new
 attr, 238
Attributes, 181
attrmod
 jit_attr_canget, 686
 jit_attr_canset, 686
 jit_attr_filter_clip_new, 687
 jit_attr_filter_proc_new, 687
 jit_attr_filterget, 687
 jit_attr_filterset, 688
 jit_attr_get, 688
 jit_attr_getchar_array, 688
 jit_attr_getdouble_array, 689
 jit_attr_getfloat, 689
 jit_attr_getfloat_array, 689
 jit_attr_getlong, 690
 jit_attr_getlong_array, 690
 jit_attr_getmethod, 690

jit_attr_getname, 690
 jit_attr_getsym, 691
 jit_attr_getsym_array, 691
 jit_attr_gettime, 691
 jit_attr_offset_array_new, 692
 jit_attr_offset_new, 692
 jit_attr_set, 692
 jit_attr_setchar_array, 693
 jit_attr_setdouble_array, 693
 jit_attr_setfloat, 693
 jit_attr_setfloat_array, 694
 jit_attr_setlong, 694
 jit_attr_setlong_array, 694
 jit_attr_setsym, 695
 jit_attr_setsym_array, 695
 jit_attr_symcompare, 695
 jit_attr_usercanget, 695
 jit_attr_usercanset, 696
 jit_attribute_new, 696

bangout
 inout, 267

BEGIN_USING_C_LINKAGE
 misc, 438

Binary Module, 697

binbuf

- binbuf_append, 377
- binbuf_eval, 377
- binbuf_getatom, 377
- binbuf_insert, 378
- binbuf_new, 378
- binbuf_set, 378
- binbuf_text, 378
- binbuf_totext, 379
- binbuf_vinsert, 379
- readatom, 380

binbuf_append

- binbuf, 377

binbuf_eval

- binbuf, 377

binbuf_getatom

- binbuf, 377

binbuf_insert

- binbuf, 378

binbuf_new

- binbuf, 378

binbuf_set

- binbuf, 378

binbuf_text

- binbuf, 378

binbuf_totext

- binbuf, 379

binbuf_vinsert

- binbuf, 379

Binbufs, 376
binmod

- jit_bin_read_chunk_info, 697
- jit_bin_read_header, 697
- jit_bin_read_matrix, 698
- jit_bin_write_header, 698
- jit_bin_write_matrix, 698

Box Layer, 671

boxlayer

- jbox_end_layer, 672
- jbox_invalidate_layer, 672
- jbox_paint_layer, 672
- jbox_start_layer, 672

Buffers, 488

Byte Ordering, 451

byteorder

- BYTEORDER_LSBF32, 452
- BYTEORDER_LSBF64, 452
- BYTEORDER LSBW16, 452
- BYTEORDER LSBW32, 453
- BYTEORDER MSBF32, 453
- BYTEORDER MSBF64, 453
- BYTEORDER MSBW16, 453
- BYTEORDER MSBW32, 454
- BYTEORDER SWAPF32, 454
- BYTEORDER SWAPF64, 454
- BYTEORDER SWAPW16, 454
- BYTEORDER SWAPW32, 455
- C74_BIG_ENDIAN, 455
- C74_LITTLE_ENDIAN, 455

BYTEORDER_LSBF32

- byteorder, 452

BYTEORDER_LSBF64

- byteorder, 452

BYTEORDER LSBW16

- byteorder, 452

BYTEORDER LSBW32

- byteorder, 453

BYTEORDER MSBF32

- byteorder, 453

BYTEORDER MSBF64

- byteorder, 453

BYTEORDER MSBW16

- byteorder, 453

BYTEORDER MSBW32

- byteorder, 454

BYTEORDER SWAPF32

- byteorder, 454

BYTEORDER SWAPF64

- byteorder, 454

BYTEORDER SWAPW16

- byteorder, 454

BYTEORDER SWAPW32

- byteorder, 455

C74_BIG_ENDIAN
 byteorder, 455
C74_LITTLE_ENDIAN
 byteorder, 455
calcoffset
 misc, 438
charset_convert
 unicode, 677
charset_unicodetutf8
 unicode, 677
charset_utf8_count
 unicode, 678
charset_utf8_offset
 unicode, 678
charset_utf8tunicode
 unicode, 678
class
 CLASS_FLAG_ALIAS, 254
 CLASS_FLAG_BOX, 254
 CLASS_FLAG_DO_NOT_PARSE_ATTR_ARGS, 254
 CLASS_FLAG_NEWDICTIONARY, 254
 CLASS_FLAG_NOATTRIBUTES, 254
 CLASS_FLAG_OWNATTRIBUTES, 254
 CLASS_FLAG_POLYGLOT, 254
 CLASS_FLAG_REGISTERED, 254
 CLASS_FLAG_SCHED_PURGE, 254
 CLASS_FLAG_UIOBJECT, 254
 class_addattr, 255
 class_addmethod, 255
 class_alias, 255
 CLASS_BOX, 254
 class_dumpout_wrap, 255
 class_findbyname, 256
 class_findbyname_casefree, 256
 class_free, 256
 class_is_ui, 257
 class_nameget, 257
 class_new, 257
 class_obexoffset_get, 258
 class_obexoffset_set, 258
 class_register, 258
 e_max_class_flags, 254
Class Module, 699
CLASS_FLAG_ALIAS
 class, 254
CLASS_FLAG_BOX
 class, 254
CLASS_FLAG_DO_NOT_PARSE_ATTR_ARGS
 class, 254
CLASS_FLAG_NEWDICTIONARY
 class, 254
CLASS_FLAG_NOATTRIBUTES
 class, 254
CLASS_FLAG_OWNATTRIBUTES
 class, 254
CLASS_FLAG_POLYGLOT
 class, 254
CLASS_FLAG_REGISTERED
 class, 254
CLASS_FLAG_SCHED_PURGE
 class, 254
CLASS_FLAG_UIOBJECT
 class, 254
class_addattr
 class, 255
class_addmethod
 class, 255
class_alias
 class, 255
CLASS_ATTR_ACCESSORS
 attr, 195
CLASS_ATTR_ADD_FLAGS
 attr, 196
CLASS_ATTR_ALIAS
 attr, 196
CLASS_ATTR_ATOM
 attr, 196
CLASS_ATTR_ATOM_ARRAY
 attr, 197
CLASS_ATTR_ATOM_VARSIZE
 attr, 197
CLASS_ATTR_CATEGORY
 attr, 197
CLASS_ATTR_CHAR
 attr, 198
CLASS_ATTR_CHAR_ARRAY
 attr, 198
CLASS_ATTR_CHAR_VARSIZE
 attr, 198
CLASS_ATTR_DEFAULT
 attr, 199
CLASS_ATTR_DEFAULT_PAINT
 attr, 199
CLASS_ATTR_DEFAULT_SAVE
 attr, 199
CLASS_ATTR_DEFAULT_SAVE_PAINT
 attr, 200
CLASS_ATTR_DEFAULTNAME
 attr, 200
CLASS_ATTR_DEFAULTNAME_PAINT
 attr, 201
CLASS_ATTR_DEFAULTNAME_SAVE
 attr, 201
CLASS_ATTR_DEFAULTNAME_SAVE_PAINT
 attr, 201
CLASS_ATTR_DOUBLE
 attr, 202

CLASS_ATTR_DOUBLE_ARRAY
 attr, 202
 CLASS_ATTR_DOUBLE_VARSIZE
 attr, 202
 CLASS_ATTR_ENUM
 attr, 203
 CLASS_ATTR_ENUMINDEX
 attr, 203
 CLASS_ATTR_FILTER_CLIP
 attr, 204
 CLASS_ATTR_FILTER_MAX
 attr, 204
 CLASS_ATTR_FILTER_MIN
 attr, 205
 CLASS_ATTR_FLOAT
 attr, 205
 CLASS_ATTR_FLOAT_ARRAY
 attr, 205
 CLASS_ATTR_FLOAT_VARSIZE
 attr, 206
 CLASS_ATTR_INVISIBLE
 attr, 206
 CLASS_ATTR_LABEL
 attr, 206
 CLASS_ATTR_LONG
 attr, 206
 CLASS_ATTR_LONG_ARRAY
 attr, 207
 CLASS_ATTR_LONG_VARSIZE
 attr, 207
 CLASS_ATTR_MAX
 attr, 207
 CLASS_ATTR_MIN
 attr, 208
 CLASS_ATTR_OBJ
 attr, 208
 CLASS_ATTR_OBJ_ARRAY
 attr, 209
 CLASS_ATTR_OBJ_VARSIZE
 attr, 209
 CLASS_ATTR_ORDER
 attr, 209
 CLASS_ATTR_PAINT
 attr, 210
 CLASS_ATTR_REMOVE_FLAGS
 attr, 210
 CLASS_ATTR_RGBA
 attr, 210
 CLASS_ATTR_SAVE
 attr, 211
 CLASS_ATTR_STYLE
 attr, 211
 CLASS_ATTR_STYLE_LABEL
 attr, 211

CLASS_ATTR_SYM
 attr, 212
 CLASS_ATTR_SYM_ARRAY
 attr, 212
 CLASS_ATTR_SYM_VARSIZE
 attr, 212
 CLASS_BOX
 class, 254
 class_dspinit
 msp, 484
 class_dspinitbbox
 msp, 485
 class_dumpout_wrap
 class, 255
 class_findbyname
 class, 256
 class_findbyname_casefree
 class, 256
 class_free
 class, 256
 class_is_ui
 class, 257
 CLASS_METHOD_ATTR_PARSE
 attr, 213
 class_nameget
 class, 257
 class_new
 class, 257
 class_obexoffset_get
 class, 258
 class_obexoffset_set
 class, 258
 class_old
 addbang, 260
 addfloat, 260
 addftx, 260
 addint, 260
 addinx, 261
 addmess, 261
 alias, 261
 class_setname, 261
 egefn, 262
 freeobject, 262
 getfn, 262
 newinstance, 263
 newobject, 263
 setup, 263
 typedmess, 264
 zgetfn, 264
 class_register
 class, 258
 class_setname
 class_old, 261
 CLASS_STICKY_ATTR

attr, 213
CLASS_STICKY_ATTR_CLEAR
 attr, 214
CLASS_STICKY_METHOD
 attr, 214
CLASS_STICKY_METHOD_CLEAR
 attr, 215
class_time_addattr
 time, 585
Classes, 252
classmod
 jit_class_addadornment, 700
 jit_class_addattr, 700
 jit_class_addmethod, 700
 jit_class_addtypedwrapper, 701
 jit_class_adornment_get, 701
 jit_class_attr_get, 701
 jit_class_findbyname, 701
 jit_class_free, 702
 jit_class_mess, 702
 jit_class_method, 702
 jit_class_method_addargsafe, 702
 jit_class_method_argsafe_get, 703
 jit_class_nameget, 703
 jit_class_new, 703
 jit_class_register, 704
 jit_class_symcompare, 704
 jit_class_typedwrapper_get, 704
classname_openhelp
 obj, 496
classname_openquery
 obj, 496
classname_openrefpage
 obj, 496
CLIP
 misc, 438
clock_delay
 clocks, 569
clock_fdelay
 clocks, 569
clock_gettime
 clocks, 570
clock_new
 clocks, 570
clock_unset
 clocks, 570
Clocks, 565
clocks
 clock_delay, 569
 clock_fdelay, 569
 clock_gettime, 570
 clock_new, 570
 clock_unset, 570
 _gettime, 570
scheduler_fromobject, 571
scheduler_get, 571
scheduler_gettime, 571
scheduler_new, 571
scheduler_run, 571
scheduler_set, 572
scheduler_settime, 572
scheduler_shift, 572
setclock_delay, 573
setclock_fdelay, 573
setclock_gettime, 573
setclock_gettime, 573
setclock_unset, 574
systimer_gettime, 574
color
 atoms_to_jrgba, 650
 jrgba_attr_get, 651
 jrgba_attr_set, 651
 jrgba_compare, 651
 jrgba_copy, 652
 jrgba_set, 652
 jrgba_to_atoms, 652
Colors, 650
Console, 446
console
 cpost, 446
 error, 447
 object_error, 447
 object_error_obtrusive, 448
 object_post, 448
 object_warn, 448
 ouchstring, 449
 post, 449
 postatom, 450
cpost
 console, 446
critical
 critical_enter, 606
 critical_exit, 607
 critical_free, 607
 critical_new, 607
 critical_tryenter, 607
Critical Regions, 605
critical_enter
 critical, 606
critical_exit
 critical, 607
critical_free
 critical, 607
critical_new
 critical, 607
critical_tryenter
 critical, 607

Data Storage, 273
 Data Types, 353
 Database, 283
 database
 db_close, 285
 db_open, 286
 db_query, 286
 db_query_getlastinsertid, 286
 db_query_silent, 286
 db_query_table_addcolumn, 287
 db_query_table_new, 287
 db_result_clear, 287
 db_result_datetimeminseconds, 288
 db_result_fieldname, 288
 db_result_float, 288
 db_result_long, 288
 db_result_nextrecord, 289
 db_result_numfields, 289
 db_result_numrecords, 289
 db_result_reset, 289
 db_result_string, 290
 db_transaction_end, 290
 db_transaction_flush, 290
 db_transaction_start, 290
 db_util_datetostring, 291
 db_util_stringtodate, 291
 db_view_create, 291
 db_view_getresult, 292
 db_view_remove, 292
 db_view_setquery, 292
 t_database, 285
 t_db_result, 285
 t_db_view, 285
 datastore
 e_max_datastore_flags, 275
 OBJ_FLAG_DATA, 275
 OBJ_FLAG_INHERITABLE, 275
 OBJ_FLAG_MEMORY, 275
 OBJ_FLAG_OBJ, 275
 OBJ_FLAG_REF, 275
 OBJ_FLAG_SILENT, 275
 t_cmpfn, 274
 datatypes
 t_max_err, 354
 DataView, 674
 db_close
 database, 285
 db_open
 database, 286
 db_query
 database, 286
 db_query_getlastinsertid
 database, 286
 db_query_silent

database, 286
 db_query_table_addcolumn
 database, 287
 db_query_table_new
 database, 287
 db_result_clear
 database, 287
 db_result_datetimeminseconds
 database, 288
 db_result_fieldname
 database, 288
 db_result_float
 database, 288
 db_result_long
 database, 288
 db_result_nextrecord
 database, 289
 db_result_numfields
 database, 289
 db_result_numrecords
 database, 289
 db_result_reset
 database, 289
 db_result_string
 database, 290
 db_transaction_end
 database, 290
 db_transaction_flush
 database, 290
 db_transaction_start
 database, 290
 db_util_datetostring
 database, 291
 db_util_stringtodate
 database, 291
 db_view_create
 database, 291
 db_view_getresult
 database, 292
 db_view_remove
 database, 292
 db_view_setquery
 database, 292
 defer
 threading, 598
 defer_low
 threading, 599
 Dictionary, 293
 dictionary
 dictionary_appendatom, 298
 dictionary_appendatomarray, 298
 dictionary_appendatoms, 299
 dictionary_appenddictionary, 299
 dictionary_appendfloat, 299

dictionary_appendlong, 300
dictionary_appendobject, 300
dictionary_appendstring, 300
dictionary_appendsym, 301
dictionary_chuckentry, 301
dictionary_clear, 301
dictionary_copyatoms, 302
dictionary_copydefatoms, 302
dictionary_copyentries, 303
dictionary_copyunique, 303
dictionary_deleteentry, 303
dictionary_dump, 304
dictionary_entry_getkey, 304
dictionary_entry_getvalue, 304
dictionary_entryisatomarray, 305
dictionary_entryisdictionary, 305
dictionary_entryisstring, 305
dictionary_freekeys, 305
dictionary_funall, 306
dictionary_getatom, 306
dictionary_getatomarray, 307
dictionary_getatoms, 307
dictionary_getdefatom, 307
dictionary_getdefatoms, 308
dictionary_getdeffloat, 308
dictionary_getdeflong, 309
dictionary_getdefstring, 309
dictionary_getdefsym, 309
dictionary_getdictionary, 310
dictionary_getentrycount, 310
dictionary_getfloat, 310
dictionary_getkeys, 311
dictionary_getlong, 311
dictionary_getobject, 312
dictionary_getstring, 312
dictionary_getsym, 312
dictionary_hasentry, 313
dictionary_new, 313
dictionary_read, 313
dictionary_sprintf, 313
dictionary_write, 314
postdictionary, 314
dictionary_appendatom
 dictionary, 298
dictionary_appendatomarray
 dictionary, 298
dictionary_appendatoms
 dictionary, 299
dictionary_appenddictionary
 dictionary, 299
dictionary_appendfloat
 dictionary, 299
dictionary_appendlong
 dictionary, 300
dictionary_appendobject
 dictionary, 300
dictionary_appendstring
 dictionary, 300
dictionary_appendsym
 dictionary, 301
dictionary_chuckentry
 dictionary, 301
dictionary_clear
 dictionary, 301
dictionary_copyatoms
 dictionary, 302
dictionary_copydefatoms
 dictionary, 302
dictionary_copyentries
 dictionary, 303
dictionary_copyunique
 dictionary, 303
dictionary_deleteentry
 dictionary, 303
dictionary_dump
 dictionary, 304
dictionary_entry_getkey
 dictionary, 304
dictionary_entry_getvalue
 dictionary, 304
dictionary_entryisatomarray
 dictionary, 305
dictionary_entryisdictionary
 dictionary, 305
dictionary_entryisstring
 dictionary, 305
dictionary_freekeys
 dictionary, 305
dictionary_funall
 dictionary, 306
dictionary_getatom
 dictionary, 306
dictionary_getatomarray
 dictionary, 307
dictionary_getatoms
 dictionary, 307
dictionary_getdefatom
 dictionary, 307
dictionary_getdefatoms
 dictionary, 308
dictionary_getdeffloat
 dictionary, 308
dictionary_getdeflong
 dictionary, 309
dictionary_getdefstring
 dictionary, 309
dictionary_getdefsym
 dictionary, 309

dictionary_getdictionary
 dictionary, 310
 dictionary_getentrycount
 dictionary, 310
 dictionary_getfloat
 dictionary, 310
 dictionary_getkeys
 dictionary, 311
 dictionary_getlong
 dictionary, 311
 dictionary_getobject
 dictionary, 312
 dictionary_getstring
 dictionary, 312
 dictionary_getsym
 dictionary, 312
 dictionary_hasentry
 dictionary, 313
 dictionary_new
 dictionary, 313
 dictionary_read
 dictionary, 313
 dictionary_sprintf
 dictionary, 313
 dictionary_write
 dictionary, 314
 disposhandle
 memory, 428
 dsp_add
 msp, 485
 dsp_addv
 msp, 485

 e_max_atom_gettext_flags
 atom, 358
 e_max_atomtypes
 atom, 359
 e_max_attrflags
 attr, 231
 e_max_class_flags
 class, 254
 e_max_datastore_flags
 datastore, 275
 e_max_dateflags
 systime, 579
 e_max_errorcodes
 misc, 440
 e_max_expr_types
 expr, 457
 e_max_fileinfo_flags
 files, 391
 e_max_openfile_permissions
 files, 391
 e_max_path_folder_flags
 files, 391
 e_max_path_styles
 files, 391
 e_max_path_types
 files, 391
 e_max_sysfile_posmodes
 files, 392
 e_max_sysfile_textflags
 files, 392
 e_max_systhread_mutex_flags
 threading, 598
 e_max_wind_advise_result
 misc, 440
 eAltKey
 jmouse, 476
 eAutoRepeat
 jmouse, 476
 eCapsLock
 jmouse, 476
 eCommandKey
 jmouse, 476
 eControlKey
 jmouse, 476
 egetfn
 class_old, 262
 eLeftButton
 jmouse, 476
 eMiddleButton
 jmouse, 476
 ePopupMenu
 jmouse, 476
 eRightButton
 jmouse, 476
 error
 console, 447
 error_subscribe
 misc, 440
 error_sym
 misc, 440
 error_unsubscribe
 misc, 441
 eShiftKey
 jmouse, 476
 ET_FI
 expr, 457
 ET_FLT
 expr, 457
 ET_FUNC
 expr, 457
 ET_II
 expr, 457
 ET_INT
 expr, 457
 ET_LB

expr, 457
ET_LP
 expr, 457
ET_OP
 expr, 457
ET_SI
 expr, 457
ET_STR
 expr, 457
ET_SYM
 expr, 457
ET_TBL
 expr, 457
ET_VSYM
 expr, 457
Event and File Serial Numbers, 466
evnum
 evnum_get, 466
 serialno, 466
evnum_get
 evnum, 466
Ex_ex, 889
Example Projects, 478
expr
 e_max_expr_types, 457
 ET_FLI, 457
 ET_FLT, 457
 ET_FUNC, 457
 ET_II, 457
 ET_INT, 457
 ET_LB, 457
 ET_LP, 457
 ET_OP, 457
 ET_SI, 457
 ET_STR, 457
 ET_SYM, 457
 ET_TBL, 457
 ET_VSYM, 457
 expr_eval, 458
 expr_new, 458
expr_eval
 expr, 458
expr_new
 expr, 458
Extending expr, 456
fileload
 loading_max_files, 468
files
 e_max_fileinfo_flags, 391
 e_max_openfile_permissions, 391
 e_max_path_folder_flags, 391
 e_max_path_styles, 391
 e_max_path_types, 391
 e_max_sysfile_posmodes, 392
 e_max_sysfile_textflags, 392
 fileusage_addfile, 393
 filewatcher_new, 393
 locatefile, 393
 locatefile_extended, 394
 locatefiletype, 395
 MAX_FILENAME_CHARS, 390
 open_dialog, 395
 open_promptset, 396
 PATH_FILEINFO_ALIAS, 391
 PATH_FILEINFO_FOLDER, 391
 PATH_FILEINFO_PACKAGE, 391
 PATH_FOLDER_SNIFF, 391
 PATH_READ_PERM, 391
 PATH_REPORTPACKAGEASFOLDER, 391
 PATH_RW_PERM, 391
 PATH_STYLE_COLON, 391
 PATH_STYLE_MAX, 391
 PATH_STYLE_NATIVE, 391
 PATH_STYLE_NATIVE_WIN, 391
 PATH_STYLE_SLASH, 391
 PATH_TYPE_ABSOLUTE, 392
 PATH_TYPE_BOOT, 392
 PATH_TYPE_C74, 392
 PATH_TYPE_IGNORE, 392
 PATH_TYPE_PATH, 392
 PATH_TYPE_RELATIVE, 392
 PATH_WRITE_PERM, 391
 path_closefolder, 396
 path_createssysfile, 396
 path_fileinfo, 396
 path_foldernextfile, 397
 path_from pathname, 397
 path_getapppath, 397
 path_getdefault, 398
 path_getfilemoddate, 398
 path_getmoddate, 398
 path_nameconform, 398
 path_openfolder, 399
 path_opensysfile, 399
 path_resolvefile, 399
 path_setdefault, 400
 path_top pathname, 400
 path_topotentialname, 400
 saveas_dialog, 401
 saveas_promptset, 401
 saveasdialog_extended, 402
 SYSFILE_ATMARK, 392
 SYSFILE_FROMEOF, 392
 SYSFILE_FROMMARK, 392
 SYSFILE_FROMSTART, 392
 sysfile_close, 403
 sysfile_geteof, 403

sysfile_getpos, 403
 sysfile_openhandle, 403
 sysfile_opendirsize, 404
 sysfile_read, 404
 sysfile_readtextfile, 404
 sysfile_readtchandle, 405
 sysfile_readttoptr, 405
 sysfile_seteof, 405
 sysfile_setpos, 406
 sysfile_spoolcopy, 406
 sysfile_write, 406
 sysfile_writetextfile, 407
 t_filehandle, 390
 TEXT_ENCODING_USE_FILE, 392
 TEXT_LB_MAC, 392
 TEXT_LB_NATIVE, 392
 TEXT_LB_PC, 392
 TEXT_LB_UNIX, 392
 TEXT_NULL_TERMINATE, 392
 Files and Folders, 384
 fileusage_addfile
 files, 393
 filewatcher_new
 files, 393
 floatin
 inout, 267
 floatout
 inout, 267
 freebytes
 memory, 428
 freebytes16
 memory, 428
 freeobject
 class_old, 262

 gensym
 symbol, 383
 getbytes
 memory, 429
 getbytes16
 memory, 429
 getfn
 class_old, 262
 gettime
 clocks, 570
 globalsymbol_bind
 misc, 441
 globalsymbol_dereference
 misc, 441
 globalsymbol_reference
 misc, 441
 globalsymbol_unbind
 misc, 442
 growhandle

memory, 429
 Hash Table, 316
 HASH_DEFSLOTS
 hashtab, 318
 hashtab
 HASH_DEFSLOTS, 318
 hashtab_chuck, 318
 hashtab_chuckkey, 318
 hashtab_clear, 318
 hashtab_delete, 319
 hashtab_findfirst, 319
 hashtab_flags, 320
 hashtab_funall, 320
 hashtab_getflags, 320
 hashtab_getkeyflags, 321
 hashtab_getkeys, 321
 hashtab_getsize, 322
 hashtab_keyflags, 322
 hashtab_lookup, 322
 hashtab_lookupflags, 322
 hashtab_methodall, 323
 hashtab_new, 323
 hashtab_print, 324
 hashtab_READONLY, 324
 hashtab_store, 324
 hashtab_store_safe, 324
 hashtab_storeflags, 325
 hashtab_chuck
 hashtab, 318
 hashtab_chuckkey
 hashtab, 318
 hashtab_clear
 hashtab, 318
 hashtab_delete
 hashtab, 319
 hashtab_findfirst
 hashtab, 319
 hashtab_flags
 hashtab, 320
 hashtab_funall
 hashtab, 320
 hashtab_getflags
 hashtab, 320
 hashtab_getkeyflags
 hashtab, 321
 hashtab_getkeys
 hashtab, 321
 hashtab_getsize
 hashtab, 322
 hashtab_keyflags
 hashtab, 322
 hashtab_lookup
 hashtab, 322

hashtab_lookupflags
 hashtab, 322
hashtab_methodall
 hashtab, 323
hashtab_new
 hashtab, 323
hashtab_print
 hashtab, 324
hashtab_READONLY
 hashtab, 324
hashtab_store
 hashtab, 324
hashtab_STOREFLAGS
 hashtab, 325
HitBox
 jbox, 538
HitGrowBox
 jbox, 538
HitInlet
 jbox, 538
HitLine
 jbox, 538
HitNothing
 jbox, 538
HitOutlet
 jbox, 538
HitTestResult
 jbox, 538

Index Map, 326

indexmap
 indexmap_append, 327
 indexmap_clear, 327
 indexmap_datafromindex, 327
 indexmap_delete, 327
 indexmap_delete_index, 328
 indexmap_delete_index_multi, 328
 indexmap_delete_multi, 328
 indexmap_getsize, 328
 indexmap_indexfromdata, 329
 indexmap_move, 329
 indexmap_new, 329
 indexmap_sort, 329

indexmap_append
 indexmap, 327

indexmap_clear
 indexmap, 327

indexmap_datafromindex
 indexmap, 327

indexmap_delete
 indexmap, 327

indexmap_delete_index
 indexmap, 327

time, 586
 itm_getnamed
 time, 586
 itm_getresolution
 time, 586
 itm_getstate
 time, 587
 itm_getticks
 time, 587
 itm_gettime
 time, 587
 itm_gettimesignature
 time, 587
 itm_isunitfixed
 time, 588
 itm_mstosamps
 time, 588
 itm_mstoticks
 time, 588
 itm_pause
 time, 588
 itm_reference
 time, 589
 itm_resume
 time, 589
 itm_sampstoms
 time, 589
 itm_setresolution
 time, 589
 itm_settimesignature
 time, 590
 itm_ticksbarbeatunits
 time, 590
 itm_ticksstoms
 time, 590

jbox, 532
 HitBox, 538
 HitGrowBox, 538
 HitInlet, 538
 HitLine, 538
 HitNothing, 538
 HitOutlet, 538
 HitTestResult, 538
 JBOX_FONTFACE_BOLD, 538
 JBOX_FONTFACE_BOLDITALIC, 538
 JBOX_FONTFACE_ITALIC, 538
 JBOX_FONTFACE_REGULAR, 538
 jbox_free, 538
 jbox_get_annotation, 538
 jbox_get_background, 539
 jbox_get_canhilite, 539
 jbox_get_color, 539
 jbox_get_drawfirstin, 539
 jbox_get_drawinlast, 540
 jbox_get_fontname, 540
 jbox_get fontsize, 540
 jbox_get_growboth, 540
 jbox_get_growy, 541
 jbox_get_hidden, 541
 jbox_get_hint, 541
 jbox_get_hintstring, 541
 jbox_get_id, 542
 jbox_get_ignoreclick, 542
 jbox_get_maxclass, 542
 jbox_get_nextobject, 542
 jbox_get_nogrow, 543
 jbox_get_object, 543
 jbox_get_outline, 543
 jbox_get_patcher, 543
 jbox_get_patching_position, 544
 jbox_get_patching_rect, 544
 jbox_get_patching_size, 544
 jbox_get_presentation, 544
 jbox_get_presentation_position, 545
 jbox_get_presentation_rect, 545
 jbox_get_presentation_size, 545
 jbox_get_prevobject, 545
 jbox_get_rect_for_sym, 546
 jbox_get_rect_for_view, 546
 jbox_get_textfield, 546
 jbox_get_varname, 546
 jbox_new, 547
 JBOX_NOINSPECTFIRSTIN, 538
 jbox_notify, 547
 jbox_ready, 547
 jbox_redraw, 548
 jbox_set_annotation, 548
 jbox_set_background, 548
 jbox_set_color, 548
 jbox_set_fontname, 549
 jbox_set fontsize, 549
 jbox_set_hidden, 549
 jbox_set_hint, 549
 jbox_set_hintstring, 550
 jbox_set_ignoreclick, 550
 jbox_set_outline, 550
 jbox_set_patching_position, 550
 jbox_set_patching_rect, 551
 jbox_set_patching_size, 551
 jbox_set_position, 551
 jbox_set_presentation, 551
 jbox_set_presentation_position, 552
 jbox_set_presentation_rect, 552
 jbox_set_presentation_size, 552
 jbox_set_rect, 552
 jbox_set_rect_for_sym, 553
 jbox_set_rect_for_view, 553

jbox_set_size, 553
jbox_set_varname, 553
JBOX_FONTFACE_BOLD
 jbox, 538
JBOX_FONTFACE_BOLDITALIC
 jbox, 538
JBOX_FONTFACE_ITALIC
 jbox, 538
JBOX_FONTFACE_REGULAR
 jbox, 538
jbox_end_layer
 boxlayer, 672
jbox_free
 jbox, 538
jbox_get_annotation
 jbox, 538
jbox_get_background
 jbox, 539
jbox_get_canhilite
 jbox, 539
jbox_get_color
 jbox, 539
jbox_get_drawfirstin
 jbox, 539
jbox_get_drawinlast
 jbox, 540
jbox_get_font_slant
 jfont, 641
jbox_get_font_weight
 jfont, 641
jbox_get_fontname
 jbox, 540
jbox_get_fontsize
 jbox, 540
jbox_get_growboth
 jbox, 540
jbox_get_growy
 jbox, 541
jbox_get_hidden
 jbox, 541
jbox_get_hint
 jbox, 541
jbox_get_hintstring
 jbox, 541
jbox_get_id
 jbox, 542
jbox_get_ignoreclick
 jbox, 542
jbox_get_maxclass
 jbox, 542
jbox_get_nextobject
 jbox, 542
jbox_get_nogrow
 jbox, 543
jbox_get_object
 jbox, 543
jbox_get_outline
 jbox, 543
jbox_get_patcher
 jbox, 543
jbox_get_patching_position
 jbox, 544
jbox_get_patching_rect
 jbox, 544
jbox_get_patching_size
 jbox, 544
jbox_get_presentation
 jbox, 544
jbox_get_presentation_position
 jbox, 545
jbox_get_presentation_rect
 jbox, 545
jbox_get_presentation_size
 jbox, 545
jbox_get_prevobject
 jbox, 545
jbox_get_rect_for_sym
 jbox, 546
jbox_get_rect_for_view
 jbox, 546
jbox_get_textfield
 jbox, 546
jbox_get_varname
 jbox, 546
jbox_invalidate_layer
 boxlayer, 672
jbox_new
 jbox, 547
JBOX_NOINSPECTFIRSTIN
 jbox, 538
jbox_notify
 jbox, 547
jbox_paint_layer
 boxlayer, 672
jbox_ready
 jbox, 547
jbox_redraw
 jbox, 548
jbox_set_annotation
 jbox, 548
jbox_set_background
 jbox, 548
jbox_set_color
 jbox, 548
jbox_set_fontname
 jbox, 549
jbox_set_fontsize
 jbox, 549

jbox_set_hidden
 jbox, 549

jbox_set_hint
 jbox, 549

jbox_set_hintstring
 jbox, 550

jbox_set_ignoreclick
 jbox, 550

jbox_set_outline
 jbox, 550

jbox_set_patching_position
 jbox, 550

jbox_set_patching_rect
 jbox, 551

jbox_set_patching_size
 jbox, 551

jbox_set_position
 jbox, 551

jbox_set_presentation
 jbox, 551

jbox_set_presentation_position
 jbox, 552

jbox_set_presentation_rect
 jbox, 552

jbox_set_presentation_size
 jbox, 552

jbox_set_rect
 jbox, 552

jbox_set_rect_for_sym
 jbox, 553

jbox_set_rect_for_view
 jbox, 553

jbox_set_size
 jbox, 553

jbox_set_varname
 jbox, 553

jbox_start_layer
 boxlayer, 672

jdataview
 jdataview_getclient, 674
 jdataview_new, 675
 jdataview_setclient, 675

jdataview_getclient
 jdataview, 674

jdataview_new
 jdataview, 675

jdataview_setclient
 jdataview, 675

JFont, 640

jfont
 jbox_get_font_slant, 641
 jbox_get_font_weight, 641
 jfont_create, 642
 jfont_destroy, 642

jfont_extents, 642

jfont_getfontlist, 642

jfont_reference, 643

jfont_set_font_size, 643

jfont_set_underline, 643

jfont_text_measure, 643

jfont_text_measure_wrapped, 644

JGRAPHICS_FONT_SLANT_ITALIC, 641

JGRAPHICS_FONT_SLANT_NORMAL, 641

JGRAPHICS_FONT_WEIGHT_BOLD, 641

JGRAPHICS_FONT_WEIGHT_NORMAL, 641

t_jgraphics_font_slant, 641

t_jgraphics_font_weight, 641

jfont_create
 jfont, 642

jfont_destroy
 jfont, 642

jfont_extents
 jfont, 642

jfont_getfontlist
 jfont, 642

jfont_reference
 jfont, 643

jfont_set_font_size
 jfont, 643

jfont_set_underline
 jfont, 643

jfont_text_measure
 jfont, 643

jfont_text_measure_wrapped
 jfont, 644

JGraphics, 613

jgraphics
 JGRAPHICS_FILEFORMAT_JPEG, 618
 JGRAPHICS_FILEFORMAT_PNG, 618
 JGRAPHICS_FORMAT_A8, 619
 JGRAPHICS_FORMAT_ARGB32, 618
 JGRAPHICS_FORMAT_RGB24, 618
 JGRAPHICS_TEXT_JUSTIFICATION_BOTTOM, 619
 JGRAPHICS_TEXT_JUSTIFICATION_CENTERED, 619
 JGRAPHICS_TEXT_JUSTIFICATION_HCENTERED, 619
 JGRAPHICS_TEXT_JUSTIFICATION_HJUSTIFIED, 619
 JGRAPHICS_TEXT_JUSTIFICATION_LEFT, 619
 JGRAPHICS_TEXT_JUSTIFICATION_RIGHT, 619
 JGRAPHICS_TEXT_JUSTIFICATION_TOP, 619

JGRAPHICS_TEXT JUSTIFICATION_-
VCENTERED, 619
JGRAPHICS_2PI, 618
JGRAPHICS_3PIOVER2, 618
jgraphics_append_path, 619
jgraphics_arc, 619
jgraphics_arc_negative, 619
jgraphics_close_path, 620
jgraphics_copy_path, 620
jgraphics_curve_to, 620
jgraphics_destroy, 620
jgraphics_device_to_user, 621
jgraphics_ellipse, 621
jgraphics_font_extents, 621
jgraphics_get_current_point, 621
jgraphics_getfiletypes, 621
jgraphics_line_to, 622
jgraphics_move_to, 622
jgraphics_new_path, 623
jgraphics_oval, 623
jgraphics_ovalarc, 623
jgraphics_path_destroy, 623
jgraphics_path_roundcorners, 624
JGRAPHICS_PI, 618
JGRAPHICS_PIOVER2, 618
jgraphics_position_one_rect_near_another_-
rect_but_keep_inside_a_third_rect, 624
jgraphics_rectangle, 624
jgraphics_rectangle_rounded, 624
jgraphics_rectcontainsrect, 625
jgraphics_rectintersectsrect, 625
jgraphics_reference, 625
jgraphics_rel_curve_to, 625
jgraphics_rel_line_to, 626
jgraphics_rel_move_to, 626
jgraphics_round, 626
jgraphics_select_font_face, 626
jgraphics_select_jfont, 627
jgraphics_set_font_size, 627
jgraphics_set_underline, 627
jgraphics_show_text, 627
jgraphics_system_-
canantialiastattotransparentbg, 627
jgraphics_text_measure, 628
jgraphics_text_measure_wrapped, 628
jgraphics_user_to_device, 628
t_jgraphics_fileformat, 618
t_jgraphics_format, 618
t_jgraphics_text_justification, 619
JGraphics Matrix Transformations, 645
JGRAPHICS_FILEFORMAT_JPEG
jgraphics, 618
JGRAPHICS_FILEFORMAT_PNG
jgraphics, 618

JGRAPHICS_FONT_SLANT_ITALIC
jfont, 641
JGRAPHICS_FONT_SLANT_NORMAL
jfont, 641
JGRAPHICS_FONT_WEIGHT_BOLD
jfont, 641
JGRAPHICS_FONT_WEIGHT_NORMAL
jfont, 641
JGRAPHICS_FORMAT_A8
jgraphics, 619
JGRAPHICS_FORMAT_ARGB32
jgraphics, 618
JGRAPHICS_FORMAT_RGB24
jgraphics, 618
JGRAPHICS_TEXT JUSTIFICATION_BOTTOM
jgraphics, 619
JGRAPHICS_TEXT JUSTIFICATION_-
CENTERED
jgraphics, 619
JGRAPHICS_TEXT JUSTIFICATION_-
HCENTERED
jgraphics, 619
JGRAPHICS_TEXT JUSTIFICATION_-
HJUSTIFIED
jgraphics, 619
JGRAPHICS_TEXT JUSTIFICATION_LEFT
jgraphics, 619
JGRAPHICS_TEXT JUSTIFICATION_RIGHT
jgraphics, 619
JGRAPHICS_TEXT JUSTIFICATION_TOP
jgraphics, 619
JGRAPHICS_TEXT JUSTIFICATION_-
VCENTERED
jgraphics, 619
JGRAPHICS_TEXTLAYOUT_NOWRAP
textlayout, 664
JGRAPHICS_TEXTLAYOUT_USEELLIPSIS
textlayout, 664
JGRAPHICS_2PI
jgraphics, 618
JGRAPHICS_3PIOVER2
jgraphics, 618
jgraphics_append_path
jgraphics, 619
jgraphics_arc
jgraphics, 619
jgraphics_arc_negative
jgraphics, 619
jgraphics_close_path
jgraphics, 620
jgraphics_copy_path
jgraphics, 620
jgraphics_create
jsurface, 630

jgraphics_curve_to
 jgraphics, 620

jgraphics_destroy
 jgraphics, 620

jgraphics_device_to_user
 jgraphics, 621

jgraphics_ellipse
 jgraphics, 621

jgraphics_font_extents
 jgraphics, 621

jgraphics_get_current_point
 jgraphics, 621

jgraphics_get_resource_data
 jsurface, 630

jgraphics_getfiletypes
 jgraphics, 621

jgraphics_image_surface_clear
 jsurface, 631

jgraphics_image_surface_create
 jsurface, 631

jgraphics_image_surface_create_for_data
 jsurface, 631

jgraphics_image_surface_create_from_file
 jsurface, 632

jgraphics_image_surface_create_from_filedata
 jsurface, 632

jgraphics_image_surface_create_from_resource
 jsurface, 632

jgraphics_image_surface_create_referenced
 jsurface, 633

jgraphics_image_surface_draw
 jsurface, 633

jgraphics_image_surface_draw_fast
 jsurface, 634

jgraphics_image_surface_get_height
 jsurface, 634

jgraphics_image_surface_get_pixel
 jsurface, 634

jgraphics_image_surface_get_width
 jsurface, 634

jgraphics_image_surface_scroll
 jsurface, 635

jgraphics_image_surface_set_pixel
 jsurface, 635

jgraphics_image_surface_writepng
 jsurface, 635

jgraphics_line_to
 jgraphics, 622

jgraphics_matrix_init
 jmatrix, 646

jgraphics_matrix_init_identity
 jmatrix, 646

jgraphics_matrix_init_rotate
 jmatrix, 646

jgraphics_matrix_init_scale
 jmatrix, 646

jgraphics_matrix_init_translate
 jmatrix, 647

jgraphics_matrix_invert
 jmatrix, 647

jgraphics_matrix_multiply
 jmatrix, 647

jgraphics_matrix_rotate
 jmatrix, 647

jgraphics_matrix_scale
 jmatrix, 647

jgraphics_matrix_transform_point
 jmatrix, 648

jgraphics_matrix_translate
 jmatrix, 648

jgraphics_move_to
 jgraphics, 622

jgraphics_new_path
 jgraphics, 623

jgraphics_oval
 jgraphics, 623

jgraphics_ovalarc
 jgraphics, 623

jgraphics_path_destroy
 jgraphics, 623

jgraphics_path_roundcorners
 jgraphics, 624

JGRAPHICS_PI
 jgraphics, 618

JGRAPHICS_PIOVER2
 jgraphics, 618

jgraphics_position_one_rect_near_another_rect_
 but_keep_inside_a_third_rect
 jgraphics, 624

jgraphics_rectangle
 jgraphics, 624

jgraphics_rectangle_rounded
 jgraphics, 624

jgraphics_rectcontainsrect
 jgraphics, 625

jgraphics_rectintersectsrect
 jgraphics, 625

jgraphics_reference
 jgraphics, 625

jgraphics_rel_curve_to
 jgraphics, 625

jgraphics_rel_line_to
 jgraphics, 626

jgraphics_rel_move_to
 jgraphics, 626

jgraphics_round
 jgraphics, 626

jgraphics_select_font_face

jgraphics, 626
jgraphics_select_jfont
 jgraphics, 627
jgraphics_set_font_size
 jgraphics, 627
jgraphics_set_underline
 jgraphics, 627
jgraphics_show_text
 jgraphics, 627
jgraphics_surface_destroy
 jsurface, 636
jgraphics_surface_reference
 jsurface, 636
jgraphics_system_canalialistexttotransparentbg
 jgraphics, 627
jgraphics_text_measure
 jgraphics, 628
jgraphics_text_measure_wrapped
 jgraphics, 628
jgraphics_user_to_device
 jgraphics, 628
jgraphics_write_image_surface_to_filedata
 jsurface, 636
jit.qt.movie Module, 877
jit.qt.record Module, 880
jit_atom_arg_getdouble
 atommod, 680
jit_atom_arg_getfloat
 atommod, 680
jit_atom_arg_getlong
 atommod, 680
jit_atom_arg_getsym
 atommod, 680
jit_atom_getcharfix
 atommod, 681
jit_atom_getfloat
 atommod, 681
jit_atom_getlong
 atommod, 681
jit_atom_getobj
 atommod, 681
jit_atom_getsym
 atommod, 682
jit_atom_setfloat
 atommod, 682
jit_atom_setlong
 atommod, 682
jit_atom_setobj
 atommod, 682
jit_atom_setsym
 atommod, 683
jit_attr_canget
 attrmod, 686
jit_attr_canset
 attrmod, 686
 attrmod, 688
 attrmod, 687
 attrmod, 687
 attrmod, 687
 attrmod, 688
 attrmod, 688
 attrmod, 688
 attrmod, 689
 attrmod, 689
 attrmod, 689
 attrmod, 689
 attrmod, 689
 attrmod, 690
 attrmod, 690
 attrmod, 690
 attrmod, 690
 attrmod, 690
 attrmod, 691
 attrmod, 691
 attrmod, 692
 attrmod, 692
 attrmod, 692
 attrmod, 692
 attrmod, 693
 attrmod, 693
 attrmod, 693
 attrmod, 693
 attrmod, 694
 attrmod, 694
 attrmod, 694
 attrmod, 694
 attrmod, 695
 attrmod, 695

attrmod, 695
 jit_attr_symcompare
 attrmod, 695
 jit_attr_usercanget
 attrmod, 695
 jit_attr_usercanset
 attrmod, 696
 jit_attribute_new
 attrmod, 696
 jit_bin_read_chunk_info
 binmod, 697
 jit_bin_read_header
 binmod, 697
 jit_bin_read_matrix
 binmod, 698
 jit_bin_write_header
 binmod, 698
 jit_bin_write_matrix
 binmod, 698
 jit_class_addadornment
 classmod, 700
 jit_class_addattr
 classmod, 700
 jit_class_addmethod
 classmod, 700
 jit_class_adtypedwrapper
 classmod, 701
 jit_class_adornment_get
 classmod, 701
 jit_class_attr_get
 classmod, 701
 jit_class_findbyname
 classmod, 701
 jit_class_free
 classmod, 702
 jit_class_mess
 classmod, 702
 jit_class_method
 classmod, 702
 jit_class_method_addargsafe
 classmod, 702
 jit_class_method_argsafe_get
 classmod, 703
 jit_class_nameget
 classmod, 703
 jit_class_new
 classmod, 703
 jit_class_register
 classmod, 704
 jit_class_symcompare
 classmod, 704
 jit_class_typedwrapper_get
 classmod, 704
 jit_coerce_matrix_pixmap
 qtutilsmod, 882
 jit_copy_bytes
 memorymod, 763
 jit_disposeptr
 memorymod, 764
 jit_err_from_max_err
 utilitymod, 713
 jit_error_code
 utilitymod, 714
 jit_error_sym
 utilitymod, 714
 jit_freebytes
 memorymod, 764
 jit_freemem
 memorymod, 764
 jit_getbytes
 memorymod, 764
 jit_gl_begincapture
 ob3dmod, 793
 jit_gl_bindtexture
 ob3dmod, 793
 jit_gl_drawinfo_active_textures
 ob3dmod, 793
 jit_gl_drawinfo_setup
 ob3dmod, 793
 jit_gl_endcapture
 ob3dmod, 793
 jit_gl_get_extensions
 ob3dmod, 794
 jit_gl_get_glu_version
 ob3dmod, 794
 jit_gl_get_renderer
 ob3dmod, 794
 jit_gl_get_vendor
 ob3dmod, 794
 jit_gl_get_version
 ob3dmod, 794
 jit_gl_is_extension_supported
 ob3dmod, 795
 jit_gl_is_min_version
 ob3dmod, 795
 jit_gl_report_error
 ob3dmod, 795
 jit_gl_texcoord1f
 ob3dmod, 795
 jit_gl_texcoord1fv
 ob3dmod, 796
 jit_gl_texcoord2f
 ob3dmod, 796
 jit_gl_texcoord2fv
 ob3dmod, 796
 jit_gl_texcoord3f
 ob3dmod, 796
 jit_gl_texcoord3fv

ob3dmod, 797
jit_gl_unbindtexture
 ob3dmod, 797
jit_glchunk_copy
 ob3dmod, 797
jit_glchunk_delete
 ob3dmod, 797
jit_glchunk_grid_new
 ob3dmod, 797
jit_glchunk_new
 ob3dmod, 798
jit_global_critical_enter
 utilitymod, 714
jit_global_critical_exit
 utilitymod, 714
jit_gworld_can_coerce_matrix
 qtutilsmod, 882
jit_gworld_clear
 qtutilsmod, 882
jit_gworld_matrix_equal_dim
 qtutilsmod, 883
jit_handle_free
 memorymod, 765
jit_handle_lock
 memorymod, 765
jit_handle_new
 memorymod, 765
jit_handle_size_get
 memorymod, 766
jit_handle_size_set
 memorymod, 766
jit_linklist_append
 linklistmod, 717
jit_linklist_chuck
 linklistmod, 717
jit_linklist_chuckindex
 linklistmod, 718
jit_linklist_clear
 linklistmod, 718
jit_linklist_deleteindex
 linklistmod, 718
jit_linklist_findall
 linklistmod, 719
jit_linklist_findcount
 linklistmod, 719
jit_linklist_findfirst
 linklistmod, 720
jit_linklist_free
 matrixmod, 743
jit_linklist_getindex
 linklistmod, 720
jit_linklist_getsize
 linklistmod, 720
jit_linklist_insertindex
 linklistmod, 721
jit_linklist_makearray
 linklistmod, 721
jit_linklist_methodall
 linklistmod, 722
jit_linklist_methodindex
 linklistmod, 722
jit_linklist_new
 linklistmod, 722
jit_linklist_objptr2index
 linklistmod, 723
jit_linklist_reverse
 linklistmod, 723
jit_linklist_rotate
 linklistmod, 723
jit_linklist_shuffle
 linklistmod, 724
jit_linklist_sort
 linklistmod, 724
jit_linklist_swap
 linklistmod, 724
jit_math_acos
 mathmod, 729
jit_math_acosh
 mathmod, 729
jit_math_asin
 mathmod, 729
jit_math_asinh
 mathmod, 729
jit_math_atan
 mathmod, 729
jit_math_atan2
 mathmod, 730
jit_math_atanh
 mathmod, 730
jit_math_ceil
 mathmod, 730
jit_math_cos
 mathmod, 730
jit_math_cosh
 mathmod, 731
jit_math_exp
 mathmod, 731
jit_math_exp2
 mathmod, 731
jit_math_expm1
 mathmod, 731
jit_math_fast acos
 mathmod, 732
jit_math_fast asin
 mathmod, 732
jit_math_fast atan
 mathmod, 732
jit_math_fast cos

jit_math_mod
 mathmod, 732
 jit_math_fast_invsqrt
 mathmod, 733
 jit_math_fast_sin
 mathmod, 733
 jit_math_fast_sqrt
 mathmod, 733
 jit_math_fast_tan
 mathmod, 733
 jit_math_floor
 mathmod, 734
 jit_math_fmod
 mathmod, 734
 jit_math_fold
 mathmod, 734
 jit_math_hypot
 mathmod, 734
 jit_math_is_finite
 mathmod, 735
 jit_math_is_nan
 mathmod, 735
 jit_math_is_poweroftwo
 mathmod, 735
 jit_math_is_valid
 mathmod, 735
 jit_math_j1
 mathmod, 736
 jit_math_j1_0
 mathmod, 736
 jit_math_log
 mathmod, 736
 jit_math_log10
 mathmod, 736
 jit_math_log2
 mathmod, 737
 jit_math_p1
 mathmod, 737
 jit_math_pow
 mathmod, 737
 jit_math_q1
 mathmod, 737
 jit_math_round
 mathmod, 738
 jit_math_roundup_poweroftwo
 mathmod, 738
 jit_math_sin
 mathmod, 738
 jit_math_sinh
 mathmod, 738
 jit_math_sqrt
 mathmod, 739
 jit_math_tan
 mathmod, 739
 jit_math_tanh
 mathmod, 739
 jit_math_trunc
 mathmod, 739
 jit_math_wrap
 mathmod, 740
 jit_matrix_clear
 matrixmod, 743
 jit_matrix_data
 matrixmod, 743
 jit_matrix_exprfill
 matrixmod, 743
 jit_matrix_fillplane
 matrixmod, 744
 jit_matrix_free
 matrixmod, 744
 jit_matrix_freedata
 matrixmod, 745
 jit_matrix_fromgworld
 matrixmod, 745
 jit_matrix_frommatrix
 matrixmod, 745
 jit_matrix_getcell
 matrixmod, 746
 jit_matrix_getdata
 matrixmod, 746
 jit_matrix_getinfo
 matrixmod, 746
 jit_matrix_info_default
 matrixmod, 747
 jit_matrix_jit_gl_texture
 matrixmod, 747
 jit_matrix_new
 matrixmod, 747
 jit_matrix_newcopy
 matrixmod, 748
 jit_matrix_op
 matrixmod, 748
 jit_matrix_setall
 matrixmod, 748
 jit_matrix_setcell
 matrixmod, 749
 jit_matrix_setcell1d
 matrixmod, 749
 jit_matrix_setcell2d
 matrixmod, 750
 jit_matrix_setcell3d
 matrixmod, 750
 jit_matrix_setinfo
 matrixmod, 750
 jit_matrix_setinfo_ex
 matrixmod, 751
 jit_matrix_setplane1d
 matrixmod, 751
 jit_matrix_setplane2d
 matrixmod, 751

matrixmod, 752
jit_matrix_setplane3d
 matrixmod, 752
jit_matrix_togworld
 matrixmod, 752
jit_mop_free
 mopmod, 769
jit_mop_getinput
 mopmod, 769
jit_mop_getinputlist
 mopmod, 769
jit_mop_getoutput
 mopmod, 769
jit_mop_getoutputlist
 mopmod, 770
jit_mop_input_nolink
 mopmod, 770
jit_mop_io_free
 mopmod, 770
jit_mop_io_getioproc
 mopmod, 771
jit_mop_io_getmatrix
 mopmod, 771
jit_mop_io_ioproc
 mopmod, 771
jit_mop_io_matrix
 mopmod, 772
jit_mop_io_new
 mopmod, 772
jit_mop_io_newcopy
 mopmod, 772
jit_mop_io_restrict_dim
 mopmod, 773
jit_mop_io_restrict_planeCount
 mopmod, 773
jit_mop_io_restrict_type
 mopmod, 773
jit_mop_ioproc_copy_adapt
 mopmod, 774
jit_mop_ioproc_copy_trunc
 mopmod, 774
jit_mop_ioproc_copy_trunc_zero
 mopmod, 775
jit_mop_ioproc_tosym
 mopmod, 775
jit_mop_methodall
 mopmod, 776
jit_mop_new
 mopmod, 776
jit_mop_newcopy
 mopmod, 776
jit_mop_output_nolink
 mopmod, 776
jit_mop_single_planeCount
 mopmod, 777
 jit_mop_single_type
 mopmod, 777
 jit_newptr
 memorymod, 766
 jit_ob3d_draw_chunk
 ob3dmod, 798
 jit_ob3d_free
 ob3dmod, 798
 jit_ob3d_new
 ob3dmod, 798
 jit_ob3d_set_context
 ob3dmod, 799
 jit_ob3d_setup
 ob3dmod, 799
 jit_object_attach
 objectmod, 706
 jit_object_attr_get
 objectmod, 706
 jit_object_attr_usercanGet
 objectmod, 707
 jit_object_attr_usercanSet
 objectmod, 707
 jit_object_class
 objectmod, 707
 jit_object_classname
 objectmod, 707
 jit_object_classname_compare
 objectmod, 707
 jit_object_detach
 objectmod, 708
 jit_object_exportAttrs
 objectmod, 708
 jit_object_exportSummary
 objectmod, 708
 jit_object_findRegistered
 objectmod, 709
 jit_object_findRegisteredByPtr
 objectmod, 709
 jit_object_free
 objectmod, 709
 jit_object_getMethod
 objectmod, 709
 jit_object_importAttrs
 objectmod, 710
 jit_object_method
 objectmod, 710
 jit_object_method_argsafe_get
 objectmod, 710
 jit_object_method_typed
 objectmod, 710
 jit_object_new
 objectmod, 711
 jit_object_notify

objectmod, 711
 jit_object_register
 objectmod, 711
 jit_object_unregister
 objectmod, 712
 jit_op_vector_abs_float32
 opvecmod, 818
 jit_op_vector_abs_float64
 opvecmod, 819
 jit_op_vector_abs_long
 opvecmod, 819
 jit_op_vector_absdiff_char
 opvecmod, 819
 jit_op_vector_absdiff_float32
 opvecmod, 819
 jit_op_vector_absdiff_float64
 opvecmod, 820
 jit_op_vector_absdiff_long
 opvecmod, 820
 jit_op_vector_acos_float32
 opvecmod, 820
 jit_op_vector_acos_float64
 opvecmod, 821
 jit_op_vector_acosh_float32
 opvecmod, 821
 jit_op_vector_acosh_float64
 opvecmod, 821
 jit_op_vector_add_char
 opvecmod, 821
 jit_op_vector_add_float32
 opvecmod, 822
 jit_op_vector_add_float64
 opvecmod, 822
 jit_op_vector_add_long
 opvecmod, 822
 jit_op_vector_adds_char
 opvecmod, 823
 jit_op_vector_and_char
 opvecmod, 823
 jit_op_vector_and_float32
 opvecmod, 823
 jit_op_vector_and_float64
 opvecmod, 823
 jit_op_vector_and_long
 opvecmod, 824
 jit_op_vector_asin_float32
 opvecmod, 824
 jit_op_vector_asin_float64
 opvecmod, 824
 jit_op_vector_asinh_float32
 opvecmod, 825
 jit_op_vector_asinh_float64
 opvecmod, 825
 jit_op_vector_atan2_float32
 opvecmod, 825
 jit_op_vector_atan2_float64
 opvecmod, 825
 jit_op_vector_atan_float32
 opvecmod, 826
 jit_op_vector_atan_float64
 opvecmod, 826
 jit_op_vector_atanh_float32
 opvecmod, 826
 jit_op_vector_atanh_float64
 opvecmod, 827
 jit_op_vector_avg_char
 opvecmod, 827
 jit_op_vector_avg_float32
 opvecmod, 827
 jit_op_vector_avg_float64
 opvecmod, 827
 jit_op_vector_avg_long
 opvecmod, 828
 jit_op_vector_bitand_char
 opvecmod, 828
 jit_op_vector_bitand_long
 opvecmod, 828
 jit_op_vector_bitnot_char
 opvecmod, 829
 jit_op_vector_bitnot_long
 opvecmod, 829
 jit_op_vector_bitor_char
 opvecmod, 829
 jit_op_vector_bitor_long
 opvecmod, 829
 jit_op_vector_bitxor_char
 opvecmod, 830
 jit_op_vector_bitxor_long
 opvecmod, 830
 jit_op_vector_ceil_float32
 opvecmod, 830
 jit_op_vector_ceil_float64
 opvecmod, 831
 jit_op_vector_cos_float32
 opvecmod, 831
 jit_op_vector_cos_float64
 opvecmod, 831
 jit_op_vector_cosh_float32
 opvecmod, 831
 jit_op_vector_cosh_float64
 opvecmod, 832
 jit_op_vector_div_char
 opvecmod, 832
 jit_op_vector_div_float32
 opvecmod, 832
 jit_op_vector_div_float64
 opvecmod, 833
 jit_op_vector_div_long

opvecmod, 833
jit_op_vector_eq_char
 opvecmod, 833
jit_op_vector_eq_float32
 opvecmod, 833
jit_op_vector_eq_float64
 opvecmod, 834
jit_op_vector_eq_long
 opvecmod, 834
jit_op_vector_eqp_char
 opvecmod, 834
jit_op_vector_eqp_float32
 opvecmod, 835
jit_op_vector_eqp_float64
 opvecmod, 835
jit_op_vector_eqp_long
 opvecmod, 835
jit_op_vector_exp2_float32
 opvecmod, 835
jit_op_vector_exp2_float64
 opvecmod, 836
jit_op_vector_exp_float32
 opvecmod, 836
jit_op_vector_exp_float64
 opvecmod, 836
jit_op_vector_flipdiv_char
 opvecmod, 837
jit_op_vector_flipdiv_float32
 opvecmod, 837
jit_op_vector_flipdiv_float64
 opvecmod, 837
jit_op_vector_flipdiv_long
 opvecmod, 837
jit_op_vector_flipmod_char
 opvecmod, 838
jit_op_vector_flipmod_float32
 opvecmod, 838
jit_op_vector_flipmod_float64
 opvecmod, 838
jit_op_vector_flipmod_long
 opvecmod, 839
jit_op_vector_flippass_char
 opvecmod, 839
jit_op_vector_flippass_float32
 opvecmod, 839
jit_op_vector_flippass_float64
 opvecmod, 839
jit_op_vector_flippass_long
 opvecmod, 840
jit_op_vector_flipsub_char
 opvecmod, 840
jit_op_vector_flipsub_float32
 opvecmod, 840
jit_op_vector_flipsub_long

opvecmod, 841
jit_op_vector_floor_float32
 opvecmod, 841
jit_op_vector_floor_float64
 opvecmod, 841
jit_op_vector_fold_float32
 opvecmod, 841
jit_op_vector_fold_float64
 opvecmod, 842
jit_op_vector_gt_char
 opvecmod, 842
jit_op_vector_gt_float32
 opvecmod, 842
jit_op_vector_gt_float64
 opvecmod, 843
jit_op_vector_gt_long
 opvecmod, 843
jit_op_vector_gte_char
 opvecmod, 843
jit_op_vector_gte_float32
 opvecmod, 843
jit_op_vector_gte_float64
 opvecmod, 844
jit_op_vector_gte_long
 opvecmod, 844
jit_op_vector_gtep_char
 opvecmod, 844
jit_op_vector_gtep_float32
 opvecmod, 845
jit_op_vector_gtep_float64
 opvecmod, 845
jit_op_vector_gtep_long
 opvecmod, 845
jit_op_vector_gtp_char
 opvecmod, 845
jit_op_vector_gtp_float32
 opvecmod, 846
jit_op_vector_gtp_float64
 opvecmod, 846
jit_op_vector_gtp_long
 opvecmod, 846
jit_op_vector_hypot_float32
 opvecmod, 847
jit_op_vector_hypot_float64
 opvecmod, 847
jit_op_vector_log10_float32
 opvecmod, 847
jit_op_vector_log10_float64
 opvecmod, 847
jit_op_vector_log2_float32
 opvecmod, 848
jit_op_vector_log2_float64
 opvecmod, 848
jit_op_vector_log_float32

opvecmod, 848
jit_op_vector_log_float64
 opvecmod, 849
jit_op_vector_lshift_char
 opvecmod, 849
jit_op_vector_lshift_long
 opvecmod, 849
jit_op_vector_lt_char
 opvecmod, 849
jit_op_vector_lt_float32
 opvecmod, 850
jit_op_vector_lt_float64
 opvecmod, 850
jit_op_vector_lt_long
 opvecmod, 850
jit_op_vector_lte_char
 opvecmod, 851
jit_op_vector_lte_float32
 opvecmod, 851
jit_op_vector_lte_float64
 opvecmod, 851
jit_op_vector_lte_long
 opvecmod, 851
jit_op_vector_ltep_char
 opvecmod, 852
jit_op_vector_ltep_float32
 opvecmod, 852
jit_op_vector_ltep_float64
 opvecmod, 852
jit_op_vector_ltep_long
 opvecmod, 853
jit_op_vector_ltp_char
 opvecmod, 853
jit_op_vector_ltp_float32
 opvecmod, 853
jit_op_vector_ltp_float64
 opvecmod, 853
jit_op_vector_ltp_long
 opvecmod, 854
jit_op_vector_max_char
 opvecmod, 854
jit_op_vector_max_float32
 opvecmod, 854
jit_op_vector_max_float64
 opvecmod, 855
jit_op_vector_max_long
 opvecmod, 855
jit_op_vector_min_char
 opvecmod, 855
jit_op_vector_min_float32
 opvecmod, 855
jit_op_vector_min_float64
 opvecmod, 856
jit_op_vector_min_long

opvecmod, 856
jit_op_vector_mod_char
 opvecmod, 856
jit_op_vector_mod_float32
 opvecmod, 857
jit_op_vector_mod_float64
 opvecmod, 857
jit_op_vector_mod_long
 opvecmod, 857
jit_op_vector_mult_char
 opvecmod, 857
jit_op_vector_mult_float32
 opvecmod, 858
jit_op_vector_mult_float64
 opvecmod, 858
jit_op_vector_mult_long
 opvecmod, 858
jit_op_vector_neq_char
 opvecmod, 859
jit_op_vector_neq_float32
 opvecmod, 859
jit_op_vector_neq_float64
 opvecmod, 859
jit_op_vector_neq_long
 opvecmod, 859
jit_op_vector_neqp_char
 opvecmod, 860
jit_op_vector_neqp_float32
 opvecmod, 860
jit_op_vector_neqp_float64
 opvecmod, 860
jit_op_vector_neqp_long
 opvecmod, 861
jit_op_vector_not_char
 opvecmod, 861
jit_op_vector_not_float32
 opvecmod, 861
jit_op_vector_not_float64
 opvecmod, 861
jit_op_vector_not_long
 opvecmod, 862
jit_op_vector_or_char
 opvecmod, 862
jit_op_vector_or_float32
 opvecmod, 862
jit_op_vector_or_float64
 opvecmod, 863
jit_op_vector_or_long
 opvecmod, 863
jit_op_vector_pass_char
 opvecmod, 863
jit_op_vector_pass_float32
 opvecmod, 863
jit_op_vector_pass_float64

opvecmod, 864
jit_op_vector_pass_long
 opvecmod, 864
jit_op_vector_pow_float32
 opvecmod, 864
jit_op_vector_pow_float64
 opvecmod, 865
jit_op_vector_round_float32
 opvecmod, 865
jit_op_vector_round_float64
 opvecmod, 865
jit_op_vector_rshift_char
 opvecmod, 865
jit_op_vector_rshift_long
 opvecmod, 866
jit_op_vector_sin_float32
 opvecmod, 866
jit_op_vector_sin_float64
 opvecmod, 866
jit_op_vector_sinh_float32
 opvecmod, 867
jit_op_vector_sinh_float64
 opvecmod, 867
jit_op_vector_sqrt_float32
 opvecmod, 867
jit_op_vector_sqrt_float64
 opvecmod, 867
jit_op_vector_sub_char
 opvecmod, 868
jit_op_vector_sub_float32
 opvecmod, 868
jit_op_vector_sub_float64
 opvecmod, 868
jit_op_vector_sub_long
 opvecmod, 869
jit_op_vector_subs_char
 opvecmod, 869
jit_op_vector_tan_float32
 opvecmod, 869
jit_op_vector_tan_float64
 opvecmod, 869
jit_op_vector_tanh_float32
 opvecmod, 870
jit_op_vector_tanh_float64
 opvecmod, 870
jit_op_vector_trunc_float32
 opvecmod, 870
jit_op_vector_trunc_float64
 opvecmod, 871
jit_op_vector_wrap_float32
 opvecmod, 871
jit_op_vector_wrap_float64
 opvecmod, 871
jit_parallel_ndim_calc
 parallelutilmod, 778
 jit_parallel_ndim_simplecalc1
 parallelutilmod, 778
 jit_parallel_ndim_simplecalc2
 parallelutilmod, 779
 jit_parallel_ndim_simplecalc3
 parallelutilmod, 779
 jit_parallel_ndim_simplecalc4
 parallelutilmod, 780
 jit_post_sym
 utilitymod, 714
 jit_qt_codec_acodec2sym
 qtcodecmod, 873
 jit_qt_codec_getcodeclist_audio
 qtcodecmod, 873
 jit_qt_codec_getcodeclist_audio_raw
 qtcodecmod, 873
 jit_qt_codec_getcodeclist_gfx
 qtcodecmod, 873
 jit_qt_codec_getcodeclist_gfx_raw
 qtcodecmod, 874
 jit_qt_codec_getcodeclist_video
 qtcodecmod, 874
 jit_qt_codec_getcodeclist_video_raw
 qtcodecmod, 874
 jit_qt_codec_qual2sym
 qtcodecmod, 874
 jit_qt_codec_sym2acodec
 qtcodecmod, 875
 jit_qt_codec_sym2qual
 qtcodecmod, 875
 jit_qt_codec_sym2type
 qtcodecmod, 875
 jit_qt_codec_sym2type_valid
 qtcodecmod, 875
 jit_qt_codec_type2sym
 qtcodecmod, 876
 jit_qt_codec_type2sym_valid
 qtcodecmod, 876
 jit_qt_movie_matrix_calc
 qtmoviemod, 877
 jit_qt_movie_matrix_to_image
 qtmoviemod, 877
 jit_qt_movie_new
 qtmoviemod, 878
 jit_qt_movie_read_typed
 qtmoviemod, 878
 jit_qt_record_matrix_calc
 qtrecordmod, 880
 jit_qt_record_new
 qtrecordmod, 880
 jit_qt_utils_moviedataref_create
 qtutilsmod, 883
 jit_qt_utils_moviefile_close

qtutilsmod, 883
 jit_qt_utils_moviefile_create
 qtutilsmod, 883
 jit_qt_utils_str2type
 qtutilsmod, 884
 jit_qt_utils_tempfile
 qtutilsmod, 884
 jit_qt_utils_tempmoviefile_create
 qtutilsmod, 884
 jit_qt_utils_trackmedia_add
 qtutilsmod, 885
 jit_qt_utils_trackmedia_dispose
 qtutilsmod, 885
 jit_qt_utils_trackmedia_get
 qtutilsmod, 885
 jit_qt_utils_trackname_get
 qtutilsmod, 886
 jit_qt_utils_trackname_set
 qtutilsmod, 886
 jit_qt_utils_tracktype_get
 qtutilsmod, 886
 jit_qt_utils_tracktypecode_get
 qtutilsmod, 886
 jit_qt_utils_type2str
 qtutilsmod, 887
 jit_rand
 utilitymod, 714
 jit_rand_setseed
 utilitymod, 715
 Jitter, 408
 jkeyboard_getcurrentmodifiers
 jmouse, 476
 jkeyboard_getcurrentmodifiers_realtime
 jmouse, 476
 jmatrix
 jgraphics_matrix_init, 646
 jgraphics_matrix_init_identity, 646
 jgraphics_matrix_init_rotate, 646
 jgraphics_matrix_init_scale, 646
 jgraphics_matrix_init_translate, 647
 jgraphics_matrix_invert, 647
 jgraphics_matrix_multiply, 647
 jgraphics_matrix_rotate, 647
 jgraphics_matrix_scale, 647
 jgraphics_matrix_transform_point, 648
 jgraphics_matrix_translate, 648
 jmonitor
 jmonitor_getdisplayrect, 471
 jmonitor_getdisplayrect_foralldisplays, 471
 jmonitor_getdisplayrect_forpoint, 471
 jmonitor_getnumdisplays, 472
 jmonitor_getdisplayrect
 jmonitor, 471
 jmonitor_getdisplayrect_foralldisplays
 jmonitor, 471
 jmonitor_getdisplayrect_forpoint
 jmonitor, 471
 jmonitor_getnumdisplays
 jmonitor, 472
 jmouse
 eAltKey, 476
 eAutoRepeat, 476
 eCapsLock, 476
 eCommandKey, 476
 eControlKey, 476
 eLeftButton, 476
 eMiddleButton, 476
 ePopupMenu, 476
 eRightButton, 476
 eShiftKey, 476
 jkeyboard_getcurrentmodifiers, 476
 jkeyboard_getcurrentmodifiers_realtime, 476
 JMOUSE_CURSOR_ARROW, 475
 JMOUSE_CURSOR_COPYING, 475
 JMOUSE_CURSOR_CROSSHAIR, 475
 JMOUSE_CURSOR_DRAGGINGHAND, 475
 JMOUSE_CURSOR_IBEAM, 475
 JMOUSE_CURSOR_NONE, 475
 JMOUSE_CURSOR_POINTINGHAND, 475
 JMOUSE_CURSOR_RESIZE_BOTTOMEDGE, 475
 JMOUSE_CURSOR_RESIZE_BOTTOMLEFTCORNER, 476
 JMOUSE_CURSOR_RESIZE_BOTTOMRIGHTCORNER, 476
 JMOUSE_CURSOR_RESIZE_FOURWAY, 475
 JMOUSE_CURSOR_RESIZE_LEFTEDGE, 475
 JMOUSE_CURSOR_RESIZE_LEFTRIGHT, 475
 JMOUSE_CURSOR_RESIZE_RIGHTEDGE, 475
 JMOUSE_CURSOR_RESIZE_TOPEDGE, 475
 JMOUSE_CURSOR_RESIZE_TOPLEFTCORNER, 475
 JMOUSE_CURSOR_RESIZE_TOPRIGHTCORNER, 476
 JMOUSE_CURSOR_RESIZE_UPDOWN, 475
 JMOUSE_CURSOR_WAIT, 475
 jmouse_getposition_global, 476
 jmouse_setcursor, 476
 jmouse_setposition_box, 477
 jmouse_setposition_global, 477
 jmouse_setposition_view, 477

t_jmouse_cursortype, 475
t_modifiers, 476
JMOUSE_CURSOR_ARROW
 jmouse, 475
JMOUSE_CURSOR_COPYING
 jmouse, 475
JMOUSE_CURSOR_CROSSHAIR
 jmouse, 475
JMOUSE_CURSOR_DRAGGINGHAND
 jmouse, 475
JMOUSE_CURSOR_IBeam
 jmouse, 475
JMOUSE_CURSOR_NONE
 jmouse, 475
JMOUSE_CURSOR_POINTINGHAND
 jmouse, 475
JMOUSE_CURSOR_RESIZE_BOTTOMEDGE
 jmouse, 475
JMOUSE_CURSOR_RESIZE_-
 BOTTOMLEFTCORNER
 jmouse, 476
JMOUSE_CURSOR_RESIZE_-
 BOTTOMRIGHTCORNER
 jmouse, 476
JMOUSE_CURSOR_RESIZE_FOURWAY
 jmouse, 475
JMOUSE_CURSOR_RESIZE_LEFTEDGE
 jmouse, 475
JMOUSE_CURSOR_RESIZE_Lleftright
 jmouse, 475
JMOUSE_CURSOR_RESIZE_RIGHTEDGE
 jmouse, 475
JMOUSE_CURSOR_RESIZE_TOPEDGE
 jmouse, 475
JMOUSE_CURSOR_RESIZE_-
 TOPLEFTCORNER
 jmouse, 475
JMOUSE_CURSOR_RESIZE_-
 TOPRIGHTCORNER
 jmouse, 476
JMOUSE_CURSOR_RESIZE_UPDOWN
 jmouse, 475
JMOUSE_CURSOR_WAIT
 jmouse, 475
jmouse_getposition_global
 jmouse, 476
jmouse_setcursor
 jmouse, 476
jmouse_setposition_box
 jmouse, 477
jmouse_setposition_global
 jmouse, 477
jmouse_setposition_view
 jmouse, 477

jpatcher, 518
jpatcher_deleteobj, 520
jpatcher_get_bgcolor, 521
jpatcher_get_bghidden, 521
jpatcher_get_bglocked, 521
jpatcher_get_box, 521
jpatcher_get_count, 521
jpatcher_get_currentfileversion, 522
jpatcher_get_default_fontface, 522
jpatcher_get_default_fontname, 522
jpatcher_get_default_fontsize, 522
jpatcher_get_defrect, 523
jpatcher_get_dirty, 523
jpatcher_get_editing_bgcolor, 523
jpatcher_get_fghidden, 523
jpatcher_get_filename, 524
jpatcher_get_filepath, 524
jpatcher_get_fileversion, 524
jpatcher_get_firstline, 524
jpatcher_get_firstobject, 525
jpatcher_get_firstview, 525
jpatcher_get_grayscale, 525
jpatcher_get_lastobject, 526
jpatcher_get_name, 526
jpatcher_get_parentpatcher, 526
jpatcher_get_presentation, 526
jpatcher_get_rect, 527
jpatcher_get_title, 527
jpatcher_get_toppatcher, 527
jpatcher_is_patch, 527
jpatcher_set_bgcolor, 528
jpatcher_set_bghidden, 528
jpatcher_set_bglocked, 528
jpatcher_set_defrect, 528
jpatcher_set_dirty, 529
jpatcher_set_editing_bgcolor, 529
jpatcher_set_fghidden, 529
jpatcher_set_grayscale, 529
jpatcher_set_locked, 530
jpatcher_set_presentation, 530
jpatcher_set_rect, 530
jpatcher_set_title, 530
jpatcher_uniqueboxname, 531

jpatcher_deleteobj
 jpatcher, 520
jpatcher_get_bgcolor
 jpatcher, 521
jpatcher_get_bghidden
 jpatcher, 521
jpatcher_get_bglocked
 jpatcher, 521
jpatcher_get_box
 jpatcher, 521
jpatcher_get_count

jpatcher, 521
 jpatcher_get_currentfileversion
 jpatcher, 522
 jpatcher_get_default_fontface
 jpatcher, 522
 jpatcher_get_default_fontname
 jpatcher, 522
 jpatcher_get_default_fontsize
 jpatcher, 522
 jpatcher_get_defrect
 jpatcher, 523
 jpatcher_get_dirty
 jpatcher, 523
 jpatcher_get_editing_bgcolor
 jpatcher, 523
 jpatcher_get_fghidden
 jpatcher, 523
 jpatcher_get_filename
 jpatcher, 524
 jpatcher_get_filepath
 jpatcher, 524
 jpatcher_get_fileversion
 jpatcher, 524
 jpatcher_get_firstline
 jpatcher, 524
 jpatcher_get_firstobject
 jpatcher, 525
 jpatcher_get_firstview
 jpatcher, 525
 jpatcher_get_gridsize
 jpatcher, 525
 jpatcher_get_lastobject
 jpatcher, 526
 jpatcher_get_name
 jpatcher, 526
 jpatcher_get_parentpatcher
 jpatcher, 526
 jpatcher_get_presentation
 jpatcher, 526
 jpatcher_get_rect
 jpatcher, 527
 jpatcher_get_title
 jpatcher, 527
 jpatcher_get_toppatcher
 jpatcher, 527
 jpatcher_is_patchter
 jpatcher, 527
 jpatcher_set_bgcolor
 jpatcher, 528
 jpatcher_set_bghidden
 jpatcher, 528
 jpatcher_set_bglocked
 jpatcher, 528
 jpatcher_set_defrect
 jpatcher, 528
 jpatcher_set_dirty
 jpatcher, 529
 jpatcher_set_editing_bgcolor
 jpatcher, 529
 jpatcher_set_fghidden
 jpatcher, 529
 jpatcher_set_gridsize
 jpatcher, 529
 jpatcher_set_locked
 jpatcher, 530
 jpatcher_set_presentation
 jpatcher, 530
 jpatcher_set_rect
 jpatcher, 530
 jpatcher_set_title
 jpatcher, 530
 jpatcher_uniqueboxname
 jpatcher, 531
 jpatcherview, 559
 patcherview_findpatcherview, 560
 patcherview_get_jgraphics, 560
 patcherview_get_locked, 560
 patcherview_get_nextview, 560
 patcherview_get_patchter, 561
 patcherview_get_presentation, 561
 patcherview_get_rect, 561
 patcherview_get_topview, 561
 patcherview_get_visible, 562
 patcherview_get_zoomfactor, 562
 patcherview_set_locked, 562
 patcherview_set_presentation, 562
 patcherview_set_rect, 563
 patcherview_set_visible, 563
 patcherview_set_zoomfactor, 563
 jpatchline, 555
 jpatchline_get_box1, 556
 jpatchline_get_box2, 556
 jpatchline_get_color, 556
 jpatchline_get_endpoint, 556
 jpatchline_get_hidden, 557
 jpatchline_get_inletnum, 557
 jpatchline_get_nextline, 557
 jpatchline_get_nummidpoints, 557
 jpatchline_get_outletnum, 557
 jpatchline_get_startpoint, 558
 jpatchline_set_color, 558
 jpatchline_set_hidden, 558
 jpatchline_get_box1
 jpatchline, 556
 jpatchline_get_box2
 jpatchline, 556
 jpatchline_get_color
 jpatchline, 556

jpatchline_get_endpoint
 jpatchline, 556
jpatchline_get_hidden
 jpatchline, 557
jpatchline_get_inletnum
 jpatchline, 557
jpatchline_get_nextline
 jpatchline, 557
jpatchline_get_nummidpoints
 jpatchline, 557
jpatchline_get_outletnum
 jpatchline, 557
jpatchline_get_startpoint
 jpatchline, 558
jpatchline_set_color
 jpatchline, 558
jpatchline_set_hidden
 jpatchline, 558
JPattern, 649
jpopupmenu
 jpopupmenu_additem, 668
 jpopupmenu_addseperator, 668
 jpopupmenu_addsubmenu, 668
 jpopupmenu_clear, 668
 jpopupmenu_create, 668
 jpopupmenu_destroy, 669
 jpopupmenu_popup, 669
 jpopupmenu_popup_abovebox, 669
 jpopupmenu_popup_nearbox, 669
 jpopupmenu_setcolors, 670
 jpopupmenu_setfont, 670
jpopupmenu_additem
 jpopupmenu, 668
jpopupmenu_addseperator
 jpopupmenu, 668
jpopupmenu_addsubmenu
 jpopupmenu, 668
jpopupmenu_clear
 jpopupmenu, 668
jpopupmenu_create
 jpopupmenu, 668
jpopupmenu_destroy
 jpopupmenu, 669
jpopupmenu_popup
 jpopupmenu, 669
jpopupmenu_popup_abovebox
 jpopupmenu, 669
jpopupmenu_popup_nearbox
 jpopupmenu, 669
jpopupmenu_setcolors
 jpopupmenu, 670
jpopupmenu_setfont
 jpopupmenu, 670
jrgba_attr_get
 color, 651
jrgba_attr_set
 color, 651
jrgba_compare
 color, 651
jrgba_copy
 color, 652
jrgba_set
 color, 652
jrgba_to_atoms
 color, 652
JSurface, 629
jsurface
 jgraphics_create, 630
 jgraphics_get_resource_data, 630
 jgraphics_image_surface_clear, 631
 jgraphics_image_surface_create, 631
 jgraphics_image_surface_create_for_data, 631
 jgraphics_image_surface_create_from_file,
 632
 jgraphics_image_surface_create_from_-
 filedata, 632
 jgraphics_image_surface_create_from_-
 resource, 632
 jgraphics_image_surface_create_referenced,
 633
 jgraphics_image_surface_draw, 633
 jgraphics_image_surface_draw_fast, 634
 jgraphics_image_surface_get_height, 634
 jgraphics_image_surface_get_pixel, 634
 jgraphics_image_surface_get_width, 634
 jgraphics_image_surface_scroll, 635
 jgraphics_image_surface_set_pixel, 635
 jgraphics_image_surface_writepng, 635
 jgraphics_surface_destroy, 636
 jgraphics_surface_reference, 636
 jgraphics_write_image_surface_to_filedata,
 636
jsvg
 jsvg_create_from_file, 638
 jsvg_create_from_resource, 638
 jsvg_create_from_xmlstring, 639
 jsvg_destroy, 639
 jsvg_get_size, 639
 jsvg_render, 639
jsvg_create_from_file
 jsvg, 638
jsvg_create_from_resource
 jsvg, 638
jsvg_create_from_xmlstring
 jsvg, 639
jsvg_destroy
 jsvg, 639
jsvg_get_size

jsvg, 639
 jsvg_render
 jsvg, 639
 jtextlayout_create
 textlayout, 664
 jtextlayout_destroy
 textlayout, 664
 jtextlayout_draw
 textlayout, 664
 jtextlayout_getchar
 textlayout, 664
 jtextlayout_getcharbox
 textlayout, 665
 jtextlayout_getnumchars
 textlayout, 665
 jtextlayout_measure
 textlayout, 665
 jtextlayout_set
 textlayout, 666
 jtextlayout_settextcolor
 textlayout, 666
 jtextlayout_withbgcolor
 textlayout, 666
 jwind
 jwind_getactive, 473
 jwind_getat, 473
 jwind_getcount, 473
 jwind_getactive
 jwind, 473
 jwind_getat
 jwind, 473
 jwind_getcount
 jwind, 473

Linked List, 331
 Linked List Module, 716
 linklist
 linklist_append, 333
 linklist_chuck, 334
 linklist_chuckindex, 334
 linklist_chuckobject, 334
 linklist_clear, 335
 linklist_deleteindex, 335
 linklist_findall, 335
 linklist_findfirst, 336
 linklist_flags, 336
 linklist_funall, 337
 linklist_funall_break, 337
 linklist_funindex, 338
 linklist_getflags, 338
 linklist_getindex, 338
 linklist_getsize, 339
 linklist_insert_sorted, 339
 linklist_insertafterobjptr, 339
 linklist_insertbeforeobjptr, 339
 linklist_insertindex, 340
 linklist_last, 340
 linklist_makearray, 340
 linklist_match, 340
 linklist_methodall, 341
 linklist_methodindex, 341
 linklist_moveafterobjptr, 341
 linklist_movebeforeobjptr, 342
 linklist_new, 342
 linklist_next, 342
 linklist_objptr2index, 342
 linklist_prev, 343
 linklist_READONLY, 343
 linklist_reverse, 343
 linklist_rotate, 343
 linklist_shuffle, 344
 linklist_sort, 344
 linklist_substitute, 344
 linklist_swap, 345
 linklist_append
 linklist, 333
 linklist_chuck
 linklist, 334
 linklist_chuckindex
 linklist, 334
 linklist_chuckobject
 linklist, 334
 linklist_clear
 linklist, 335
 linklist_deleteindex
 linklist, 335
 linklist_findall
 linklist, 335
 linklist_findfirst
 linklist, 336
 linklist_flags
 linklist, 336
 linklist_funall
 linklist, 337
 linklist_funall_break
 linklist, 337
 linklist_funindex
 linklist, 338
 linklist_getflags
 linklist, 338
 linklist_getindex
 linklist, 338
 linklist_getsize
 linklist, 339
 linklist_insert_sorted
 linklist, 339
 linklist_insertafterobjptr
 linklist, 339

linklist_insertbeforeobjptr
 linklist, 339
linklist_insertindex
 linklist, 340
linklist_last
 linklist, 340
linklist_makearray
 linklist, 340
linklist_match
 linklist, 340
linklist_methodall
 linklist, 341
linklist_methodindex
 linklist, 341
linklist_moveafterobjptr
 linklist, 341
linklist_movebeforeobjptr
 linklist, 342
linklist_new
 linklist, 342
linklist_next
 linklist, 342
linklist_objptr2index
 linklist, 342
linklist_prev
 linklist, 343
linklist_readonly
 linklist, 343
linklist_reverse
 linklist, 343
linklist_rotate
 linklist, 343
linklist_shuffle
 linklist, 344
linklist_sort
 linklist, 344
linklist_substitute
 linklist, 344
linklist_swap
 linklist, 345
linklistmod
 jit_linklist_append, 717
 jit_linklist_chuck, 717
 jit_linklist_chuckindex, 718
 jit_linklist_clear, 718
 jit_linklist_deleteindex, 718
 jit_linklist_findall, 719
 jit_linklist_findcount, 719
 jit_linklist_findfirst, 720
 jit_linklist_getindex, 720
 jit_linklist_getsize, 720
 jit_linklist_insertindex, 721
 jit_linklist_makearray, 721
 jit_linklist_methodall, 722
 jit_linklist_methodindex, 722
 jit_linklist_new, 722
 jit_linklist_objptr2index, 723
 jit_linklist_reverse, 723
 jit_linklist_rotate, 723
 jit_linklist_shuffle, 724
 jit_linklist_sort, 724
 jit_linklist_swap, 724
listout
 inout, 269
Loading Max Files, 468
loading_max_files
 fileload, 468
 intload, 468
 readtohandle, 469
 stringload, 469
locatefile
 files, 393
locatefile_extended
 files, 394
locate filetype
 files, 395
Math Module, 726
mathmod
 jit_math_acos, 729
 jit_math_acosh, 729
 jit_math_asin, 729
 jit_math_asinh, 729
 jit_math_atan, 729
 jit_math_atan2, 730
 jit_math_atanh, 730
 jit_math_ceil, 730
 jit_math_cos, 730
 jit_math_cosh, 731
 jit_math_exp, 731
 jit_math_exp2, 731
 jit_math_expm1, 731
 jit_math_fast acos, 732
 jit_math_fast asin, 732
 jit_math_fast atan, 732
 jit_math_fast cos, 732
 jit_math_fast_invsqrt, 733
 jit_math_fast_sin, 733
 jit_math_fast_sqrt, 733
 jit_math_fast_tan, 733
 jit_math_floor, 734
 jit_math_fmod, 734
 jit_math_fold, 734
 jit_math_hypot, 734
 jit_math_is_finite, 735
 jit_math_is_nan, 735
 jit_math_is_poweroftwo, 735
 jit_math_is_valid, 735

jit_math_j1, [736](#)
 jit_math_j1_0, [736](#)
 jit_math_log, [736](#)
 jit_math_log10, [736](#)
 jit_math_log2, [737](#)
 jit_math_p1, [737](#)
 jit_math_pow, [737](#)
 jit_math_q1, [737](#)
 jit_math_round, [738](#)
 jit_math_roundup_poweroftwo, [738](#)
 jit_math_sin, [738](#)
 jit_math_sinh, [738](#)
 jit_math_sqrt, [739](#)
 jit_math_tan, [739](#)
 jit_math_tanh, [739](#)
 jit_math_trunc, [739](#)
 jit_math_wrap, [740](#)
Matrix Module, [741](#)
matrixmod
 jit_linklist_free, [743](#)
 jit_matrix_clear, [743](#)
 jit_matrix_data, [743](#)
 jit_matrix_exprfill, [743](#)
 jit_matrix_fillplane, [744](#)
 jit_matrix_free, [744](#)
 jit_matrix_freedata, [745](#)
 jit_matrix_fromgworld, [745](#)
 jit_matrix_frommatrix, [745](#)
 jit_matrix_getcell, [746](#)
 jit_matrix_getdata, [746](#)
 jit_matrix_getinfo, [746](#)
 jit_matrix_info_default, [747](#)
 jit_matrix_jit_gl_texture, [747](#)
 jit_matrix_new, [747](#)
 jit_matrix_newcopy, [748](#)
 jit_matrix_op, [748](#)
 jit_matrix_setall, [748](#)
 jit_matrix_setcell, [749](#)
 jit_matrix_setcell1d, [749](#)
 jit_matrix_setcell2d, [750](#)
 jit_matrix_setcell3d, [750](#)
 jit_matrix_setinfo, [750](#)
 jit_matrix_setinfo_ex, [751](#)
 jit_matrix_setplane1d, [751](#)
 jit_matrix_setplane2d, [752](#)
 jit_matrix_setplane3d, [752](#)
 jit_matrix_togworld, [752](#)
MAX
 misc, [439](#)
Max Wrapper Module, [754](#)
MAX_ERR_DUPLICATE
 misc, [440](#)
MAX_ERR_GENERIC
 misc, [440](#)
 MAX_ERR_INVALID_PTR
 misc, [440](#)
MAX_ERR_NONE
 misc, [440](#)
MAX_ERR_OUT_OF_MEM
 misc, [440](#)
max_addmethod_defer
 maxwrapmod, [755](#)
max_addmethod_defer_low
 maxwrapmod, [756](#)
max_addmethod_usurp
 maxwrapmod, [756](#)
max_addmethod_usurp_low
 maxwrapmod, [756](#)
MAX_FILENAME_CHARS
 files, [390](#)
max_jit_attr_args
 maxwrapmod, [756](#)
max_jit_attr_args_offset
 maxwrapmod, [756](#)
max_jit_attr_get
 maxwrapmod, [757](#)
max_jit_attr_getdump
 maxwrapmod, [757](#)
max_jit_attr_set
 maxwrapmod, [757](#)
max_jit_classex_addattr
 maxwrapmod, [757](#)
max_jit_classex_mop_mproc
 maxmopmod, [783](#)
max_jit_classex_mop_wrap
 maxmopmod, [783](#)
max_jit_classex_setup
 maxwrapmod, [758](#)
max_jit_classex_standard_wrap
 maxwrapmod, [758](#)
max_jit_mop_adapt_matrix_all
 maxmopmod, [784](#)
max_jit_mop_assist
 maxmopmod, [784](#)
max_jit_mop_bang
 maxmopmod, [784](#)
max_jit_mop_clear
 maxmopmod, [784](#)
max_jit_mop_free
 maxmopmod, [785](#)
max_jit_mop_get_io_by_name
 maxmopmod, [785](#)
max_jit_mop_getinput
 maxmopmod, [785](#)
max_jit_mop_getoutput
 maxmopmod, [785](#)
max_jit_mop_getoutputmode
 maxmopmod, [786](#)

max_jit_mop_inputs
 maxmopmod, 786
max_jit_mop_jit_matrix
 maxmopmod, 786
max_jit_mop_matrix_args
 maxmopmod, 786
max_jit_mop_matrixout_new
 maxmopmod, 787
max_jit_mop_notify
 maxmopmod, 787
max_jit_mop_outputmatrix
 maxmopmod, 787
max_jit_mop_outputs
 maxmopmod, 787
max_jit_mop_setup
 maxmopmod, 788
max_jit_mop_setup_simple
 maxmopmod, 788
max_jit_mop_variable_addinputs
 maxmopmod, 788
max_jit_mop_variable_addoutputs
 maxmopmod, 789
max_jit_ob3d_assist
 ob3dmod, 799
max_jit_ob3d_attach
 ob3dmod, 800
max_jit_ob3d_detach
 ob3dmod, 800
max_jit_obex_adornment_get
 maxwrapmod, 758
max_jit_obex_attr_get
 maxwrapmod, 758
max_jit_obex_attr_set
 maxwrapmod, 759
max_jit_obex_dumpout
 maxwrapmod, 759
max_jit_obex_dumpout_get
 maxwrapmod, 759
max_jit_obex_dumpout_set
 maxwrapmod, 760
max_jit_obex_free
 maxwrapmod, 760
max_jit_obex_gimmeback
 maxwrapmod, 760
max_jit_obex_gimmeback_dumpout
 maxwrapmod, 760
max_jit_obex_inletnumber_get
 maxwrapmod, 760
max_jit_obex_inletnumber_set
 maxwrapmod, 761
max_jit_obex_jitob_get
 maxwrapmod, 761
max_jit_obex_jitob_set
 maxwrapmod, 761
max_jit_obex_new
 maxwrapmod, 761
max_jit_obex_proxy_new
 maxwrapmod, 762
max_ob3d_bang
 ob3dmod, 800
max_ob3d_notify
 ob3dmod, 800
MAXARG
 obj, 495
maxmopmod
 max_jit_classex_mop_mproc, 783
 max_jit_classex_mop_wrap, 783
 max_jit_mop_adapt_matrix_all, 784
 max_jit_mop_assist, 784
 max_jit_mop_bang, 784
 max_jit_mop_clear, 784
 max_jit_mop_free, 785
 max_jit_mop_get_io_by_name, 785
 max_jit_mop_getinput, 785
 max_jit_mop_getoutput, 785
 max_jit_mop_getoutputmode, 786
 max_jit_mop_inputs, 786
 max_jit_mop_jit_matrix, 786
 max_jit_mop_matrix_args, 786
 max_jit_mop_matrixout_new, 787
 max_jit_mop_notify, 787
 max_jit_mop_outputmatrix, 787
 max_jit_mop_outputs, 787
 max_jit_mop_setup, 788
 max_jit_mop_setup_simple, 788
 max_jit_mop_variable_addinputs, 788
 max_jit_mop_variable_addoutputs, 789
maxversion
 misc, 442
maxwrapmod
 max_addmethod_defer, 755
 max_addmethod_defer_low, 756
 max_addmethod_usurp, 756
 max_addmethod_usurp_low, 756
 max_jit_attr_args, 756
 max_jit_attr_args_offset, 756
 max_jit_attr_get, 757
 max_jit_attr_getdump, 757
 max_jit_attr_set, 757
 max_jit_classex_addattr, 757
 max_jit_classex_setup, 758
 max_jit_classex_standard_wrap, 758
 max_jit_obex_adornment_get, 758
 max_jit_obex_attr_get, 758
 max_jit_obex_attr_set, 759
 max_jit_obex_dumpout, 759
 max_jit_obex_dumpout_get, 759
 max_jit_obex_dumpout_set, 760

max_jit_obex_free, 760
 max_jit_obex_gimmeback, 760
 max_jit_obex_gimmeback_dumpout, 760
 max_jit_obex_inletnumber_get, 760
 max_jit_obex_inletnumber_set, 761
 max_jit_obex_jitob_get, 761
 max_jit_obex_jitob_set, 761
 max_jit_obex_new, 761
 max_jit_obex_proxy_new, 762
memory
 disposhandle, 428
 freebytes, 428
 freebytes16, 428
 getbytes, 429
 getbytes16, 429
 growhandle, 429
 MM_UNIFIED, 428
 newhandle, 429
 sysmem_copyptr, 430
 sysmem_freehandle, 430
 sysmem_freeptr, 430
 sysmem_handlesize, 430
 sysmem_lockhandle, 431
 sysmem_newhandle, 431
 sysmem_newhandleclear, 431
 sysmem_newptr, 432
 sysmem_newptrclear, 432
 sysmem_nullterminatehandle, 432
 sysmem_ptrandhand, 432
 sysmem_ptrbeforehand, 433
 sysmem_ptrsize, 433
 sysmem_resizehandle, 433
 sysmem_resizeptr, 434
 sysmem_resizeptrclear, 434
Memory Management, 426
Memory Module, 763
memorymod
 jit_copy_bytes, 763
 jit_disposeptr, 764
 jit_freebytes, 764
 jit_freemem, 764
 jit_getbytes, 764
 jit_handle_free, 765
 jit_handle_lock, 765
 jit_handle_new, 765
 jit_handle_size_get, 766
 jit_handle_size_set, 766
 jit_newptr, 766
MIN
 misc, 439
misc
 aaCancel, 440
 aaNo, 440
 aaYes, 440
BEGIN_USING_C_LINKAGE, 438
calcoffset, 438
CLIP, 438
e_max_errorcodes, 440
e_max_wind_advise_result, 440
error_subscribe, 440
error_sym, 440
error_unsubscribe, 441
globalsymbol_bind, 441
globalsymbol_dereference, 441
globalsymbol_reference, 441
globalsymbol_unbind, 442
InRange, 439
MAX, 439
MAX_ERR_DUPLICATE, 440
MAX_ERR_GENERIC, 440
MAX_ERR_INVALID_PTR, 440
MAX_ERR_NONE, 440
MAX_ERR_OUT_OF_MEM, 440
maxversion, 442
MIN, 439
object_obex_quickref, 443
post_sym, 443
quittask_install, 443
quittask_remove, 443
snprintf_zero, 443
strncat_zero, 444
strncpy_zero, 444
symbol_unique, 444
symbolarray_sort, 444
wind_advise, 445
wind_setcursor, 445
Miscellaneous, 435
Miscellaneous Utility Module, 713
MM_UNIFIED
 memory, 428
Monitors and Displays, 471
MOP Max Wrapper Module, 782
MOP Module, 767
mopmod
 jit_mop_free, 769
 jit_mop_getinput, 769
 jit_mop_getinputlist, 769
 jit_mop_getoutput, 769
 jit_mop_getoutputlist, 770
 jit_mop_input_nolink, 770
 jit_mop_io_free, 770
 jit_mop_io_getioproc, 771
 jit_mop_io_getmatrix, 771
 jit_mop_io_ioproc, 771
 jit_mop_io_matrix, 772
 jit_mop_io_new, 772
 jit_mop_io_newcopy, 772
 jit_mop_io_restrict_dim, 773

jit_mop_io_restrict_planecount, 773
jit_mop_io_restrict_type, 773
jit_mop_ioproc_copy_adapt, 774
jit_mop_ioproc_copy_trunc, 774
jit_mop_ioproc_copy_trunc_zero, 775
jit_mop_ioproc_tosym, 775
jit_mop_methodall, 776
jit_mop_new, 776
jit_mop_newcopy, 776
jit_mop_output_nolink, 776
jit_mop_single_planecount, 777
jit_mop_single_type, 777
Mouse and Keyboard, 474
MSP, 481
msp
 class_dspinit, 484
 class_dspinitjbox, 485
 dsp_add, 485
 dsp_addv, 485
 PI, 483
 PIOVERTWO, 483
 SYS_MAXBLKSIZE, 484
 SYS_MAXSIGS, 484
 sys_getblksize, 486
 sys_getdspobjdspstate, 486
 sys_getdspstate, 486
 sys_getmaxblksize, 486
 sys_gettsr, 486
 t_float, 484
 t_int, 484
 t_perfroutine, 484
 t_sample, 484
 t_vptr, 484
 TWOP, 483
 vptr, 484
 z_dsp_free, 486
 z_dsp_setup, 487
mutex
 systhread_mutex_free, 609
 systhread_mutex_lock, 609
 systhread_mutex_new, 610
 systhread_mutex_newlock, 610
 systhread_mutex_trylock, 610
 systhread_mutex_unlock, 611
Mutexes, 609
newhandle
 memory, 429
newinstance
 class_old, 263
newobject
 class_old, 263
newobject_fromdictionary
 obj, 496
 newobject_sprintf
 obj, 497
 OB3D Module, 790
 ob3d_auto_get
 ob3dmod, 800
 ob3d_dest_dim_get
 ob3dmod, 801
 ob3d_dest_dim_set
 ob3dmod, 801
 ob3d_dirty_get
 ob3dmod, 801
 ob3d_dirty_set
 ob3dmod, 801
 ob3d_enable_get
 ob3dmod, 801
 ob3d_jitob_get
 ob3dmod, 802
 ob3d_outlet_get
 ob3dmod, 802
 ob3d_render_ptr_get
 ob3dmod, 802
 ob3d_render_ptr_set
 ob3dmod, 802
 ob3d_ui_get
 ob3dmod, 803
 ob3dmod
 jit_gl_begincapture, 793
 jit_gl_bindtexture, 793
 jit_gl_drawinfo_active_textures, 793
 jit_gl_drawinfo_setup, 793
 jit_gl_endcapture, 793
 jit_gl_get_extensions, 794
 jit_gl_get_glu_version, 794
 jit_gl_get_renderer, 794
 jit_gl_get_vendor, 794
 jit_gl_get_version, 794
 jit_gl_is_extension_supported, 795
 jit_gl_is_min_version, 795
 jit_gl_report_error, 795
 jit_gl_texcoord1f, 795
 jit_gl_texcoord1fv, 796
 jit_gl_texcoord2f, 796
 jit_gl_texcoord2fv, 796
 jit_gl_texcoord3f, 796
 jit_gl_texcoord3fv, 797
 jit_gl_unbindtexture, 797
 jit_glchunk_copy, 797
 jit_glchunk_delete, 797
 jit_glchunk_grid_new, 797
 jit_glchunk_new, 798
 jit_ob3d_draw_chunk, 798
 jit_ob3d_free, 798
 jit_ob3d_new, 798

jit_ob3d_set_context, 799
 jit_ob3d_setup, 799
 max_jit_ob3d_assist, 799
 max_jit_ob3d_attach, 800
 max_jit_ob3d_detach, 800
 max_ob3d_bang, 800
 max_ob3d_notify, 800
 ob3d_auto_get, 800
 ob3d_dest_dim_get, 801
 ob3d_dest_dim_set, 801
 ob3d_dirty_get, 801
 ob3d_dirty_set, 801
 ob3d_enable_get, 801
 ob3d_jitob_get, 802
 ob3d_outlet_get, 802
 ob3d_render_ptr_get, 802
 ob3d_render_ptr_set, 802
 ob3d_ui_get, 803
**OBEX_UTIL_ATOM_GETTEXT_COMMA_-
 DELIM**
 atom, 359
OBEX_UTIL_ATOM_GETTEXT_DEFAULT
 atom, 358
**OBEX_UTIL_ATOM_GETTEXT_FORCE_-
 ZEROS**
 atom, 359
OBEX_UTIL_ATOM_GETTEXT_NUM_HI_RES
 atom, 359
**OBEX_UTIL_ATOM_GETTEXT_SYM_-
 FORCE_QUOTE**
 atom, 359
**OBEX_UTIL_ATOM_GETTEXT_SYM_NO_-
 QUOTE**
 atom, 359
**OBEX_UTIL_ATOM_GETTEXT_TRUNCATE_-
 ZEROS**
 atom, 358
obj
 classname_openhelp, 496
 classname_openquery, 496
 classname_openrefpage, 496
 MAXARG, 495
 newobject_fromdictionary, 496
 newobject_sprintf, 497
 object_alloc, 497
 object_attach, 498
 object_attach_byptr, 498
 object_attach_byptr_register, 499
 object_class, 499
 object_classname, 499
 object_classname_compare, 500
 object_detach, 500
 object_detach_byptr, 500
 object_dictionaryarg, 501
 object_findregistered, 501
 object_findregisteredbyptr, 501
 object_free, 502
 object_getmethod, 502
 object_getvalueof, 502
 object_method, 503
 object_method_char, 504
 object_method_char_array, 504
 object_method_double, 504
 object_method_double_array, 505
 object_method_float, 505
 object_method_float_array, 506
 object_method_format, 506
 object_method_long, 506
 object_method_long_array, 507
 object_method_obj, 507
 object_method_obj_array, 508
 object_method_parse, 508
 object_method_sym, 508
 object_method_sym_array, 509
 object_method_typed, 509
 object_method_typedfun, 510
 object_new, 510
 object_new_typed, 510
 object_notify, 511
 object_obex_dumpout, 511
 object_obex_lookup, 512
 object_obex_store, 512
 object_openhelp, 513
 object_openquery, 513
 object_openrefpage, 513
 object_register, 513
 object_setvalueof, 514
 object_unregister, 514
OBJ_FLAG_DATA
 datastore, 275
OBJ_FLAG_INHERITABLE
 datastore, 275
OBJ_FLAG_MEMORY
 datastore, 275
OBJ_FLAG_OBJ
 datastore, 275
OBJ_FLAG_REF
 datastore, 275
OBJ_FLAG_SILENT
 datastore, 275
OBJ_ATTR_ATOM
 attr, 215
OBJ_ATTR_ATOM_ARRAY
 attr, 215
OBJ_ATTR_CHAR
 attr, 216
OBJ_ATTR_CHAR_ARRAY
 attr, 216

OBJ_ATTR_DEFAULT
 attr, 216
OBJ_ATTR_DEFAULT_SAVE
 attr, 217
OBJ_ATTR_DOUBLE
 attr, 217
OBJ_ATTR_DOUBLE_ARRAY
 attr, 217
OBJ_ATTR_FLOAT
 attr, 217
OBJ_ATTR_FLOAT_ARRAY
 attr, 218
OBJ_ATTR_LONG
 attr, 218
OBJ_ATTR_LONG_ARRAY
 attr, 218
OBJ_ATTR_OBJ
 attr, 218
OBJ_ATTR_OBJ_ARRAY
 attr, 219
OBJ_ATTR_SAVE
 attr, 219
OBJ_ATTR_SYM
 attr, 219
OBJ_ATTR_SYM_ARRAY
 attr, 219
Object Module, 705
object_addattr
 attr, 239
object_alloc
 obj, 497
object_attach
 obj, 498
object_attach_byptr
 obj, 498
object_attach_byptr_register
 obj, 499
object_attr_get
 attr, 239
object_attr_get_rect
 attr, 240
object_attr_getchar_array
 attr, 240
object_attr_getcolor
 attr, 240
object_attr_getdouble_array
 attr, 241
object_attr_getdump
 attr, 241
object_attr_getfloat
 attr, 241
object_attr_getfloat_array
 attr, 242
object_attr_getjrgba
 attr, 242
object_attr_getlong
 attr, 242
object_attr_getlong_array
 attr, 243
object_attr_getpt
 attr, 243
object_attr_getsize
 attr, 243
object_attr_getsym
 attr, 244
object_attr_getsym_array
 attr, 244
object_attr_method
 attr, 244
object_attr_set_rect
 attr, 245
object_attr_setchar_array
 attr, 245
object_attr_setcolor
 attr, 245
object_attr_setdouble_array
 attr, 246
object_attr_setfloat
 attr, 246
object_attr_setfloat_array
 attr, 246
object_attr_setjrgba
 attr, 247
object_attr_setlong
 attr, 247
object_attr_setlong_array
 attr, 247
object_attr_setparse
 attr, 248
object_attr_setpt
 attr, 248
object_attr_setsize
 attr, 248
object_attr_setsym
 attr, 249
object_attr_setsym_array
 attr, 249
object_attr_setvalueof
 attr, 249
object_attr_usercanget
 attr, 250
object_attr_usercanset
 attr, 250
object_chuckattr
 attr, 250
object_class
 obj, 499
object_classname

obj, 499
 object_classname_compare
 obj, 500
 object_deleteattr
 attr, 251
 object_detach
 obj, 500
 object_detach_byptr
 obj, 500
 object_dictionaryarg
 obj, 501
 object_error
 console, 447
 object_error_obtrusive
 console, 448
 object_findregistered
 obj, 501
 object_findregisteredbyptr
 obj, 501
 object_free
 obj, 502
 object_getmethod
 obj, 502
 object_getvalueof
 obj, 502
 object_method
 obj, 503
 object_method_char
 obj, 504
 object_method_char_array
 obj, 504
 object_method_double
 obj, 504
 object_method_double_array
 obj, 505
 object_method_float
 obj, 505
 object_method_float_array
 obj, 506
 object_method_format
 obj, 506
 object_method_long
 obj, 506
 object_method_long_array
 obj, 507
 object_method_obj
 obj, 507
 object_method_obj_array
 obj, 508
 object_method_parse
 obj, 508
 object_method_sym
 obj, 508
 object_method_sym_array

 obj, 509
 object_method_typed
 obj, 509
 object_method_typedfun
 obj, 510
 object_new
 obj, 510
 object_new_parse
 attr, 251
 object_new_typed
 obj, 510
 object_notify
 obj, 511
 object_obex_dumpout
 obj, 511
 object_obex_lookup
 obj, 512
 object_obex_quickref
 misc, 443
 object_obex_store
 obj, 512
 object_openhelp
 obj, 513
 object_openquery
 obj, 513
 object_openrefpage
 obj, 513
 object_post
 console, 448
 object_register
 obj, 513
 object_setvalueof
 obj, 514
 object_unregister
 obj, 514
 object_warn
 console, 448
 objectmod
 jit_object_attach, 706
 jit_object_attr_get, 706
 jit_object_attr_usercanget, 707
 jit_object_attr_usercanset, 707
 jit_object_class, 707
 jit_object_classname, 707
 jit_object_classname_compare, 707
 jit_object_detach, 708
 jit_object_exportattrs, 708
 jit_object_exportsummary, 708
 jit_object_findregistered, 709
 jit_object_findregisteredbyptr, 709
 jit_object_free, 709
 jit_object_getmethod, 709
 jit_object_importattrs, 710
 jit_object_method, 710

jit_object_method_argsafe_get, 710
jit_object_method_typed, 710
jit_object_new, 711
jit_object_notify, 711
jit_object_register, 711
jit_object_unregister, 712
Objects, 492
Old-Style Classes, 259
open_dialog
 files, 395
open_promptset
 files, 396
Operator Vector Module, 804
opvecmod
 jit_op_vector_abs_float32, 818
 jit_op_vector_abs_float64, 819
 jit_op_vector_abs_long, 819
 jit_op_vector_absdiff_char, 819
 jit_op_vector_absdiff_float32, 819
 jit_op_vector_absdiff_float64, 820
 jit_op_vector_absdiff_long, 820
 jit_op_vector_acos_float32, 820
 jit_op_vector_acos_float64, 821
 jit_op_vector_acosh_float32, 821
 jit_op_vector_acosh_float64, 821
 jit_op_vector_add_char, 821
 jit_op_vector_add_float32, 822
 jit_op_vector_add_float64, 822
 jit_op_vector_add_long, 822
 jit_op_vector_adds_char, 823
 jit_op_vector_and_char, 823
 jit_op_vector_and_float32, 823
 jit_op_vector_and_float64, 823
 jit_op_vector_and_long, 824
 jit_op_vector_asin_float32, 824
 jit_op_vector_asin_float64, 824
 jit_op_vector_asinh_float32, 825
 jit_op_vector_asinh_float64, 825
 jit_op_vector_atan2_float32, 825
 jit_op_vector_atan2_float64, 825
 jit_op_vector_atan_float32, 826
 jit_op_vector_atan_float64, 826
 jit_op_vector_atanh_float32, 826
 jit_op_vector_atanh_float64, 827
 jit_op_vector_avg_char, 827
 jit_op_vector_avg_float32, 827
 jit_op_vector_avg_float64, 827
 jit_op_vector_avg_long, 828
 jit_op_vector_bitand_char, 828
 jit_op_vector_bitand_long, 828
 jit_op_vector_bitnot_char, 829
 jit_op_vector_bitnot_long, 829
 jit_op_vector_bitor_char, 829
 jit_op_vector_bitor_long, 829
 jit_op_vector_bitxor_char, 830
 jit_op_vector_bitxor_long, 830
 jit_op_vector_ceil_float32, 830
 jit_op_vector_ceil_float64, 831
 jit_op_vector_cos_float32, 831
 jit_op_vector_cos_float64, 831
 jit_op_vector_cosh_float32, 831
 jit_op_vector_cosh_float64, 832
 jit_op_vector_div_char, 832
 jit_op_vector_div_float32, 832
 jit_op_vector_div_float64, 833
 jit_op_vector_div_long, 833
 jit_op_vector_eq_char, 833
 jit_op_vector_eq_float32, 833
 jit_op_vector_eq_float64, 834
 jit_op_vector_eq_long, 834
 jit_op_vector_eqp_char, 834
 jit_op_vector_eqp_float32, 835
 jit_op_vector_eqp_float64, 835
 jit_op_vector_eqp_long, 835
 jit_op_vector_exp2_float32, 835
 jit_op_vector_exp2_float64, 836
 jit_op_vector_exp_float32, 836
 jit_op_vector_exp_float64, 836
 jit_op_vector_flipdiv_char, 837
 jit_op_vector_flipdiv_float32, 837
 jit_op_vector_flipdiv_float64, 837
 jit_op_vector_flipdiv_long, 837
 jit_op_vector_flipmod_char, 838
 jit_op_vector_flipmod_float32, 838
 jit_op_vector_flipmod_float64, 838
 jit_op_vector_flipmod_long, 839
 jit_op_vector_flippass_char, 839
 jit_op_vector_flippass_float32, 839
 jit_op_vector_flippass_float64, 839
 jit_op_vector_flippass_long, 840
 jit_op_vector_flipsub_char, 840
 jit_op_vector_flipsub_float32, 840
 jit_op_vector_flipsub_long, 841
 jit_op_vector_floor_float32, 841
 jit_op_vector_floor_float64, 841
 jit_op_vector_fold_float32, 841
 jit_op_vector_fold_float64, 842
 jit_op_vector_gt_char, 842
 jit_op_vector_gt_float32, 842
 jit_op_vector_gt_float64, 843
 jit_op_vector_gt_long, 843
 jit_op_vector_gte_char, 843
 jit_op_vector_gte_float32, 843
 jit_op_vector_gte_float64, 844
 jit_op_vector_gte_long, 844
 jit_op_vector_gtep_char, 844
 jit_op_vector_gtep_float32, 845
 jit_op_vector_gtep_float64, 845

jit_op_vector_gtep_long, 845
 jit_op_vector_gtp_char, 845
 jit_op_vector_gtp_float32, 846
 jit_op_vector_gtp_float64, 846
 jit_op_vector_gtp_long, 846
 jit_op_vector_hypot_float32, 847
 jit_op_vector_hypot_float64, 847
 jit_op_vector_log10_float32, 847
 jit_op_vector_log10_float64, 847
 jit_op_vector_log2_float32, 848
 jit_op_vector_log2_float64, 848
 jit_op_vector_log_float32, 848
 jit_op_vector_log_float64, 849
 jit_op_vector_lshift_char, 849
 jit_op_vector_lshift_long, 849
 jit_op_vector_lt_char, 849
 jit_op_vector_lt_float32, 850
 jit_op_vector_lt_float64, 850
 jit_op_vector_lt_long, 850
 jit_op_vector_lte_char, 851
 jit_op_vector_lte_float32, 851
 jit_op_vector_lte_float64, 851
 jit_op_vector_lte_long, 851
 jit_op_vector_ltep_char, 852
 jit_op_vector_ltep_float32, 852
 jit_op_vector_ltep_float64, 852
 jit_op_vector_ltep_long, 853
 jit_op_vector_ltp_char, 853
 jit_op_vector_ltp_float32, 853
 jit_op_vector_ltp_float64, 853
 jit_op_vector_ltp_long, 854
 jit_op_vector_max_char, 854
 jit_op_vector_max_float32, 854
 jit_op_vector_max_float64, 855
 jit_op_vector_max_long, 855
 jit_op_vector_min_char, 855
 jit_op_vector_min_float32, 855
 jit_op_vector_min_float64, 856
 jit_op_vector_min_long, 856
 jit_op_vector_mod_char, 856
 jit_op_vector_mod_float32, 857
 jit_op_vector_mod_float64, 857
 jit_op_vector_mod_long, 857
 jit_op_vector_mult_char, 857
 jit_op_vector_mult_float32, 858
 jit_op_vector_mult_float64, 858
 jit_op_vector_mult_long, 858
 jit_op_vector_neq_char, 859
 jit_op_vector_neq_float32, 859
 jit_op_vector_neq_float64, 859
 jit_op_vector_neq_long, 859
 jit_op_vector_neqp_char, 860
 jit_op_vector_neqp_float32, 860
 jit_op_vector_neqp_float64, 860
 jit_op_vector_neqp_long, 861
 jit_op_vector_not_char, 861
 jit_op_vector_not_float32, 861
 jit_op_vector_not_float64, 861
 jit_op_vector_not_long, 862
 jit_op_vector_or_char, 862
 jit_op_vector_or_float32, 862
 jit_op_vector_or_float64, 863
 jit_op_vector_or_long, 863
 jit_op_vector_pass_char, 863
 jit_op_vector_pass_float32, 863
 jit_op_vector_pass_float64, 864
 jit_op_vector_pass_long, 864
 jit_op_vector_pow_float32, 864
 jit_op_vector_pow_float64, 865
 jit_op_vector_round_float32, 865
 jit_op_vector_round_float64, 865
 jit_op_vector_rshift_char, 865
 jit_op_vector_rshift_long, 866
 jit_op_vector_sin_float32, 866
 jit_op_vector_sin_float64, 866
 jit_op_vector_sinh_float32, 867
 jit_op_vector_sinh_float64, 867
 jit_op_vector_sqrt_float32, 867
 jit_op_vector_sqrt_float64, 867
 jit_op_vector_sub_char, 868
 jit_op_vector_sub_float32, 868
 jit_op_vector_sub_float64, 868
 jit_op_vector_sub_long, 869
 jit_op_vector_subs_char, 869
 jit_op_vector_tan_float32, 869
 jit_op_vector_tan_float64, 869
 jit_op_vector_tanh_float32, 870
 jit_op_vector_tanh_float64, 870
 jit_op_vector_trunc_float32, 870
 jit_op_vector_trunc_float64, 871
 jit_op_vector_wrap_float32, 871
 jit_op_vector_wrap_float64, 871
 ouchstring
 console, 449
 outlet_anything
 inout, 269
 outlet_bang
 inout, 270
 outlet_float
 inout, 270
 outlet_int
 inout, 270
 outlet_list
 inout, 270
 outlet_new
 inout, 271
 Parallel Utility Module, 778

parallelutilmod
 jit_parallel_ndim_calc, 778
 jit_parallel_ndim_simplecalc1, 778
 jit_parallel_ndim_simplecalc2, 779
 jit_parallel_ndim_simplecalc3, 779
 jit_parallel_ndim_simplecalc4, 780

Patcher, 516

patcher
 PI_DEEP, 517
 PI_REQUIREFIRSTIN, 517
 PI_WANTBOX, 517
 t_box, 517
 t_patcher, 517

patcherview_findpatcherview
 jpatcherview, 560

patcherview_get_jgraphics
 jpatcherview, 560

patcherview_get_locked
 jpatcherview, 560

patcherview_get_nextview
 jpatcherview, 560

patcherview_get_patcher
 jpatcherview, 561

patcherview_get_presentation
 jpatcherview, 561

patcherview_get_rect
 jpatcherview, 561

patcherview_get_topview
 jpatcherview, 561

patcherview_get_visible
 jpatcherview, 562

patcherview_get_zoomfactor
 jpatcherview, 562

patcherview_set_locked
 jpatcherview, 562

patcherview_set_presentation
 jpatcherview, 562

patcherview_set_rect
 jpatcherview, 563

patcherview_set_visible
 jpatcherview, 563

patcherview_set_zoomfactor
 jpatcherview, 563

PATH_FILEINFO_ALIAS
 files, 391

PATH_FILEINFO_FOLDER
 files, 391

PATH_FILEINFO_PACKAGE
 files, 391

PATH_FOLDER_SNIFF
 files, 391

PATH_READ_PERM
 files, 391

PATH_REPORTPACKAGEASFOLDER
 files, 391

PATH_RW_PERM
 files, 391

PATH_STYLE_COLON
 files, 391

PATH_STYLE_MAX
 files, 391

PATH_STYLE_NATIVE
 files, 391

PATH_STYLE_NATIVE_WIN
 files, 391

PATH_STYLE_SLASH
 files, 391

PATH_TYPE_ABSOLUTE
 files, 392

PATH_TYPE_BOOT
 files, 392

PATH_TYPE_C74
 files, 392

PATH_TYPE_IGNORE
 files, 392

PATH_TYPE_PATH
 files, 392

PATH_TYPE_RELATIVE
 files, 392

PATH_WRITE_PERM
 files, 391

path_closefolder
 files, 396

path_createsysfile
 files, 396

path_fileinfo
 files, 396

path_foldernextfile
 files, 397

path_from pathname
 files, 397

path_getapppath
 files, 397

path_getdefault
 files, 398

path_getfilemoddate
 files, 398

path_getmoddate
 files, 398

path_nameconform
 files, 398

path_openfolder
 files, 399

path_opensysfile
 files, 399

path_resolvefile
 files, 399

path_setdefault

files, 400
 path_top pathname
 files, 400
 path_topotentialname
 files, 400
 PFFT, 490
 PI
 msp, 483
 PI_DEEP
 patcher, 517
 PI_REQUIREFIRSTIN
 patcher, 517
 PI_WANTBOX
 patcher, 517
 PIOVERTWO
 msp, 483
 Poly, 491
 Popup Menus, 667
 post
 console, 449
 post_sym
 misc, 443
 postargs
 atom, 373
 postatom
 console, 450
 postdictionary
 dictionary, 314
 preset_int
 presets, 464
 preset_set
 presets, 464
 preset_store
 presets, 464
 Presets, 463
 presets
 preset_int, 464
 preset_set, 464
 preset_store, 464
 proxy_getinlet
 inout, 271
 proxy_new
 inout, 272
 qelem_free
 qelems, 576
 qelem_front
 qelems, 576
 qelem_new
 qelems, 576
 qelem_set
 qelems, 576
 qelem_unset
 qelems, 577
 Qelems, 575
 qelems
 qelem_free, 576
 qelem_front, 576
 qelem_new, 576
 qelem_set, 576
 qelem_unset, 577
 qtcodecmod
 jit/qt_codec_acodec2sym, 873
 jit/qt_codec_getcodeclist_audio, 873
 jit/qt_codec_getcodeclist_audio_raw, 873
 jit/qt_codec_getcodeclist_gfx, 873
 jit/qt_codec_getcodeclist_gfx_raw, 874
 jit/qt_codec_getcodeclist_video, 874
 jit/qt_codec_getcodeclist_video_raw, 874
 jit/qt_codec_qual2sym, 874
 jit/qt_codec_sym2acodec, 875
 jit/qt_codec_sym2qual, 875
 jit/qt_codec_sym2type, 875
 jit/qt_codec_sym2type_valid, 875
 jit/qt_codec_type2sym, 876
 jit/qt_codec_type2sym_valid, 876
 qtmoviemod
 jit/qt_movie_matrix_calc, 877
 jit/qt_movie_matrix_to_image, 877
 jit/qt_movie_new, 878
 jit/qt_movie_read_typed, 878
 qtrecordmod
 jit/qt_record_matrix_calc, 880
 jit/qt_record_new, 880
 qtutilsmod
 jit/coerce_matrix_pixmap, 882
 jit/gworld_can_coerce_matrix, 882
 jit/gworld_clear, 882
 jit/gworld_matrix_equal_dim, 883
 jit/qt_utils_moviedataref_create, 883
 jit/qt_utils_moviefile_close, 883
 jit/qt_utils_moviefile_create, 883
 jit/qt_utils_str2type, 884
 jit/qt_utils_tempfile, 884
 jit/qt_utils_tempmoviefile_create, 884
 jit/qt_utils_trackmedia_add, 885
 jit/qt_utils_trackmedia_dispose, 885
 jit/qt_utils_trackmedia_get, 885
 jit/qt_utils_trackname_get, 886
 jit/qt_utils_trackname_set, 886
 jit/qt_utils_tracktype_get, 886
 jit/qt_utils_tracktypecode_get, 886
 jit/qt_utils_type2str, 887
 Quick Map, 346
 quickmap
 quickmap_add, 346
 quickmap_drop, 347
 quickmap_lookup_key1, 347

quickmap_lookup_key2, 347
quickmap_new, 347
quickmap_READONLY, 348
quickmap_add
 quickmap, 346
quickmap_drop
 quickmap, 347
quickmap_LOOKUP_KEY1
 quickmap, 347
quickmap_LOOKUP_KEY2
 quickmap, 347
quickmap_new
 quickmap, 347
quickmap_READONLY
 quickmap, 348
QuickTime Codec Module, 872
QuickTime Utilities Module, 881
quittask_install
 misc, 443
quittask_remove
 misc, 443

readatom
 binbuf, 380
readtohandle
 loading_max_files, 469

saveas_dialog
 files, 401
saveas_promptset
 files, 401
saveasdialog_extended
 files, 402
Scalable Vector Graphics, 638
schedule
 threading, 600
schedule_delay
 threading, 600
scheduler_fromobject
 clocks, 571
scheduler_get
 clocks, 571
scheduler_gettime
 clocks, 571
scheduler_new
 clocks, 571
scheduler_run
 clocks, 571
scheduler_set
 clocks, 572
scheduler_settime
 clocks, 572
scheduler_shift
 clocks, 572

serialno
 evnum, 466
setclock_delay
 clocks, 573
setclock_fdelay
 clocks, 573
setclock_gettime
 clocks, 573
setclock_gettime
 clocks, 573
setclock_unset
 clocks, 574
setup
 class_old, 263
snprintf_zero
 misc, 443
STATIC_ATTR_ATOM
 attr, 220
STATIC_ATTR_ATOM_ARRAY
 attr, 220
STATIC_ATTR_CHAR
 attr, 220
STATIC_ATTR_CHAR_ARRAY
 attr, 220
STATIC_ATTR_DOUBLE
 attr, 221
STATIC_ATTR_DOUBLE_ARRAY
 attr, 221
STATIC_ATTR_FLOAT
 attr, 221
STATIC_ATTR_FLOAT_ARRAY
 attr, 221
STATIC_ATTR_LONG
 attr, 222
STATIC_ATTR_LONG_ARRAY
 attr, 222
STATIC_ATTR_OBJ
 attr, 222
STATIC_ATTR_OBJ_ARRAY
 attr, 222
STATIC_ATTR_SYM
 attr, 223
STATIC_ATTR_SYM_ARRAY
 attr, 223
string
 string_getptr, 349
 string_new, 349
String Object, 349
string_getptr
 string, 349
string_new
 string, 349
stringload
 loading_max_files, 469

strncat_zero
 misc, 444
 strncpy_zero
 misc, 444
 STRUCT_ATTR_ATOM
 attr, 223
 STRUCT_ATTR_ATOM_ARRAY
 attr, 224
 STRUCT_ATTR_ATOM_VARSIZE
 attr, 224
 STRUCT_ATTR_CHAR
 attr, 224
 STRUCT_ATTR_CHAR_ARRAY
 attr, 225
 STRUCT_ATTR_CHAR_VARSIZE
 attr, 225
 STRUCT_ATTR_DOUBLE
 attr, 225
 STRUCT_ATTR_DOUBLE_ARRAY
 attr, 226
 STRUCT_ATTR_DOUBLE_VARSIZE
 attr, 226
 STRUCT_ATTR_FLOAT
 attr, 226
 STRUCT_ATTR_FLOAT_ARRAY
 attr, 227
 STRUCT_ATTR_FLOAT_VARSIZE
 attr, 227
 STRUCT_ATTR_LONG
 attr, 227
 STRUCT_ATTR_LONG_ARRAY
 attr, 228
 STRUCT_ATTR_LONG_VARSIZE
 attr, 228
 STRUCT_ATTR_OBJ
 attr, 228
 STRUCT_ATTR_OBJ_ARRAY
 attr, 229
 STRUCT_ATTR_OBJ_VARSIZE
 attr, 229
 STRUCT_ATTR_SYM
 attr, 229
 STRUCT_ATTR_SYM_ARRAY
 attr, 230
 STRUCT_ATTR_SYM_VARSIZE
 attr, 230
 swapf32
 utilitymod, 715
 swapf64
 utilitymod, 715
 symbol
 gensym, 383
 Symbol Object, 351
 symbol_unique

misc, 444
 symbolarray_sort
 misc, 444
 Symbols, 382
 symobject
 symobject_linklist_match, 351
 symobject_new, 352
 symobject_linklist_match
 symobject, 351
 symobject_new
 symobject, 352
 SYS_MAXBLKSIZE
 msp, 484
 SYS_MAXSIGS
 msp, 484
 sys_getblksize
 msp, 486
 sys_getdspobjdspstate
 msp, 486
 sys_getdspstate
 msp, 486
 sys_getmaxblksize
 msp, 486
 sys_getsr
 msp, 486
 SYSDATEFORMAT_FLAGS_LONG
 systime, 579
 SYSDATEFORMAT_FLAGS_MEDIUM
 systime, 579
 SYSDATEFORMAT_FLAGS_SHORT
 systime, 579
 sysdateformat_formatdatetime
 systime, 579
 sysdateformat_strftimetodatetime
 systime, 579
 SYSFILE_ATMARK
 files, 392
 SYSFILE_FROMLEOF
 files, 392
 SYSFILE_FROMMARK
 files, 392
 SYSFILE_FROMSTART
 files, 392
 sysfile_close
 files, 403
 sysfile_geteof
 files, 403
 sysfile_getpos
 files, 403
 sysfile_openhandle
 files, 403
 sysfile_openptrsize
 files, 404
 sysfile_read

files, 404
sysfile_readtextfile
 files, 404
sysfile_readtohandle
 files, 405
sysfile_readtoptr
 files, 405
sysfile_seteof
 files, 405
sysfile_setpos
 files, 406
sysfile_spoolcopy
 files, 406
sysfile_write
 files, 406
sysfile_writetextfile
 files, 407
sysmem_copyptr
 memory, 430
sysmem_freehandle
 memory, 430
sysmem_freeptr
 memory, 430
sysmem_handlesize
 memory, 430
sysmem_lockhandle
 memory, 431
sysmem_newhandle
 memory, 431
sysmem_newhandleclear
 memory, 431
sysmem_newptr
 memory, 432
sysmem_newptrclear
 memory, 432
sysmem_nullterminatehandle
 memory, 432
sysmem_ptrandhand
 memory, 432
sysmem_ptrbeforehand
 memory, 433
sysmem_ptrsize
 memory, 433
sysmem_resizehandle
 memory, 433
sysmem_resizeptr
 memory, 434
sysmem_resizeptrclear
 memory, 434
SYSTHREAD_MUTEX_ERRORCHECK
 threading, 598
SYSTHREAD_MUTEX_NORMAL
 threading, 598
SYSTHREAD_MUTEX_RECURSIVE
 threading, 598
systhread_create
 threading, 601
systhread_exit
 threading, 602
systhread_getpriority
 threading, 602
systhread_ismainthread
 threading, 602
systhread_istimerthread
 threading, 602
systhread_join
 threading, 602
systhread_mutex_free
 mutex, 609
systhread_mutex_lock
 mutex, 609
systhread_mutex_new
 mutex, 610
systhread_mutex_newlock
 mutex, 610
systhread_mutex_trylock
 mutex, 610
systhread_mutex_unlock
 mutex, 611
systhread_self
 threading, 603
systhread_setpriority
 threading, 603
systhread_sleep
 threading, 603
systhread_terminate
 threading, 603
systime
 e_max_dateflags, 579
 SYSDATEFORMAT_FLAGS_LONG, 579
 SYSDATEFORMAT_FLAGS_MEDIUM, 579
 SYSDATEFORMAT_FLAGS_SHORT, 579
 sysdateformat_formatdatetime, 579
 sysdateformat_strtimetodatetime, 579
 systime_datetime, 579
 systime_datetoseconds, 580
 systime_ms, 580
 systime_seconds, 580
 systime_secondstodate, 580
 systime_ticks, 580
 Systime API, 578
 systime_datetime
 systime, 579
 systime_datetoseconds
 systime, 580
 systime_ms
 systime, 580
 systime_seconds

systime, 580
 systime_secondstodate
 systime, 580
 systime_ticks
 systime, 580
 systimer_gettime
 clocks, 574

 t_atom, 890
 t_atomarray, 891
 t_atombuf, 892
 t_attr, 893
 t_box
 patcher, 517
 t_buffer, 894
 t_celldesc, 897
 t_charset_converter, 898
 t_class, 899
 t_cmpfn
 datastore, 274
 t_database
 database, 285
 t_datetime, 900
 t_db_result
 database, 285
 t_db_view
 database, 285
 t_dictionary, 901
 t_dictionary_entry, 902
 t_expr, 903
 t_filehandle
 files, 390
 t_fileinfo, 904
 t_float
 msp, 484
 t_funbuff, 905
 t_hashtab, 907
 t_hashtab_entry, 908
 t_indexmap, 909
 t_indexmap_entry, 910
 t_int
 msp, 484
 t_itm
 time, 594
 t_jbox, 911
 t_jboxdrawparams, 912
 t_jcolumn, 913
 t_jdataview, 917
 t_jgraphics_fileformat
 jgraphics, 618
 t_jgraphics_font_extents, 921
 t_jgraphics_font_slant
 jfont, 641
 t_jgraphics_font_weight

 jfont, 641
 t_jgraphics_format
 jgraphics, 618
 t_jgraphics_text_justification
 jgraphics, 619
 t_jgraphics_textlayout_flags
 textlayout, 664
 t_jit_attr, 922
 t_jit_attr_filter_clip, 924
 t_jit_attr_filter_proc, 926
 t_jit_attr_offset, 927
 t_jit_attr_offset_array, 929
 t_jit_attribute, 931
 t_jit_gl_drawinfo, 933
 t_jit_glchunk, 934
 t_jit_matrix_info, 936
 t_jit_mop, 938
 t_jit_mop_io, 940
 t_jit_op_info, 942
 t_jmatrix, 943
 t_jmouse_cursortype
 jmouse, 475
 t_jrgb, 944
 t_jrgba, 945
 t_line_3d, 946
 t_linklist, 947
 t_llelem, 948
 t_matrix_conv_info, 949
 t_max_err
 datatypes, 354
 t_messlist, 950
 t_modifiers
 jmouse, 476
 t_object, 951
 t_patch
 patcher, 517
 t_path, 952
 t_pathlink, 953
 t_perfroutine
 msp, 484
 t_pfftpub, 954
 t_privatesortrec, 955
 t_pt, 957
 t_pxjbox, 958
 t_pxobject, 960
 t_quickmap, 961
 t_rect, 962
 t_sample
 msp, 484
 t_signal, 963
 t_size, 964
 t_stack_splat, 965
 t_string, 966
 t_symbol, 967

t_symobject, 968
t_timeobject
 time, 594
t_tinyobject, 969
t_vptr
 msp, 484
t_wind_mouse_info, 970
t_zll, 971
Table Access, 460
table_dirty
 tables, 460
table_get
 tables, 460
tables
 table_dirty, 460
 table_get, 460
Text Editor Windows, 462
TEXT_ENCODING_USE_FILE
 files, 392
TEXT_LB_MAC
 files, 392
TEXT_LB_NATIVE
 files, 392
TEXT_LB_PC
 files, 392
TEXT_LB_UNIX
 files, 392
TEXT_NULL_TERMINATE
 files, 392
TextField, 653
textfield
 textfield_get_autoscroll, 655
 textfield_get_bgcolor, 655
 textfield_get_editonclick, 655
 textfield_get_emptytext, 655
 textfield_get_noactivate, 656
 textfield_get_owner, 656
 textfield_get_READONLY, 656
 textfield_get_selectallonedit, 656
 textfield_get_textcolor, 656
 textfield_get_textmargins, 657
 textfield_get_underline, 657
 textfield_get_useellipsis, 657
 textfield_get_wantsreturn, 657
 textfield_get_wantstab, 658
 textfield_get_wordwrap, 658
 textfield_set_autoscroll, 658
 textfield_set_bgcolor, 658
 textfield_set_editonclick, 659
 textfield_set_emptytext, 659
 textfield_set_noactivate, 659
 textfield_set_READONLY, 659
 textfield_set_selectallonedit, 660
 textfield_set_textcolor, 660
 textfield_set_textmargins, 660
 textfield_set_underline, 660
 textfield_set_useellipsis, 661
 textfield_set_wantsreturn, 661
 textfield_set_wantstab, 661
 textfield_set_wordwrap, 661
textfield_get_autoscroll
 textfield, 655
textfield_get_bgcolor
 textfield, 655
textfield_get_editonclick
 textfield, 655
textfield_get_emptytext
 textfield, 655
textfield_get_noactivate
 textfield, 656
textfield_get_owner
 textfield, 656
textfield_get_READONLY
 textfield, 656
textfield_get_selectallonedit
 textfield, 656
textfield_get_textcolor
 textfield, 656
textfield_get_textmargins
 textfield, 657
textfield_get_underline
 textfield, 657
textfield_get_useellipsis
 textfield, 657
textfield_get_wantsreturn
 textfield, 657
textfield_get_wantstab
 textfield, 658
textfield_get_wordwrap
 textfield, 658
textfield_set_autoscroll
 textfield, 658
textfield_set_bgcolor
 textfield, 658
textfield_set_editonclick
 textfield, 659
textfield_set_emptytext
 textfield, 659
textfield_set_noactivate
 textfield, 659
textfield_set_READONLY
 textfield, 659
textfield_set_selectallonedit
 textfield, 660
textfield_set_textcolor
 textfield, 660
textfield_set_textmargins
 textfield, 660

textfield_set_underline
 textfield, 660

textfield_set_useellipsis
 textfield, 661

textfield_set_wantsreturn
 textfield, 661

textfield_set_wantstab
 textfield, 661

textfield_set_wordwrap
 textfield, 661

TextLayout, 663

textlayout
 JGRAPHICS_TEXTLAYOUT_NOWRAP, 664

 JGRAPHICS_TEXTLAYOUT_ - USEELLIPSIS, 664

 jtextlayout_create, 664

 jtextlayout_destroy, 664

 jtextlayout_draw, 664

 jtextlayout_getchar, 664

 jtextlayout_getcharbox, 665

 jtextlayout_getnumchars, 665

 jtextlayout_measure, 665

 jtextlayout_set, 666

 jtextlayout_settextcolor, 666

 jtextlayout_withbgcolor, 666

 t_jgraphics_textlayout_flags, 664

threading
 ATOMIC_DECREMENT, 598

 ATOMIC_INCREMENT, 598

 defer, 598

 defer_low, 599

 e_max_systhread_mutex_flags, 598

 isr, 600

 schedule, 600

 schedule_delay, 600

 SYSTHREAD_MUTEX_ERRORCHECK, 598

 SYSTHREAD_MUTEX_NORMAL, 598

 SYSTHREAD_MUTEX_RECURSIVE, 598

 systhread_create, 601

 systhread_exit, 602

 systhread_getpriority, 602

 systhread_ismainthread, 602

 systhread_istimerthread, 602

 systhread_join, 602

 systhread_self, 603

 systhread_setpriority, 603

 systhread_sleep, 603

 systhread_terminate, 603

Threads, 596

time
 class_time_addattr, 585

 itm_barbeatunitstoticks, 585

 itm_dereference, 585

 itm_dump, 585

 itm_getglobal, 586

 itm_getname, 586

 itm_getnamed, 586

 itm_getresolution, 586

 itm_getstate, 587

 itm_getticks, 587

 itm_gettime, 587

 itm_gettimesignature, 587

 itm_isunitfixed, 588

 itm_mstosamps, 588

 itm_mstoticks, 588

 itm_pause, 588

 itm_reference, 589

 itm_resume, 589

 itm_sampstoms, 589

 itm_setresolution, 589

 itm_settimesignature, 590

 itm_ticksbarbeatunits, 590

 itm_tickstoms, 590

 t_itm, 594

 t_timeobject, 594

TIME_FLAGS_BBUSOURCE, 585

TIME_FLAGS_CHECKSCHEDULE, 584

TIME_FLAGS_EVENTLIST, 584

TIME_FLAGS_FIXED, 584

TIME_FLAGS_FIXEDONLY, 584

TIME_FLAGS_LISTENTICKS, 585

TIME_FLAGS_LOCATION, 584

TIME_FLAGS_LOOKAHEAD, 584

TIME_FLAGS_NOUNITS, 585

TIME_FLAGS_PERMANENT, 584

TIME_FLAGS_POSITIVE, 585

TIME_FLAGS_TICKSONLY, 584

TIME_FLAGS_TRANSPORT, 584

TIME_FLAGS_USECLOCK, 584

TIME_FLAGS_USEQELEM, 584

time_calcquantize, 590

time_getitm, 591

time_getms, 591

time_getnamed, 591

time_getphase, 591

time_getticks, 592

time_isfixedunit, 592

time_listen, 592

time_new, 592

time_now, 593

time_schedule, 593

time_schedule_limit, 593

time_setclock, 593

time_setvalue, 594

time_stop, 594

time_tick, 594

TIME_FLAGS_BBUSOURCE
 time, 585
TIME_FLAGS_CHECKSCHEDULE
 time, 584
TIME_FLAGS_EVENTLIST
 time, 584
TIME_FLAGS_FIXED
 time, 584
TIME_FLAGS_FIXEDONLY
 time, 584
TIME_FLAGS_LISTENTICKS
 time, 585
TIME_FLAGS_LOCATION
 time, 584
TIME_FLAGS_LOOKAHEAD
 time, 584
TIME_FLAGS_NOUNITS
 time, 585
TIME_FLAGS_PERMANENT
 time, 584
TIME_FLAGS_POSITIVE
 time, 585
TIME_FLAGS_TICKSONLY
 time, 584
TIME_FLAGS_TRANSPORT
 time, 584
TIME_FLAGS_USECLOCK
 time, 584
TIME_FLAGS_USEQELEM
 time, 584
time_calcquantize
 time, 590
time_getitm
 time, 591
time_getms
 time, 591
time_getnamed
 time, 591
time_getphase
 time, 591
time_getticks
 time, 592
time_isfixedunit
 time, 592
time_listen
 time, 592
time_new
 time, 592
time_now
 time, 593
time_schedule
 time, 593
time_schedule_limit
 time, 593
time_setclock
 time, 593
time_setvalue
 time, 594
time_stop
 time, 594
time_tick
 time, 594
Timing, 564
TWOPI
 msp, 483
typedmess
 class_old, 264
Unicode, 676
unicode
 charset_convert, 677
 charset_unicodetoutf8, 677
 charset_utf8_count, 678
 charset_utf8_offset, 678
 charset_utf8tounicode, 678
User Interface, 612
utilitymod
 jit_err_from_max_err, 713
 jit_error_code, 714
 jit_error_sym, 714
 jit_global_critical_enter, 714
 jit_global_critical_exit, 714
 jit_post_sym, 714
 jit_rand, 714
 jit_rand_setseed, 715
 swapf32, 715
 swapf64, 715
vptr
 msp, 484
wind_advise
 misc, 445
wind_setcursor
 misc, 445
Windows, 473
word, 972
z_dsp_free
 msp, 486
z_dsp_setup
 msp, 487
zgetfn
 class_old, 264