

Max API

7.1.0

Generated by Doxygen 1.8.10

Tue Dec 15 2015 09:20:55

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.

Max 6 introduced support for passing structured data with the [Dictionary Passing API](#).

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.

Information on updating MSP objects from Max 5 or earlier for 64-bit audio (introduced in Max 6) is located in

Appendix: Updating Externals for Max 6.

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 "externals" at the top level of the SDK package. This is located a couple of folders up from the project and source file.

Place the SDK package in your "Packages" folder in order to load and test the externals you build in Max itself.

2.2 Build Script

We have provided a basic script that will build all of the projects in the SDK at once. This script is written using the Ruby language. A Ruby interpreter is standard on the Mac. On windows you can download and install Ruby (1.9.3 recommended) from <http://rubyinstaller.org/> .

Run from Terminal.app (Mac) or a Command Prompt (Windows) by cd'ing into the examples directory, and then running:

```
ruby build.rb
```

2.3 Mac

Max external objects for the Mac are Mach-O bundles (folders that appear to be files) whose filenames must end with the .mxo extension. The example projects have been tested using Xcode 6.2. Xcode is available through the Apple Mac Store.

After installing Xcode, if you wish to run the aforementioned Ruby script, you will also need to install the Command Line Tools. This is done via the menu item: Xcode > Open Developer Tool > More Development Tools...

2.3.1 XCode Project Setup

The example projects are set up to have Development and Deployment build configurations. The Development configuration does not optimize and ensures debugging symbols are present. The Deployment configuration creates a universal binary and performs optimization.

The files required for the projects are included in the project folders with the exception 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.

2.3.2 Linking and Frameworks

External objects use dynamic linking to access the API functions provided by the Max application. When an object is loaded, calls to functions inside the application are resolved by the operating system to the correct memory address. Due to the fact that "Max" could exist as an application, a standalone you create, or a library inside another application, an object's Xcode project does not link directly to a framework, library, or application. Instead, the list of permitted symbols is provided to the linker. This list is defined in the aforementioned maxmspsdk.xcconfig file.

Audio objects will link against MaxAudioAPI.framework and Jitter objects link against JitterAPI.framework. Alternatively, you could also provide linker flags as we have provided for Max itself. 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.

2.4 Windows

Max external objects for Windows are Dynamic Link Libraries (DLLs) whose filenames must end with the .mxe extension (for 32-bit builds) or .mxe64 (for 64-bit builds). These DLLs will export a single function called "ext_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\. External objects that use Jitter functionality will import functions from the import library "jitlib.lib" which is located in c74support\jit-includes\.

2.4.1 Compiling with Visual Studio

The example projects are in Visual C++ 2013 format (vcxproj files). A free version of Visual C++ can be obtained from Microsoft at <http://www.microsoft.com/express/>. You will want to choose "Visual Studio Express 2013 for Windows Desktop".

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 for historical and backward compatibility reasons. It is recommended that you link with Microsoft's standard C runtime library rather than the Max C runtime library. 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. To use the Microsoft C Runtime define the C preprocessor macro MAXAPI_USE_MSCRT before including ext.h.

2.5 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.

2.5.1 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.

2.5.2 Windows

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

2.6 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.

2.7 Configuration

As the Max API evolves, the use of a number of older legacy functions are discouraged. Use of said functions will issue a 'deprecation' warning when you try to compile the code. To disable these deprecation warnings you can define the preprocessor symbol C74_NO_DEPRECATED in the target preprocessor section of your IDE.

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 ext_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 `ext_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 ext_main()

void ext_main(void *r)
{
    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;
}
```

In order for Max to call the `ext_main()` function on your compiled external, that function must be "exported" or made public. This is accomplished by using the `C74_EXPORT` macro in the prototype of the `ext_main()` function, which is provided for you automatically in the "ext.h" header file.

`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 your 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_addmethod\(\)](#) 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 outlet. 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_addmethod()` 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-prority 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.

- 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;
```

- 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!");
}
```

- 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, sizeof(
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_dsp64, "dsp64",
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 `dsp64` method specifies the signal processing function your object defines along with its arguments. Your object's `dsp64` 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 `dsp64` method is declared as follows:

```
void mydspobject_dsp64(t_mydspobject **x, t_object *dsp64, short *count, double samplerate, long maxvectorsize, long flags);
```

To add an entry to the DSP chain, your `dsp64` method uses the "dsp_add64" method of the DSP chain. The `dsp_add64` method is passed an a pointer to your object, a pointer to a `perform64` routine that calculates the samples, an optional flag which may alter behavior, and a generic pointer which will be passed on to your `perform` routine.

```
object_method(dsp64, gensym("dsp_add64"), x, mydspobject_perform64, 0, NULL);
```

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.

Here is a `perform` routine that adds a constant of 1 to any incoming signal:

```
void mydspobject_perform64(t_mydspobject **x, t_object *dsp64, double **ins, long numins, double **outs, long numouts, long sampleframes, long flags, void *userparam)
{
    double    *in = ins[0];      // first inlet
    double    *out = outs[0];    // first outlet
    int       n = sampleframes; // vector size
    t_double  value;

    while (n--) {              // perform calculation on all samples
        value = *in++;
        *out++ = value + 1.0;
    }
}
```

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;
    double      f_sample;
} t_myfilter;
```

Then, in our dsp method, 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_dsp64(t_myfilter **x, t_object *dsp64, short *count, double samplerate, long maxvectorsize, long flags)
{
    object_method(dsp64, gensym("dsp_add64"), x, myfilter_perform64, 0, NULL);
    x->f_sample = 0.0;
}
```

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.

```
void myfilter_perform64(t_myfilter **x, t_object *dsp64, double **ins, long numins, double **outs,
                        long numouts, long sampleframes, long flags, void *userparam)
{
    double    *in = ins[0];           // first inlet
    double    *out = outs[0];         // first outlet
    int       n = sampleframes;     // vector size
    double   samp = x->f_sample;    // read from internal state
    double   val;

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

9.2 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. 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_add64`' altogether. The MSP signal operator objects (such as `+~` and `*~`) use this 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_add64`.

```
if (count[1]) // signal connected to second inlet
    object_method(dsp64, gensym("dsp_add64"), x, mydspobject_twosigperform64, 0, NULL);
else
    object_method(dsp64, gensym("dsp_add64"), x, mydspobject_scalarperform64, 0, NULL);
```

The second option is to store the value of the `count` array for a particular signal in your object's struct. Then the perform method can make the decision whether to use the signal value or a scalar value that has been stored by 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 stored in a struct member named `m_count`:

```
x->m_count = count[1];
object_method(dsp64, gensym("dsp_add64"), x, mydspobject_perform64, 0, NULL);
```

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

```
void mydspobject_perform64(t_mydspobject *x, t_object *dsp64, double **ins, long numins, double **
                           outs, long numouts, long sampleframes, long flags, void *userparam)
{
    t_mydspobject *x = (t_mydspobject *)w[1];
    int connected = x->m_count;
    double *in = ins[0];
    double *in2 = ins[1];
    double *out = outs[0];
    int n = sampleframes;
    double in2value;

    // get scalar sample or use signal depending on whether signal is connected
    in2value = connected ? *in2 : x->m_scalarvalue;

    // do calculation here
    // ...
}
```

9.3 Working with Buffer Objects

To access a named `buffer~` object for either reading or writing sample values, refer to the [Buffers](#) reference.

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_obj;
    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_obj;
    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 not 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 when 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 (`ext_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 to it:

```
c = class_new("uisimp", (method)uisimp_new, (method)uisimp_free, sizeof(t_uisimp), 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,       "mouseshift",
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. `mouseshift` 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 `mouseshift` 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 / mouseshift / 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 your 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 mousdrag method that performs this test and redraws the object if the "mouse inside" condition has changed from its previous state. The mousdrag 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_mousdrag(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 implemented mouseenter, 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 the mouseenter message is received, and cleared when the mouseleave 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)
{
    t_fourcc filetype = 'TEXT', outtype;
    short numtypes = 1;
    char filename[MAX_PATH_CHARS];
    short path;

    if (s == gensym("")) {           // if no argument supplied, ask for file
        if (open_dialog(filename, &path, &outtype, &filetype, 1))           // non-zero: 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_handlesize(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_freeptr(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_get_parentpatcher\(\)](#). 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_get_parentpatcher(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(patch, 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
fccount	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
bblocked	char	Yes	Whether background is locked
rect	double[4]	Yes	Patcher's rect (left, top, width, height)
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 patcher (NULL for toplevel patchers)
toppatcher	object	No	Topmost parent patcher (NULL for toplevel patchers)
parentclass	object	No	Class object of parent (patcher, poly~, bpatcher etc.)
bgcolor	double[4]	Yes	Locked background color (RGBA)
editing_bgcolor	double[4]	Yes	Unlocked background color (RGBA)
edit_framecolor	double[4]	Yes	Text editing frame color
locked_iocolor	double[4]	Yes	Locked inlet/outlet color
unlocked_iocolor	double[4]	Yes	Unlocked inlet/outlet color

boguscolor	double[4]	Yes	Color of uninitialized (bogus) objects
gridsize	double[2]	Yes	Editing grid size
gridonopen	char	Yes	Show grid on open
gridsnapopen	char	Yes	Snap to grid on open
imprint	char	Yes	Save default-valued object attributes
defaultfocusbox	symbol	Yes	Default focus box (varname)
enablehscroll	char	Yes	Show horizontal scrollbar
enablevscroll	char	Yes	Show vertical scrollbar
boxanimatetime	long	Yes	Box animation time
default_fontname	symbol	Yes	Default font name
default_fontface	long	Yes	Default "fake" font face (0 plain, 1, bold, 2 italic, 3 bold italic)
default_fontsize	long	Yes	Default font size in points
toolbarvisible	char	Yes	Show toolbar on open
toolbarheight	long	Yes	Height of toolbar (can use 0 for invisible)
toolbarid	symbol	Yes	Name (in maxinterface.json) of toolbar, none = empty symbol

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
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 drawn?
growy	char	No	Can object grow with fixed aspect ratio?
growboth	char	No	Can object grow independently in width and height?
nogrow	char	No	Is object fixed size?
mousedowndragdelta	char	No	Does object use hidden-mouse drag tracking (number box)
textfield	object	No	Textfield object associated with this box if any
editactive	char	No	Is object the currently focused box in an unlocked patcher?
prototypename	symbol	No	Name of the prototype file used to create this object

presentation	char	Yes	Is object included in the presentation?
annotation	symbol	Yes	Text shown in clue window when mouse is over the object
numinlets	long	No	Number of inlets visible
numoutlets	long	No	Number of outlets visible
outlettype	symbol[]	No	Array of symbols with outlet types ("signal" etc.)

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:
                    strcpy_zero(s, "This is a description of the leftmost inlet", 512);
                    break;
                case 1:
                    strcpy_zero(s, "This is a description of the rightmost inlet", 512);
                    break;
            }
            break;
        case 2:
            strcpy_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_edsav, "edsav",
    A_CANT, 0);

long myobject_edsav(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"));
}
```

Chapter 15

Data Structures

15.1 Available Data Structures

The Max API provides a variety of useful data structures which may be used across platforms and provide basic thread-safety.

- [Atom Array](#) : container for an array of atoms
- [Linked List](#) : doubly-linked-list
- [Hash Table](#) : hash table for mapping symbols to data
- [Quick Map](#) : a double hash with keys mapped to values and vice-versa
- [Database](#) : SQLite database access
- [Index Map](#) : managed array of pointers
- [String Object](#) : wrapper for C-strings with an API for manipulating them
- [Symbol Object](#) : wrapper for symbols
- [Dictionary](#) : structured/hierarchical data that is both sortable and fast

15.2 Passing Data Structures

Most often, the use of a particular instance of a data structure will be limited to within the confines a single class or object you create. However, in some cases you may wish to pass structured data from one object to another. For this purpose, Max 6 introduced facilities for passing named [`t_dictionary`](#) instances.

Examples, descriptions, and API documentation can be found in [Dictionary Passing API](#).

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 ("myobject"). For example, consider what happens when a button is connected to the left inlet of myobject and 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 *view);
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 `ext_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** (constraints 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 **x)
{
    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_addmethod()` 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:

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

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.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 ext_main(void *r)
{
    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_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 object types, please consult the Max 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 `ext_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`, `importattrs`, `exportattrs`, 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.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 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_adddmethod(_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 *x,
    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,planeCount,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 `ext_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          0xFFFFFFFF
#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 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 sub-regions 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 `ext_main()` function. Here's an example from `jit.op`:

```
// add custom jit_matrix method in ext_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 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 up 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,attrrstart,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);

    attrrstart = max_jit_attr_args_offset(argc,argv);
    if (attrrstart&&argv) {
        jit_atom_arg_getlong(&info.planecount, 0, attrrstart, argv);
        jit_atom_arg_getsym(&info.type, 1, attrrstart, argv);
        i=2; j=0;
        while (i<attrrstart) { //dimensions
            jit_atom_arg_getlong(&(info.dim[j]), i, attrrstart, 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_typelink)) {
        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_typelink)) {
        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 ext_main(void *r)
{
    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
    address((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 && argv)
        {
            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 when 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 glBlendFunc.

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 **,
    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. \leftarrow 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 Max. 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_DEFFLOAT](#), [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_adddmethod(_jit_gl_videoplane_class, (method) jit_gl_videoplane_jit_matrix
    , "jit_matrix", A_USURP_LOW, 0);

// add a deferred sendtexture method
jit_class_adddmethod(_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,attrflags,
    (method)OL,(method) jit_gl_videoplane_displaylist,calcoffset(t_jit_gl_videoplane,
        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)OL,(method)OL,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 sent 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 input/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.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.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  
{  
    t_int32 id;  
    t_int32 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  
{  
    t_int32 id;  
    t_int32 size;  
    t_int32 planecount;  
    t_int32 type; //0=char,1=long,2=float32,3=float64  
    t_int32 dimcount;  
    t_int32 dim[JIT_MATRIX_MAX_DIMCOUNT];  
    t_int32 dimstride[JIT_MATRIX_MAX_DIMCOUNT];  
    t_int32 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
{
    t_int32 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` *, 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.

All data is represented with little endian byte ordering.

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	<code>t_max_err dbviewer_notify(t_dbviewer *x, t_symbol *s, t_symbol *msg, void *sender, void *data);</code>	
stdargs	<code>t_max_err my_obj(t_dictionary *d, t_symbol *s, long argc, t_atom *argv)</code>	when loading an old (Max 3 or 4) patcher, this will be called prior to your new method. You can then fill in the dictionary with key/value pairs from your previous args.
quickref		obsolete, this is provided automatically now

30.2 Messages for Non-UI Objects

dblclick	<code>void scripto_dblclick(t_scripto *x);</code>
----------	---

30.3 Messages for User Interface Objects

getdrawparams	<code>void uisimp_getdrawparams(t_uisimp *x, t_object *patcherview, t_jboxdrawparams *params);</code>	
mousedown	<code>void scripto_ui_mousedown(t_← scripto_ui *x, t_object *patcherview, t_pt pt, long modifiers);</code>	
mouseup	<code>void uisimp_mouseup(t_uisimp *x, t_object *patcherview, t_pt pt, long modifiers);</code>	
mousedrag	<code>void scripto_ui_mousedrag(t_← scripto_ui *x, t_object *patcherview, t_pt pt, long modifiers);</code>	
mouseenter	<code>void uisimp_mouseenter(t_uisimp *x, t_object *patcherview, t_pt pt, long modifiers);</code>	
mouseleave	<code>void uisimp_mouseleave(t_uisimp *x, t_object *patcherview, t_pt pt, long modifiers);</code>	
mousemove	<code>void uisimp_mousemove(t_uisimp *x, t_object *patcherview, t_pt pt, long modifiers);</code>	
paint	<code>void pictmeter_paint(t_pictmeter *x, t_object *patcherview);</code>	

30.4 Message for Audio Objects

dsp	<code>void plus_dsp(t_plus *x, t_signal **sp, short *count);</code>	
dspstate	<code>plus_dspstate(t_plus *x, long n);</code>	

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-icons** 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 `ext_main()` function) with which they are associated.

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.

It is recommended that you distribute your object as a Max Package (see [Appendix: SDK changes for Max 7](#)). Within this package you shoulo place your svg in the 'interfaces' folder.

31.2 Quick Lookup Definition

Adding the svg file will make the icon available to Max for use in some ways. To make your icon appear in Max's Object Explorer, however, you must create a quick-lookup (or qlookup) entry for your object. If you look in the **Cycling '74/interfaces** folder, you should notice some files with names like "obj-qlookup.json" and "doc-qlookup.json". For your object, you should create a similar qlookup file.

For the following example we will assume you have created an object called 'littleuifoo'. For this object we will create a qlookup called 'littleuifoo-obj-qlookup.json'. The contents of this file will look like this:

```
{
  "littleuifoo": {
    "digest": "Little UI Object that does Foo",
    "module": "max",
    "category": [
      "U/I"
    ],
    "palette": {
      "category": [
        "Interface"
      ],
      "action": "littleuifoo",
      "pic": "littleuifoo.svg"
    }
  }
}
```

}

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

Appendix: Updating Externals for Max 6

33.1 Updating MSP Externals for 64-bit Audio Processing

33.1.1 Background

In Max 5 and prior versions, the signal chain for processing audio was compiled by sending all objects in the patcher a "dsp" message. Objects responding to this message then executed their dsp method, typically adding one of the object's perform methods to the signal chain.

In Max 6, the signal chain is compiled by first sending objects a "dsp64" message. When your object responds to this message, you can add your 64-bit audio perform methods. If an object supports the old "dsp" message but not the "dsp64" message, it then wraps the older 32-bit perform routine with conversion on the inputs and outputs.

This means that the 64-bit engine will work just fine with the older 32-bit objects. However, the conversion comes with some computational expense. For the best performance your objects should support the 64-bit dsp chain natively by implementing the "dsp64" message as explained below.

33.1.2 API

As noted, instead of the "dsp" method used by objects for Max 5 and earlier, Max 6 objects implement a "dsp64" method. This has the same purpose as the original dsp method. One notable difference is that the signals are not passed to the dsp64 method. This is to allow for the signal that is used to change dynamically at runtime. However, the relevant info (samplerate, number of signals connected, etc) is passed in.

The main purpose of the dsp64 method is to call back into the audio lib to put perform methods on the dsp chain. This is done by sending the 'dsp_add64' message to the dspchain object using [object_method\(\)](#).

The perform routine is now of type t_perfroutine64, defined in z_dsp.h, and now has a fixed function signature. It does take a user-defined parameter that is passed back from the call to 'dsp_add64'.

33.1.3 Example Code

The simplemsp~ examples in the 'audio' folder of the SDK have been updated for 64-bit audio processing in Max 6. Several projects, including the simplemsp~ example, demonstrate how to support both 64-bit audio processing in Max 6 and 32-bit audio processing for compatibility with Max 5.

33.2 Updating Max Externals for Cocoa

On the Macintosh platform, Max 6 made the transition from using the Carbon API to using the Cocoa API for interacting with the Mac OS. In most cases the transition for third-party developers should be seamless. If you are operating directly using native Carbon calls then your code will need to be updated to Cocoa using Objective-C.

The most common scenario is where you ask a patcherview for the native window handle with a call such as:

```
WindowRef viewWindow;
object_method(patcherview, gensym("nativewindow"), (void**)&viewWindow);
```

In Max 6 this will not work because the returned 'viewWindow' is not the Carbon WindowRef but is instead a Cocoa NSWindow*. You may update your code to use Cocoa instead of Carbon, or you may wish to transition more slowly by continuing to use a WindowRef. Here is an example to assist in obtaining a WindowRef:

```
NSView      *cocoa_view = NULL;
NSWindow    *cocoa_window = NULL;
WindowRef   carbon_window;

object_method(patcherview, gensym("nativewindow"), (void**)&cocoa_view);
if (cocoa_view) {
    cocoa_window = [cocoa_view window];
    if (cocoa_window) {
        carbon_window = [cocoa_window windowRef];
    }
}

// now you can use your carbon_window as before
```

Chapter 34

Appendix: Updating Externals for Max 6.1 (x64 architecture)

34.1 Background

The Max 6.0.x application binary, and the external objects and libraries it uses, are compiled for the i386 processor architecture. This architecture uses 32-bit memory addressing, meaning that the size of a pointer is 32 bits (or 4 bytes).

Max 6.1 introduces support for the x86_64 (or x64) architecture which uses 64-bit (8 bytes) memory addressing. Among the benefits are the ability to use more than 2 GB of memory in Max. Additionally, the size of the `t_atom` is 8-bytes on x64, meaning that double-precision floating pointer numbers can be represented.

For backwards compatibility, Max 6.1 also continues to be distributed as a 32-bit application binary. On the Windows platform the 32-bit and 64-bit applications are distributed separately, as are the external objects you create for them. On the Mac platform a Universal Binary (or "FAT" binary) is distributed containing both the 32-bit and 64-bit versions in the same dynamically-loaded library.

All externals on the Mac remain bundles using the ".mxo" filename extension.

32-bit externals on Windows remain DLLs using the ".mxe" filename extension.

64-bit externals on Windows are still DLLs but use a new ".mxe64" filename extension.

34.1.1 New Types

In addition to the change of size in a pointer, there are some additional changes for 64-bit. For example, a "long" integer for 32-bit targets is 4 bytes on both the Mac and Windows. However, a "long" integer for 64-bit targets is 4 bytes on Windows but 8 bytes (the size of a pointer) on the Mac!

To facilitate cross platform code that is independent of these platform differences, the Max 6.1 API defines some new types used throughout the SDK.

Types of fixed size:

```
t_int8  
t_uint8  
t_int16  
t_uint16  
t_int32  
t_uint32  
t_int64  
t_uint64
```

Types of architecture-dependent size:

```
t_ptr_int : an int that is the same size as a pointer
```

```
t_ptr_uint : an unsigned int that is the same size as a pointer
t_atom_long : the type that is an A_LONG in an atom (32-bits on i386; 64-bits on x64)
t_atom_float : the type that is an A_FLOAT in an atom (float on i386; double on x64)
```

Types for specific contexts:

```
t_filepath : i.e. path/vol in file APIs identifying a folder
t_fourcc : use for type codes in locatefile_extended(), file dialogs, etc. to
           represent a four char code
t_getbytes_size : if you are using getbytes() and freebytes(), use this type to
                   represent the size of the memory
```

34.2 Xcode and Visual Studio Projects

For new objects, you can base projects on those in the new SDK. To update existing projects you will need to make a few changes to your project settings.

34.2.1 Mac / Xcode

Max 6.1 on the Mac no longer uses the intermediary MaxAPI.framework for linking. Instead, the linking is handled at runtime and the symbols are checked using special flags to the linker. To update an existing project:

1. remove references to the MaxAPI.framework
2. update the .xcconfig file on which the project is based with the .xcconfig file in the new Max SDK
3. in your target's build settings find the "Other Linker Flags" and set it to "\$(C74_SYM_LINKER_FLAGS)"
4. in your target's build settings find the "Architectures" and set it to "i386 x86_64"

34.2.2 Windows / Visual Studio

In order to build for x64 with Visual Studio 2008, you must have the "Pro" version. The free "Express" version will not work. The "Express" versions of Visual Studio 2010 and 2012 do work (2012 is recommended).

Due to bugs in Visual Studio 2008, it is really difficult to update an existing project. Instead, it is recommended to simply create a new Visual Studio project based on an existing example. For Visual Studio 2008 use the "vcproj" files. For Visual Studio 2010 and 2012 use the "vcxproj" files.

1. choose a relevant starting point, e.g., the "dummy" example project
2. copy it to the folder your old project was in, rename it
3. open it in a text editor such as "sublime text 2"
4. do a find/replace for all instances of the text "dummy" changing it to your object's name
5. open the Visual Studio project and build you can choose either "Win32" or "x64" from the platform drop-down menu in the IDE

34.3 Changes to Code

34.3.1 Atoms

Any assumptions in your code about the size of a `t_atom` or the size of its members should be reviewed. When setting or getting values to and from atoms you should use the types `t_atom_long` and `t_atom_float` as appropriate.

34.3.2 Return Values

All methods which return a value must return a pointer-sized value, e.g., [t_ptr_int](#), [t_ptr_uint](#), [t_max_err](#), etc.

34.3.3 File Code

File access in Max involves several areas subject to either required or suggested update.

A path in Max has traditionally been represented with a short int; it is recommended to now use the new [t_filepath](#) type.

File types in Max are represented using four char codes. Traditionally these have been defined using variables of type "long", which is now problematic. This is a 4-byte type but the long on the Mac for x64 is 8-bytes. These must be updated to use the new [t_fourcc](#) type.

34.3.4 Miscellaneous Gotchas

Look for any Id or lu strings in sprintf() or related formatting functions.

34.4 Common Scenarios

34.4.1 "Long" integers

One of biggest areas we've had to address is the use of the long datatype. The reason for this is that under 64bit windows a long integer is 32 bits and under 64bit OS X (and Unix), a long is a 64 bit integer.

To assist in this process, we have a the new data types documented above. We'll distinguish these from what we are calling a "platform long".

This platform long discrepancy can lead to all sorts of problems which are outlined with a brief statement of the problem scenario, and our recommended fix with types:

Problem: long integers as A_LONG/A_DEFLONG method arguments (this includes your object constructors)

Solution: type your A_LONG/A_DEFLONG methods' function signatures to use the [t_atom_long](#) in place of long

Problem: long integers as A_CANT method arguments called only through [object_method\(\)](#)

Solution: either redefine your A_CANT method's arguments to [t_atom_long](#), or define your type as A_DIRECT, and make use of the [object_method_direct\(\)](#) macro, passing in a function prototype to the macro (also see under floating point how this is required for anything which previously was A_CANT with floating point values). Technically many of these will still work properly due to the nature of how integers are passed on the stack under x64, without any change, it is still best practice.

Problem: long integers being used to store pointers as integer values either for pointer arithmetic, attributes, or other situations.

Solution: use [t_atom_long](#) or even better [t_ptr_uint](#) (for pointer sized unsigned integer) or the actual pointer type.

Problem: long integers as four character codes for filetypes ([t_fourcc](#)) which applies to [locatefile_extended](#) and [path](#) functions and friends

Solution: Use [t_foucc](#) instead of long, for anywhere you are using filetype codes.

Problem: long integers as return values for functions called via [object_method\(\)](#)

Solution: These should always return a [t_atom_long](#) or other pointer sized integer

Problem: long integers passed as pointers into functions like [dictionary_getlong\(\)](#) which are now prototyped to take a [t_atom_long](#) *

Solution: Use a t_atom_long value, and pass a pointer to it. A cast from a platform long *to a t_atom_long * is not safe.

Problem: long integers for performing bitwise manipulation of 32bit floating point values including byteswapping

Solution: Use t_int32/t_uint32

There are many cases where it is safe to use long integers, and we have continued to use them in our code. Below are the scenarios where they are okay and in several cases required. This might provide some confusion at some points, but hopefully it makes the porting process a little bit easier, allowing more code to remain unchanged.

- Attributes defined as _sym_long should remain a platform long. If you need to have a t_atom_long attribute, you will need to use the new atom_long attribute type. This is probably the most confusing aspect of porting to 64bit and a very real ambiguity of the word "long". Unfortunately, having to balance the difficulties of porting with the clarity of API, this is something we felt necessary to do.
- Attribute getters/setters should still use the long type for ac. this is especially important for getters which are passed a pointer to a platform long in the ac value.
- A_GIMME methods may still use the long type for ac without issues

34.4.2 Floating point values

For A_FLOAT/A_DEFFLOAT function signatures, you should always use double as is currently recommended in the Max SDK. You should not use the new t_atom_float datatype. (this includes your object constructors)

For A_CANT functions with floating point arguments that currently use [object_method\(\)](#). You will need to use [object_method_direct\(\)](#) or pass in pointers to the floating point values (which is safe as it is a pointer sized integer). It is no longer possible to pass floats through [object_method\(\)](#) or the many functions like it ([linklist_methodall\(\)](#), [hashtab_methodall\(\)](#), etc.)

Attributes are already defined in terms of their bitsize float32 or float 64. If you wish for your attribute to make use of the new atom support for double precision. You will want to change your struct definition, as well as your attribute constructor to be a double (_sym_float64). There isn't currently a t_atom_float attribute type like we've added for long.

34.4.3 Deprecated Apple types

like Byte/Boolean/Point/etc

Use the t_byte/t_bool/t_point/etc types instead.

34.5 Additional Miscellaneous Changes

The old 32-bit 'dsp' (and perform) methods are no longer supported as of Max 6.1. They must be updated as per [Updating MSP Externals for 64-bit Audio Processing](#).

34.6 Additional Resources

- <http://www.viva64.com/en/a/0004/>
- <https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/Cocoa64BitGuide/64BitChangesCocoa/64BitChangesCocoa.html>
- <https://developer.apple.com/library/mac/#documentation/Carbon/Conceptual/Carbon64BitGuide/OtherAPIChanges/OtherAPIChanges.html>

- <http://msdn.microsoft.com/en-us/magazine/cc300794.aspx>

Chapter 35

Appendix: SDK changes for Max 7

35.1 Using `ext_main` for Entry Point

Prior to Max 7 the entry point for externals was the `main()` function exported from the dynamic library you create. Beginning with Max 7 the entry point for externals is called `ext_main()`. This addresses compatibility problems with various newer compilers and frees us from the constraints enforced for `main()` as the standard entry point for programs.

Objects that do not define `ext_main()` will still be loaded using the older `main()`. Support for `ext_main()` is also present in Max 6.1.9.

35.2 Support for Max 7 UI Object Styles

Max 7 introduces the concept of styles which determine the appearance of UI objects. For attributes of your U←I object to map to colors or attributes of a style you need to add the required attribute properties in your class definition.

See the section on [Styles](#) for more information.

Chapter 36

Module Documentation

36.1 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.

Macros

- `#define CLASS_ATTR_CHAR(c, attrname, flags, structname, structmember)`
Create a char attribute and add it to a Max class.
- `#define CLASS_ATTR_LONG(c, attrname, flags, structname, structmember)`
Create a t_atom_long integer attribute and add it to a Max class.
- `#define CLASS_ATTR_ATOM_LONG(c, attrname, flags, structname, structmember)`
Create a t_atom_long integer attribute and add it to a Max class.
- `#define CLASS_ATTR_INT32(c, attrname, flags, structname, structmember)`
Create a t_int32 integer attribute and add it to a Max class.
- `#define CLASS_ATTR_FLOAT(c, attrname, flags, structname, structmember)`
Create a 32-bit float attribute and add it to a Max class.
- `#define CLASS_ATTR_DOUBLE(c, attrname, flags, structname, structmember)`
Create a 64-bit float attribute and add it to a Max class.
- `#define CLASS_ATTR_SYMBOL(c, attrname, flags, structname, structmember)`
Create a t_symbol attribute and add it to a Max class.*
- `#define CLASS_ATTR_ATOM(c, attrname, flags, structname, structmember)`
Create a t_atom attribute and add it to a Max class.
- `#define CLASS_ATTR_OBJECT(c, attrname, flags, 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)`
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)`
Create an array-of-long-integers attribute of fixed length, and add it to a Max class.
- `#define CLASS_ATTR_ATOM_LONG_ARRAY(c, attrname, flags, structname, structmember, size)`
Create an array-of-t_atom_long-integers attribute of fixed length, and add it to a Max class.
- `#define CLASS_ATTR_FLOAT_ARRAY(c, attrname, flags, structname, structmember, size)`

- `#define CLASS_ATTR_DOUBLE_ARRAY(c, attrname, flags, structname, structmember, size)`

Create an array-of-32bit-floats attribute of fixed length, and add it to a Max class.
- `#define CLASS_ATTR_SYM_ARRAY(c, attrname, flags, structname, structmember, size)`

Create an array-of-64bit-floats attribute of fixed length, and add it to a Max class.
- `#define CLASS_ATTR_ATOM_ARRAY(c, attrname, flags, structname, structmember, size)`

Create an array-of-symbols attribute of fixed length, and add it to a Max class.
- `#define CLASS_ATTR_OBJ_ARRAY(c, attrname, flags, structname, structmember, size)`

Create an array-of-atoms attribute of fixed length, and add it to a Max class.
- `#define CLASS_ATTR_CHAR_VARSIZE(c, attrname, flags, structname, structmember, sizemember, maxsize)`

Create an array-of-objects attribute of fixed length, and add it to a Max class.
- `#define CLASS_ATTR_LONG_VARSIZE(c, attrname, flags, structname, structmember, sizemember, maxsize)`

Create an array-of-chars attribute of variable length, and add it to a Max class.
- `#define CLASS_ATTR_FLOAT_VARSIZE(c, attrname, flags, structname, structmember, sizemember, maxsize)`

Create an array-of-long-integers attribute of variable length, and add it to a Max class.
- `#define CLASS_ATTR_DOUBLE_VARSIZE(c, attrname, flags, structname, structmember, sizemember, maxsize)`

Create an array-of-32bit-floats attribute of variable length, and add it to a Max class.
- `#define CLASS_ATTR_SYM_VARSIZE(c, attrname, flags, structname, structmember, sizemember, maxsize)`

Create an array-of-64bit-floats attribute of variable length, and add it to a Max class.
- `#define CLASS_ATTR_ATOM_VARSIZE(c, attrname, flags, structname, structmember, sizemember, maxsize)`

Create an array-of-symbols attribute of variable length, and add it to a Max class.
- `#define CLASS_ATTR_OBJ_VARSIZE(c, attrname, flags, structname, structmember, maxsize)`

Create an array-of-atoms attribute of variable length, and add it to a Max class.
- `#define STRUCT_ATTR_CHAR(c, flags, structname, structmember)`

Create a char attribute and add it to a Max class.
- `#define STRUCT_ATTR_LONG(c, flags, structname, structmember)`

Create a long integer attribute and add it to a Max class.
- `#define STRUCT_ATTR_ATOM_LONG(c, flags, structname, structmember)`

Create a t_atom_long integer attribute and add it to a Max class.
- `#define STRUCT_ATTR_FLOAT(c, flags, structname, structmember)`

Create a 32bit float attribute and add it to a Max class.
- `#define STRUCT_ATTR_DOUBLE(c, flags, structname, structmember)`

Create a 64bit float attribute and add it to a Max class.
- `#define STRUCT_ATTR_SYM(c, flags, structname, structmember)`

Create a t_symbol attribute and add it to a Max class.*
- `#define STRUCT_ATTR_ATOM(c, flags, structname, structmember)`

Create a t_atom attribute and add it to a Max class.
- `#define STRUCT_ATTR_OBJ(c, 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)`

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)`

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)`

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)`

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)`
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)`
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)`
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)`
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)`
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)`
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)`
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)`
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)`
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)`
Create an array-of-objects attribute of variable length, and add it to a Max class.
- `#define OBJ_ATTR_CHAR(x, attrname, flags, val)`
Create an instance-local char attribute and add it to a Max class.
- `#define OBJ_ATTR_LONG(x, attrname, flags, val)`
Create an instance-local long integer attribute and add it to a Max class.
- `#define OBJ_ATTR_FLOAT(x, attrname, flags, val)`
Create an instance-local 32bit float attribute and add it to a Max class.
- `#define OBJ_ATTR_DOUBLE(x, attrname, flags, val)`
Create an instance-local 64bit float attribute and add it to a Max class.
- `#define OBJ_ATTR_SYM(x, attrname, flags, val)`
Create an instance-local `t_symbol` attribute and add it to a Max class.*
- `#define OBJ_ATTR_ATOM(x, attrname, flags, val)`
Create an instance-local `t_atom` attribute and add it to a Max class.
- `#define OBJ_ATTR_OBJ(x, attrname, flags, 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)`
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)`
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)`
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)`
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)`
Create an instance-local array-of-symbols attribute of fixed length, and add it to the object.
- `#define OBJ_ATTR_ATOM_ARRAY`
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)`
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)`
Add a new attribute to the specified attribute to specify a default value.
- `#define CLASS_ATTR_SAVE(c, attrname, flags)`
Add a new attribute to the specified attribute to indicate that the specified attribute should be saved with the patcher.
- `#define CLASS_ATTR_SELFSAVE(c, attrname, flags)`
Add a new attribute to the specified attribute to indicate that it is saved by the object (so it does not appear in italics in the inspector).
- `#define CLASS_ATTR_DEFAULT_SAVE(c, attrname, flags, parsestr)`
A convenience wrapper for both `CLASS_ATTR_DEFAULT` and `CLASS_ATTR_SAVE`.
- `#define CLASS_ATTR_DEFAULTNAME(c, attrname, 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)`
A convenience wrapper for both `CLASS_ATTR_DEFAULTNAME` and `CLASS_ATTR_SAVE`.
- `#define CLASS_ATTR_MIN(c, attrname, flags, parsestr)`
Add a new attribute to the specified attribute to specify a lower range.
- `#define CLASS_ATTR_MAX(c, attrname, flags, parsestr)`
Add a new attribute to the specified attribute to specify an upper range.
- `#define CLASS_ATTR_PAINT(c, attrname, flags)`
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)`
A convenience wrapper for both `CLASS_ATTR_DEFAULT` and `CLASS_ATTR_PAINT`.
- `#define CLASS_ATTR_DEFAULT_SAVE_PAINT(c, attrname, flags, parsestr)`
A convenience wrapper for `CLASS_ATTR_DEFAULT`, `CLASS_ATTR_SAVE`, and `CLASS_ATTR_PAINT`.
- `#define CLASS_ATTR_DEFAULTNAME_PAINT(c, attrname, flags, parsestr)`
A convenience wrapper for `CLASS_ATTR_DEFAULTNAME`, `CLASS_ATTR_SAVE`, and `CLASS_ATTR_PAINT`.
- `#define CLASS_ATTR_DEFAULTNAME_SAVE_PAINT(c, attrname, flags, parsestr)`
A convenience wrapper for `CLASS_ATTR_DEFAULTNAME`, `CLASS_ATTR_SAVE`, and `CLASS_ATTR_PAINT`.
- `#define CLASS_ATTR_STYLE(c, attrname, 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)`
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)`
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)`
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)`
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)`

- A convenience wrapper for `CLASS_ATTR_STYLE`, and `CLASS_ATTR_LABEL`.
- `#define CLASS_ATTR_INVISIBLE(c, attrname, flags)`

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)`

Add a new attribute to the specified attribute to specify a default order in which to list attributes.
- `#define CLASS_ATTR_BASIC(c, attrname, flags)`

Add a new attribute to the specified attribute to specify that it should appear in the inspector's Basic tab.
- `#define CLASS_METHOD_ATTR_PARSE(c, methodname, attrname, type, flags, parsestring)`

Define and add attributes to class methods.
- `#define CLASS_ATTR_LEGACYDEFAULT(c, legacyattrname, newattrname, flags, parsestr)`

Add a new attribute to the specified attribute to specify a legacy default value.
- `#define CLASS_ATTR_OBSOLETE(c, attrname, flags)`

Add a new attribute to the specified attribute to flag it as obsolete.
- `#define CLASS_ATTR_RENAMED(c, oldname, newname, flags)`

Add a new attribute to the specified attribute to flag it as renamed.
- `#define CLASS_ATTR_INTRODUCED(c, attrname, flags, versionstr)`

Add a new attribute to the specified attribute to indicate in which version the attribute was introduced.
- `#define CLASS_METHOD_OBSOLETE(c, methodname, flags)`

Add a new attribute to the specified method to flag it as obsolete.
- `#define CLASS_METHOD_RENAMED(c, oldname, newname, flags)`

Add a new attribute to the specified method to flag a method as renamed.
- `#define CLASS_METHOD_INTRODUCED(c, methodname, flags, versionstr)`

Add a new attribute to the specified method to indicate in which version the method was introduced.
- `#define OBJ_ATTR_DEFAULT(x, attrname, flags, parsestr)`

An instance-attribute version of `CLASS_ATTR_DEFAULT`.
- `#define OBJ_ATTR_SAVE(x, attrname, flags)`

An instance-attribute version of `CLASS_ATTR_SAVE`.
- `#define OBJ_ATTR_DEFAULT_SAVE(x, attrname, flags, parsestr)`

An instance-attribute version of `CLASS_ATTR_DEFAULT_SAVE`.
- `#define CLASS_STICKY_ATTR(c, name, flags, parsestr)`

Create an attribute, and add it to all following attribute declarations.
- `#define CLASS_STICKY_ATTR_CLEAR(c, name)`

Close a `CLASS_STICKY_ATTR` block.
- `#define CLASS_STICKY_METHOD(c, name, flags, parsestr)`

Create an attribute, and add it to all following method declarations.
- `#define CLASS_STICKY_METHOD_CLEAR(c, name)`

Close a `CLASS_STICKY_METHOD` block.
- `#define CLASS_ATTR_RGBA(c, attrname, flags, structname, structmember)`

Create a color (`t_rgba`) attribute and add it to a Max class.

Enumerations

- enum `e_max_attrflags` {
`ATTR_FLAGS_NONE, ATTR_GET_OPAQUE, ATTR_SET_OPAQUE, ATTR_GET_OPAQUE_USER,`
`ATTR_SET_OPAQUE_USER }`
- 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_CON←
ST 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.
- `t_atom_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, t_atom_long c)`
Sets the value of an attribute, given its parent object and name.
- `t_atom_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, t_atom_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, t_atom_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, t_atom_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, t_uint8 *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 `t_uint8` *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.
- `void attr_dictionary_check` (void *x, `t_dictionary` *d)

Check that a dictionary only contains values for existing attributes of an object.
- `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)

- void [object_attr_set_xywh](#) ([t_object](#) *o, [t_symbol](#) *attr, double x, double y, double w, double h)

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)

Sets the value of a [t_rect](#) attribute, given its parent object and name.
- [t_max_err object_attr_setpt](#) ([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_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.

36.1.1 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 its class initialization or [ext-<-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.

36.1.2 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_A->TTR_ACCESSORS](#) macro.

36.1.2.1 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..

36.1.2.2 Writing a custom Attribute Setter

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;
}

```

36.1.3 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.

36.1.4 Macro Definition Documentation

36.1.4.1 #define CLASS_ATTR_ACCESSORS(c, attrname, getter, 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

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>getter</i>	An appropriate getter method as discussed in Setting and Getting Attribute Values , or NULL to use the default getter.
<i>setter</i>	An appropriate setter method as discussed in Setting and Getting Attribute Values , or NULL to use the default setter.

36.1.4.2 #define CLASS_ATTR_ADD_FLAGS(c, attrname, flags)

Add flags to an attribute.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to add to this attribute, as defined in e_max_attrflags .

36.1.4.3 #define CLASS_ATTR_ALIAS(*c, attrname, aliasname*)

Create a new attribute that is an alias of an existing attribute.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the actual attribute as a C-string.
<i>aliasname</i>	The name of the new alias attribute.

36.1.4.4 #define CLASS_ATTR_ATOM(*c, attrname, flags, structname, structmember*)

Create a [t_atom](#) attribute and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.5 #define CLASS_ATTR_ATOM_ARRAY(*c, attrname, flags, structname, structmember, size*)

Create an array-of-atoms attribute of fixed length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of items in the t_atom array.

36.1.4.6 #define CLASS_ATTR_ATOM_LONG(*c, attrname, flags, structname, structmember*)

Create a [t_atom_long](#) integer attribute and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .

<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.7 `#define CLASS_ATTR_ATOM_LONG_ARRAY(c, attrname, flags, structname, structmember, size)`

Create an array-of-[t_atom](#)_long-integers attribute of fixed length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of longs in the array.

36.1.4.8 `#define CLASS_ATTR_ATOM_VARSIZE(c, attrname, flags, structname, structmember, sizemember, maxsize)`

Create an array-of-atoms attribute of variable length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>sizemember</i>	The actual number of items in the t_atom array at any given moment.
<i>maxsize</i>	The maximum number of items in the t_atom array, i.e. the number of members allocated for the array in the struct.

36.1.4.9 `#define CLASS_ATTR_BASIC(c, attrname, flags)`

Add a new attribute to the specified attribute to specify that it should appear in the inspector's Basic tab.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .

Referenced by [jit_ob3d_setup\(\)](#).

36.1.4.10 `#define CLASS_ATTR_CATEGORY(c, attrname, 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

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	A C-string, which will be parsed into an array of atoms to set the initial value.

36.1.4.11 #define CLASS_ATTR_CHAR(*c*, *attrname*, *flags*, *structname*, *structmember*)

Create a char attribute and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.12 #define CLASS_ATTR_CHAR_ARRAY(*c*, *attrname*, *flags*, *structname*, *structmember*, *size*)

Create an array-of-chars attribute of fixed length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of chars in the array.

36.1.4.13 #define CLASS_ATTR_CHAR_VARSIZE(*c*, *attrname*, *flags*, *structname*, *structmember*, *sizemember*, *maxsize*)

Create an array-of-chars attribute of variable length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>sizemember</i>	The actual number of items in the char array at any given moment.
<i>maxsize</i>	The maximum number of items in the char array, i.e. the number of members allocated for the array in the struct.

36.1.4.14 #define CLASS_ATTR_DEFAULT(*c*, *attrname*, *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

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	A C-string, which will be parsed into an array of atoms to set the initial value.

36.1.4.15 #define CLASS_ATTR_DEFAULT_PAINT(*c*, *attrname*, *flags*, *parsestr*)

A convenience wrapper for both [CLASS_ATTR_DEFAULT](#) and [CLASS_ATTR_PAINT](#).

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	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](#)

36.1.4.16 #define CLASS_ATTR_DEFAULT_SAVE(*c*, *attrname*, *flags*, *parsestr*)

A convenience wrapper for both [CLASS_ATTR_DEFAULT](#) and [CLASS_ATTR_SAVE](#).

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	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](#)

36.1.4.17 #define CLASS_ATTR_DEFAULT_SAVE_PAINT(*c*, *attrname*, *flags*, *parsestr*)

A convenience wrapper for [CLASS_ATTR_DEFAULT](#), [CLASS_ATTR_SAVE](#), and [CLASS_ATTR_PAINT](#).

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	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](#)

36.1.4.18 #define CLASS_ATTR_DEFAULTNAME(*c*, *attrname*, *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

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	A C-string, which will be parsed into an array of atoms to set the initial value.

36.1.4.19 #define CLASS_ATTR_DEFAULTNAME_PAINT(*c*, *attrname*, *flags*, *parsestr*)

A convenience wrapper for [CLASS_ATTR_DEFAULTNAME](#), [CLASS_ATTR_SAVE](#), and [CLASS_ATTR_PAINT](#).

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	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](#)

36.1.4.20 #define CLASS_ATTR_DEFAULTNAME_SAVE(*c*, *attrname*, *flags*, *parsestr*)

A convenience wrapper for both [CLASS_ATTR_DEFAULTNAME](#) and [CLASS_ATTR_SAVE](#).

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	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](#)

36.1.4.21 #define CLASS_ATTR_DEFAULTNAME_SAVE_PAINT(*c*, *attrname*, *flags*, *parsestr*)

A convenience wrapper for [CLASS_ATTR_DEFAULTNAME](#), [CLASS_ATTR_SAVE](#), and [CLASS_ATTR_PAINT](#).

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	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](#)

36.1.4.22 #define CLASS_ATTR_DOUBLE(*c*, *attrname*, *flags*, *structname*, *structmember*)

Create a 64-bit float attribute and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.23 #define CLASS_ATTR_DOUBLE_ARRAY(*c*, *attrname*, *flags*, *structname*, *structmember*, *size*)

Create an array-of-64bit-floats attribute of fixed length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of doubles in the array.

36.1.4.24 #define CLASS_ATTR_DOUBLE_VARSIZE(*c*, *attrname*, *flags*, *structname*, *structmember*, *sizemember*, *maxsize*)

Create an array-of-64bit-floats attribute of variable length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.

<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>sizemember</i>	The actual number of items in the double array at any given moment.
<i>maxsize</i>	The maximum number of items in the double array, i.e. the number of members allocated for the array in the struct.

36.1.4.25 #define CLASS_ATTR_ENUM(*c*, *attrname*, *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

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	A C-string, which will be parsed into an array of atoms to set the initial value.

Remarks

This macro automatically calls

```
1 CLASS_ATTR_STYLE(c, attrname, flags, "enum").
```

See also

[CLASS_ATTR_ENUMINDEX](#)

Referenced by `jit_ob3d_setup()`.

36.1.4.26 #define CLASS_ATTR_ENUMINDEX(*c*, *attrname*, *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

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	A C-string, which will be parsed into an array of atoms to set the initial value.

Remarks

This macro automatically calls

```
1 CLASS_ATTR_STYLE(c, attrname, flags, "enumindex").
```

See also

[CLASS_ATTR_ENUM](#)

36.1.4.27 #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.

The limiting will be performed by the default attribute accessor.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>minval</i>	The maximum acceptable value to which the attribute will be limited.
<i>maxval</i>	The maximum acceptable value to which the attribute will be limited.

See also

[36.1.4.28 #define CLASS_ATTR_FILTER_MAX\(*c*, *attrname*, *maxval* \)](#)

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

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>maxval</i>	The maximum acceptable value to which the attribute will be limited.

See also

[CLASS_ATTR_FILTER_MIN](#)
[CLASS_ATTR_FILTER_CLIP](#)
[CLASS_ATTR_MAX](#)

[36.1.4.29 #define CLASS_ATTR_FILTER_MIN\(*c*, *attrname*, *minval* \)](#)

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

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>minval</i>	The minimum acceptable value to which the attribute will be limited.

See also

[CLASS_ATTR_FILTER_MAX](#)
[CLASS_ATTR_FILTER_CLIP](#)
[CLASS_ATTR_MIN](#)

[36.1.4.30 #define CLASS_ATTR_FLOAT\(*c*, *attrname*, *flags*, *structname*, *structmember* \)](#)

Create a 32-bit float attribute and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.31 #define CLASS_ATTR_FLOAT_ARRAY(*c*, *attrname*, *flags*, *structname*, *structmember*, *size*)

Create an array-of-32bit-floats attribute of fixed length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of floats in the array.

36.1.4.32 #define CLASS_ATTR_FLOAT_VARSIZE(*c*, *attrname*, *flags*, *structname*, *structmember*, *sizemember*, *maxsize*)

Create an array-of-32bit-floats attribute of variable length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>sizemember</i>	The actual number of items in the float array at any given moment.
<i>maxsize</i>	The maximum number of items in the float array, i.e. the number of members allocated for the array in the struct.

36.1.4.33 #define CLASS_ATTR_INT32(*c*, *attrname*, *flags*, *structname*, *structmember*)

Create a [t_int32](#) integer attribute and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.34 #define CLASS_ATTR_INTRODUCED(*c*, *attrname*, *flags*, *versionstr*)

Add a new attribute to the specified attribute to indicate in which version the attribute was introduced.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>versionstr</i>	A C-string, which will be parsed set the version number (e.g. "7.0.0").

36.1.4.35 #define CLASS_ATTR_INVISIBLE(*c*, *attrname*, *flags*)

Add a new attribute to the specified attribute to flag an attribute as invisible to the Max inspector.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .

36.1.4.36 #define CLASS_ATTR_LABEL(*c*, *attrname*, *flags*, *labelstr*)

Add a new attribute to the specified attribute to specify a human-friendly label for the Max inspector.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>labelstr</i>	A C-string, which will be parsed into an array of atoms to set the initial value.

36.1.4.37 #define CLASS_ATTR_LEGACYDEFAULT(*c*, *legacyattrname*, *newattrname*, *flags*, *parsestr*)

Add a new attribute to the specified attribute to specify a legacy 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

<i>c</i>	The class pointer.
<i>legacyattrname</i>	The name of the attribute.
<i>newattrname</i>	The name of the attribute.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	A C-string, which will be parsed into an array of atoms to set the legacy value, used by <code>jbox_processlegacydefaults()</code>

36.1.4.38 #define CLASS_ATTR_LONG(*c*, *attrname*, *flags*, *structname*, *structmember*)

Create a t_atom_long integer attribute and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.

<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.39 #define CLASS_ATTR_LONG_ARRAY(*c*, *attrname*, *flags*, *structname*, *structmember*, *size*)

Create an array-of-long-integers attribute of fixed length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of longs in the array.

36.1.4.40 #define CLASS_ATTR_LONG_VARSIZE(*c*, *attrname*, *flags*, *structname*, *structmember*, *sizemember*, *maxsize*)

Create an array-of-long-integers attribute of variable length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>sizemember</i>	The actual number of items in the long array at any given moment.
<i>maxsize</i>	The maximum number of items in the long array, i.e. the number of members allocated for the array in the struct.

36.1.4.41 #define CLASS_ATTR_MAX(*c*, *attrname*, *flags*, *parsestr*)

Add a new attribute to the specified attribute to specify an upper range.

The values will not be automatically limited.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	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](#)

36.1.4.42 #define CLASS_ATTR_MIN(*c*, *attrname*, *flags*, *parsestr*)

Add a new attribute to the specified attribute to specify a lower range.

The values will not be automatically limited.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	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](#)

36.1.4.43 #define CLASS_ATTR_OBJ(*c*, *attrname*, *flags*, *structname*, *structmember*)

Create a [t_object*](#) attribute and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.44 #define CLASS_ATTR_OBJ_ARRAY(*c*, *attrname*, *flags*, *structname*, *structmember*, *size*)

Create an array-of-objects attribute of fixed length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of items in the t_object* array.

36.1.4.45 #define CLASS_ATTR_OBJ_VARSIZE(*c*, *attrname*, *flags*, *structname*, *structmember*, *sizemember*, *maxsize*)

Create an array-of-objects attribute of variable length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
----------	--------------------

<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>sizemember</i>	The actual number of items in the t_object * array at any given moment.
<i>maxsize</i>	The maximum number of items in the t_object * array, i.e. the number of members allocated for the array in the struct.

36.1.4.46 #define CLASS_ATTR_OBSOLETE(*c, attrname, flags*)

Add a new attribute to the specified attribute to flag it as obsolete.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .

36.1.4.47 #define CLASS_ATTR_ORDER(*c, attrname, flags, parsestr*)

Add a new attribute to the specified attribute to specify a default order in which to list attributes.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	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:

```
1 CLASS_ATTR_ORDER(c, "firstattr",    0, "1");
2 CLASS_ATTR_ORDER(c, "secondattr",   0, "2");
3 CLASS_ATTR_ORDER(c, "thirdattr",    0, "3");
```

36.1.4.48 #define CLASS_ATTR_PAINT(*c, attrname, flags*)

Add a new attribute indicating that any changes to the specified attribute will trigger a call to the object's paint method.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .

36.1.4.49 #define CLASS_ATTR_REMOVE_FLAGS(*c, attrname, flags*)

Remove flags from an attribute.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to remove from this attribute, as defined in e_max_attrflags .

36.1.4.50 `#define CLASS_ATTR_RENAMED(c, oldname, newname, flags)`

Add a new attribute to the specified attribute to flag it as renamed.

Parameters

<i>c</i>	The class pointer.
<i>oldname</i>	The name of the old attribute as a C-string.
<i>newname</i>	The name of the new attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .

36.1.4.51 `#define CLASS_ATTR_RGBA(c, attrname, flags, structname, structmember)`

Create a color ([t_rgba](#)) attribute and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.52 `#define CLASS_ATTR_SAVE(c, attrname, flags)`

Add a new attribute to the specified attribute to indicate that the specified attribute should be saved with the patcher.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .

36.1.4.53 `#define CLASS_ATTR_SELFSAVE(c, attrname, flags)`

Add a new attribute to the specified attribute to indicate that it is saved by the object (so it does not appear in italics in the inspector).

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .

36.1.4.54 `#define CLASS_ATTR_STYLE(c, attrname, 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

<code>c</code>	The class pointer.
<code>attrname</code>	The name of the attribute as a C-string.
<code>flags</code>	Any flags you wish to declare for this new attribute, as defined in <code>e_max_attrflags</code> .
<code>parsestr</code>	A C-string, which will be parsed into an array of atoms to set the initial value.

36.1.4.55 #define CLASS_ATTR_STYLE_LABEL(`c`, `attrname`, `flags`, `stylestr`, `labelstr`)

A convenience wrapper for `CLASS_ATTR_STYLE`, and `CLASS_ATTR_LABEL`.

Parameters

<code>c</code>	The class pointer.
<code>attrname</code>	The name of the attribute as a C-string.
<code>flags</code>	Any flags you wish to declare for this new attribute, as defined in <code>e_max_attrflags</code> .
<code>stylestr</code>	A C-string that names the style for the attribute. See <code>CLASS_ATTR_STYLE</code> for the available styles.
<code>labelstr</code>	A C-string that names the category to which the attribute is assigned in the inspector.

See also

[CLASS_ATTR_STYLE](#)
[CLASS_ATTR_LABEL](#)

36.1.4.56 #define CLASS_ATTR_SYM(`c`, `attrname`, `flags`, `structname`, `structmember`)

Create a `t_symbol*` attribute and add it to a Max class.

Parameters

<code>c</code>	The class pointer.
<code>attrname</code>	The name of this attribute as a C-string.
<code>flags</code>	Any flags you wish to declare for this attribute, as defined in <code>e_max_attrflags</code> .
<code>structname</code>	The C identifier for the struct (containing a valid <code>t_object</code> header) representing an instance of this class.

<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
---------------------	--

36.1.4.57 #define CLASS_ATTR_SYM_ARRAY(*c*, *attrname*, *flags*, *structname*, *structmember*, *size*)

Create an array-of-symbols attribute of fixed length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of items in the t_symbol * array.

36.1.4.58 #define CLASS_ATTR_SYM_VARSIZE(*c*, *attrname*, *flags*, *structname*, *structmember*, *sizemember*, *maxsize*)

Create an array-of-symbols attribute of variable length, and add it to a Max class.

Parameters

<i>c</i>	The class pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>sizemember</i>	The actual number of items in the t_symbol * array at any given moment.
<i>maxsize</i>	The maximum number of items in the t_symbol * array, i.e. the number of members allocated for the array in the struct.

36.1.4.59 #define CLASS_METHOD_ATTR_PARSE(*c*, *methodname*, *attrname*, *type*, *flags*, *parsestring*)

Define and add attributes to class methods.

Parameters

<i>c</i>	The class pointer.
<i>methodname</i>	The name of the existing method as a C-string.
<i>attrname</i>	The name of the attribute to add as a C-string.
<i>type</i>	The datatype of the attribute to be added.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestring</i>	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:

```
1 class_addmethod(c, (method)my_foo, "foo", 0);
2 CLASS_METHOD_ATTR_PARSE(c, "foo", "undocumented", gensym("long"), 0, "1");
```

Referenced by [jit_ob3d_setup\(\)](#).

```
36.1.4.60 #define CLASS_METHOD_INTRODUCED( c, methodname, flags, versionstr )
```

Add a new attribute to the specified method to indicate in which version the method was introduced.

Parameters

<i>c</i>	The class pointer.
<i>methodname</i>	The name of the method as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>versionstr</i>	A C-string, which will be parsed set the version number (e.g. "7.0.0").

36.1.4.61 #define CLASS_METHOD_OBSOLETE(*c*, *methodname*, *flags*)

Add a new attribute to the specified method to flag it as obsolete.

Parameters

<i>c</i>	The class pointer.
<i>methodname</i>	The name of the method as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .

36.1.4.62 #define CLASS_METHOD_RENAMED(*c*, *oldname*, *newname*, *flags*)

Add a new attribute to the specified method to flag a method as renamed.

Parameters

<i>c</i>	The class pointer.
<i>oldname</i>	The name of the old method as a C-string.
<i>newname</i>	The name of the new method as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .

36.1.4.63 #define CLASS_STICKY_ATTR(*c*, *name*, *flags*, *parsestr*)

Create an attribute, and add it to all following attribute declarations.

The block is closed by a call to [CLASS_STICKY_ATTR_CLEAR](#).

Parameters

<i>c</i>	The class pointer.
<i>name</i>	The name of the new attribute to create as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	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:

```

1 CLASS_STICKY_ATTR(c, "category", 0, "Foo");
2
3 CLASS_ATTR_DOUBLE(c, "bar", 0, t_myobject, x_bar);
4 CLASS_ATTR_LABEL(c, "bar", 0, "A Bar");
5
6 CLASS_ATTR_CHAR(c, "switch", 0, t_myobject, x_switch);
7 CLASS_ATTR_STYLE_LABEL(c, "switch", 0, "onoff", "Bar Switch");
8
9 CLASS_ATTR_DOUBLE(c, "flow", 0, t_myobject, x_flow);
10 CLASS_ATTR_LABEL(c, "flow", 0, "Flow Amount");
11
12 CLASS_STICKY_ATTR_CLEAR(c, "category");

```

See also

[CLASS_STICKY_ATTR_CLEAR](#)

36.1.4.64 #define CLASS_STICKY_ATTR_CLEAR(*c*, *name*)

Close a [CLASS_STICKY_ATTR](#) block.

Parameters

<i>c</i>	The class pointer.
<i>name</i>	The name of the sticky attribute as a C-string.

See also

[CLASS_STICKY_ATTR](#)

36.1.4.65 #define CLASS_STICKY_METHOD(*c*, *name*, *flags*, *parsestr*)

Create an attribute, and add it to all following method declarations.

The block is closed by a call to [CLASS_STICKY_METHOD_CLEAR](#).

Parameters

<i>c</i>	The class pointer.
<i>name</i>	The name of the new attribute to create as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	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:

```
1 CLASS_STICKY_METHOD(c, "undocumented", 0, "1");
2
3 // add some methods here with class_addmethod()
4 // the undocumented attribute for methods means that the ref-page
5 // generator will ignore these methods.
6
7 CLASS_STICKY_METHOD_CLEAR(c, "undocumented");
```

See also

[CLASS_STICKY_METHOD_CLEAR](#)

36.1.4.66 #define CLASS_STICKY_METHOD_CLEAR(*c*, *name*)

Close a [CLASS_STICKY_METHOD](#) block.

Parameters

<i>c</i>	The class pointer.
<i>name</i>	The name of the sticky attribute as a C-string.

See also

[CLASS_STICKY_METHOD](#)

36.1.4.67 #define OBJ_ATTR_ATOM(*x*, *attrname*, *flags*, *val*)

Create an instance-local [t_atom](#) attribute and add it to a Max class.

Parameters

<i>x</i>	The object pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>val</i>	Pointer to the value.

36.1.4.68 #define OBJ_ATTR_ATOM_ARRAY

Create an instance-local array-of-atoms attribute of fixed length, and add it to the object.

Parameters

<i>x</i>	The object pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>count</i>	The number of items in the t_atom array.
<i>vals</i>	Pointer to the values.

36.1.4.69 #define OBJ_ATTR_CHAR(*x*, *attrname*, *flags*, *val*)

Create an instance-local char attribute and add it to a Max class.

Parameters

<i>x</i>	The object pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>val</i>	Pointer to the value.

36.1.4.70 #define OBJ_ATTR_CHAR_ARRAY(*x*, *attrname*, *flags*, *count*, *vals*)

Create an instance-local array-of-chars attribute of fixed length, and add it to the object.

Parameters

<i>x</i>	The object pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>count</i>	The number of items in the char array.
<i>vals</i>	Pointer to the values.

36.1.4.71 #define OBJ_ATTR_DEFAULT(*x*, *attrname*, *flags*, *parsestr*)

An instance-attribute version of [CLASS_ATTR_DEFAULT](#).

Parameters

<i>x</i>	The t_object instance pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .

<i>parsestr</i>	A C-string, which will be parsed into an array of atoms to set the initial value.
-----------------	---

See also

[CLASS_ATTR_DEFAULT](#)

36.1.4.72 `#define OBJ_ATTR_DEFAULT_SAVE(x, attrname, flags, parsestr)`

An instance-attribute version of [CLASS_ATTR_DEFAULT_SAVE](#).

Parameters

<i>x</i>	The t_object instance pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .
<i>parsestr</i>	A C-string, which will be parsed into an array of atoms to set the initial value.

See also

[CLASS_ATTR_DEFAULT_SAVE](#)

36.1.4.73 `#define OBJ_ATTR_DOUBLE(x, attrname, flags, val)`

Create an instance-local 64bit float attribute and add it to a Max class.

Parameters

<i>x</i>	The object pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>val</i>	Pointer to the value.

36.1.4.74 `#define OBJ_ATTR_DOUBLE_ARRAY(x, attrname, flags, count, vals)`

Create an instance-local array-of-64bit-floats attribute of fixed length, and add it to the object.

Parameters

<i>x</i>	The object pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>count</i>	The number of items in the double array.
<i>vals</i>	Pointer to the values.

36.1.4.75 `#define OBJ_ATTR_FLOAT(x, attrname, flags, val)`

Create an instance-local 32bit float attribute and add it to a Max class.

Parameters

<i>x</i>	The object pointer.
----------	---------------------

<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>val</i>	Pointer to the value.

36.1.4.76 `#define OBJ_ATTR_FLOAT_ARRAY(x, attrname, flags, count, vals)`

Create an instance-local array-of-32bit-floats attribute of fixed length, and add it to the object.

Parameters

<i>x</i>	The object pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>count</i>	The number of items in the float array.
<i>vals</i>	Pointer to the values.

36.1.4.77 `#define OBJ_ATTR_LONG(x, attrname, flags, val)`

Create an instance-local long integer attribute and add it to a Max class.

Parameters

<i>x</i>	The object pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>val</i>	Pointer to the value.

36.1.4.78 `#define OBJ_ATTR_LONG_ARRAY(x, attrname, flags, count, vals)`

Create an instance-local array-of-long-integers attribute of fixed length, and add it to the object.

Parameters

<i>x</i>	The object pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>count</i>	The number of items in the long array.
<i>vals</i>	Pointer to the values.

36.1.4.79 `#define OBJ_ATTR_OBJ(x, attrname, flags, val)`

Create an instance-local `t_object*` attribute and add it to a Max class.

Parameters

<i>x</i>	The object pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>val</i>	Pointer to the value.

36.1.4.80 `#define OBJ_ATTR_OBJ_ARRAY(x, attrname, flags, count, vals)`

Create an instance-local array-of-objects attribute of fixed length, and add it to the object.

Parameters

<i>x</i>	The object pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>count</i>	The number of items in the t_object * array.
<i>vals</i>	Pointer to the values.

36.1.4.81 #define OBJ_ATTR_SAVE(*x*, *attrname*, *flags*)

An instance-attribute version of [CLASS_ATTR_SAVE](#).

Parameters

<i>x</i>	The t_object instance pointer.
<i>attrname</i>	The name of the attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this new attribute, as defined in e_max_attrflags .

See also

[CLASS_ATTR_SAVE](#)

36.1.4.82 #define OBJ_ATTR_SYM(*x*, *attrname*, *flags*, *val*)

Create an instance-local [t_symbol](#)* attribute and add it to a Max class.

Parameters

<i>x</i>	The object pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>val</i>	Pointer to the value.

36.1.4.83 #define OBJ_ATTR_SYM_ARRAY(*x*, *attrname*, *flags*, *count*, *vals*)

Create an instance-local array-of-symbols attribute of fixed length, and add it to the object.

Parameters

<i>x</i>	The object pointer.
<i>attrname</i>	The name of this attribute as a C-string.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>count</i>	The number of items in the t_symbol * array.
<i>vals</i>	Pointer to the values.

36.1.4.84 #define STRUCT_ATTR_ATOM(*c*, *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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.85 #define STRUCT_ATTR_ATOM_ARRAY(*c, 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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of items in the t_atom array.

36.1.4.86 #define STRUCT_ATTR_ATOM_LONG(*c, flags, structname, structmember*)

Create a [t_atom_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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.87 #define STRUCT_ATTR_ATOM_VARSIZE(*c, 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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>sizemember</i>	The actual number of items in the t_atom array at any given moment.
<i>maxsize</i>	The maximum number of items in the t_atom array, i.e. the number of members allocated for the array in the struct.

36.1.4.88 #define STRUCT_ATTR_CHAR(*c, 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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.89 #define STRUCT_ATTR_CHAR_ARRAY(*c*, *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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of items in the char array.

36.1.4.90 #define STRUCT_ATTR_CHAR_VARSIZE(*c*, *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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>sizemember</i>	The actual number of items in the char array at any given moment.
<i>maxsize</i>	The maximum number of items in the char array, i.e. the number of members allocated for the array in the struct.

36.1.4.91 #define STRUCT_ATTR_DOUBLE(*c*, *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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.92 #define STRUCT_ATTR_DOUBLE_ARRAY(*c*, *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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of items in the double array.

36.1.4.93 `#define STRUCT_ATTR_DOUBLE_VARSIZE(c, 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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>sizemember</i>	The actual number of items in the double array at any given moment.
<i>maxsize</i>	The maximum number of items in the double array, i.e. the number of members allocated for the array in the struct.

36.1.4.94 `#define STRUCT_ATTR_FLOAT(c, 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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.95 `#define STRUCT_ATTR_FLOAT_ARRAY(c, 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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

<code>size</code>	The number of items in the floats array.
-------------------	--

36.1.4.96 #define STRUCT_ATTR_FLOAT_VARSIZE(*c, 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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>sizemember</i>	The actual number of items in the float array at any given moment.
<i>maxsize</i>	The maximum number of items in the float array, i.e. the number of members allocated for the array in the struct.

36.1.4.97 #define STRUCT_ATTR_LONG(*c, 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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.98 #define STRUCT_ATTR_LONG_ARRAY(*c, 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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of items in the long array.

36.1.4.99 #define STRUCT_ATTR_LONG_VARSIZE(*c, 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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>sizemember</i>	The actual number of items in the long array at any given moment.
<i>maxsize</i>	The maximum number of items in the long array, i.e. the number of members allocated for the array in the struct.

36.1.4.100 #define STRUCT_ATTR_OBJ(*c*, *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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.101 #define STRUCT_ATTR_OBJ_ARRAY(*c*, *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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of items in the t_object* array.

36.1.4.102 #define STRUCT_ATTR_OBJ_VARSIZE(*c*, *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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

<i>sizemember</i>	The actual number of items in the <code>t_object*</code> array at any given moment.
<i>maxsize</i>	The maximum number of items in the <code>t_object*</code> array, i.e. the number of members allocated for the array in the struct.

36.1.4.103 #define STRUCT_ATTR_SYM(*c*, *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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in <code>e_max_attrflags</code> .
<i>structname</i>	The C identifier for the struct (containing a valid <code>t_object</code> header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.

36.1.4.104 #define STRUCT_ATTR_SYM_ARRAY(*c*, *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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in <code>e_max_attrflags</code> .
<i>structname</i>	The C identifier for the struct (containing a valid <code>t_object</code> header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>size</i>	The number of items in the <code>t_symbol*</code> array.

36.1.4.105 #define STRUCT_ATTR_SYM_VARSIZE(*c*, *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

<i>c</i>	The class pointer.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in <code>e_max_attrflags</code> .
<i>structname</i>	The C identifier for the struct (containing a valid <code>t_object</code> header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>sizemember</i>	The actual number of items in the <code>t_symbol*</code> array at any given moment.
<i>maxsize</i>	The maximum number of items in the <code>t_symbol*</code> array, i.e. the number of members allocated for the array in the struct.

36.1.5 Enumeration Type Documentation

36.1.5.1 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.

36.1.6 Function Documentation**36.1.6.1 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

<i>x</i>	Pointer to the attribute to receive the filter
<i>min</i>	Minimum value for the clip filter
<i>max</i>	Maximum value for the clip filter
<i>usemin</i>	Sets this value to 0 if the minimum clip value should <i>not</i> be used. Otherwise, set the value to non-zero.
<i>usemax</i>	Sets this value to 0 if the maximum clip value should <i>not</i> 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.

36.1.6.2 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

<i>x</i>	Pointer to the attribute to receive the filter
<i>scale</i>	Scale value. Data sent to the attribute will be scaled by this amount. Data retrieved from the attribute will be scaled by its reciprocal. <i>Scaling occurs previous to clipping.</i>
<i>min</i>	Minimum value for the clip filter
<i>max</i>	Maximum value for the clip filter
<i>usemin</i>	Sets this value to 0 if the minimum clip value should <i>not</i> be used. Otherwise, set the value to non-zero.
<i>usemax</i>	Sets this value to 0 if the maximum clip value should <i>not</i> 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.

36.1.6.3 `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

<i>x</i>	Pointer to the attribute to receive the filter
<i>min</i>	Minimum value for the clip filter
<i>max</i>	Maximum value for the clip filter
<i>usemin</i>	Sets this value to 0 if the minimum clip value should <i>not</i> be used. Otherwise, set the value to non-zero.
<i>usemax</i>	Sets this value to 0 if the maximum clip value should <i>not</i> 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.

36.1.6.4 `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

<i>x</i>	Pointer to the attribute to receive the filter
<i>scale</i>	Scale value. Data retrieved from the attribute will be scaled by this amount. <i>Scaling occurs previous to clipping.</i>
<i>min</i>	Minimum value for the clip filter
<i>max</i>	Maximum value for the clip filter
<i>usemin</i>	Sets this value to 0 if the minimum clip value should <i>not</i> be used. Otherwise, set the value to non-zero.
<i>usemax</i>	Sets this value to 0 if the maximum clip value should <i>not</i> 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.

36.1.6.5 `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

<i>x</i>	Pointer to the attribute to receive the filter
<i>proc</i>	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.

36.1.6.6 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

<i>x</i>	Pointer to the attribute to receive the filter
<i>min</i>	Minimum value for the clip filter
<i>max</i>	Maximum value for the clip filter
<i>usemin</i>	Sets this value to 0 if the minimum clip value should <i>not</i> be used. Otherwise, set the value to non-zero.
<i>usemax</i>	Sets this value to 0 if the maximum clip value should <i>not</i> 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.

36.1.6.7 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

<i>x</i>	Pointer to the attribute to receive the filter
<i>scale</i>	Scale value. Data sent to the attribute will be scaled by this amount. <i>Scaling occurs previous to clipping.</i>
<i>min</i>	Minimum value for the clip filter
<i>max</i>	Maximum value for the clip filter
<i>usemin</i>	Sets this value to 0 if the minimum clip value should <i>not</i> be used. Otherwise, set the value to non-zero.
<i>usemax</i>	Sets this value to 0 if the maximum clip value should <i>not</i> 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.

36.1.6.8 `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

<code>x</code>	Pointer to the attribute to receive the filter
<code>proc</code>	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:

```

1 t_max_err myfiltermethod(void *parent, void *attr, long ac, t_atom *av);
2
3 t_max_err myfiltermethod(void *parent, void *attr, long ac, t_atom *av)
4 {
5     long i;
6     float temp;
7
8     // this filter rounds off all values
9     // assumes that the data is float
10    for (i = 0; i < ac; i++) {
11        temp = atom_getfloat(av + i);
12        temp = (float)((long)(temp + 0.5));
13        atom_setfloat(av + i, temp);
14    }
15    return MAX_ERR_NONE;
16 }
```

36.1.6.9 `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

<code>x</code>	A dictionary instance pointer.
<code>ac</code>	The number of atoms to parse in <code>av</code> .
<code>av</code>	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\(\)](#).

```

1 long ac = 0;
2 t_atom *av = NULL;
3 char parsebuf[4096];
4 t_dictionary *d = dictionary_new();
5 t_atom a;
6
7 sprintf(parsebuf, "@defrect %.6f %.6f %.6f %.6f @title Untitled @presentation 0 ", r->x, r->y, r->width,
8         r->height);
8 atom_setparse(&ac, &av, parsebuf);
9 attr_args_dictionary(d, ac, av);
10 atom_setobj(&a, d);
```

36.1.6.10 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

<i>ac</i>	The count of <i>t_atoms</i> in <i>av</i>
<i>av</i>	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).

Referenced by `max_jit_attr_args_offset()`.

36.1.6.11 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

<i>x</i>	The object whose attributes will be processed
<i>ac</i>	The count of <i>t_atoms</i> in <i>av</i>
<i>av</i>	An atom list

Remarks

Here is a typical example of usage:

```

1 void *myobject_new(t_symbol *s, long ac, t_atom *av)
2 {
3     t_myobject *x = NULL;
4
5     if (x=(t_myobject *)object_alloc(myobject_class))
6     {
7         // initialize any data before processing
8         // attributes to avoid overwriting
9         // attribute argument-set values
10        x->data = 0;
11
12        // process attr args, if any
13        attr_args_process(x, ac, av);
14    }
15    return x;
16 }
```

36.1.6.12 void attr_dictionary_check (void * *x*, *t_dictionary* * *d*)

Check that a dictionary only contains values for existing attributes of an object.

If a key in the dictionary doesn't correspond an one of the object's attributes, an error will be posted to the Max window.

Parameters

<i>x</i>	The object instance pointer.
<i>d</i>	The dictionary containing the attributes.

See also

[attr_dictionary_process\(\)](#)

36.1.6.13 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\(\)](#)

36.1.6.14 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

<i>name</i>	A name for the attribute, as a C-string
<i>type</i>	A t_symbol * representing a valid attribute type. At the time of this writing, the valid type-symbols are: <code>_sym_char</code> (char), <code>_sym_long</code> (long), <code>_sym_float32</code> (32-bit float), <code>↔_sym_float64</code> (64-bit float), <code>_sym_atom</code> (Max t_atom pointer), <code>_sym_symbol</code> (Max t_symbol pointer), <code>_sym_pointer</code> (generic pointer) and <code>_sym_object</code> (Max t_object pointer).
<i>size</i>	Maximum number of items that may be in the array.
<i>flags</i>	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 .
<i>mget</i>	The method to use for the attribute's get functionality. If <i>mget</i> is NULL, the default method is used. See the discussion under attribute_new() , for more information.
<i>mset</i>	The method to use for the attribute's set functionality. If <i>mset</i> is NULL, the default method is used. See the discussion under attribute_new() , for more information.
<i>offsetcount</i>	Byte offset into the object class's data structure of a long variable describing how many array elements (up to <i>size</i>) comprise the data to be referenced by the attribute. Typically, the calcoffset macro is used to calculate this offset.
<i>offset</i>	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:

```
1 t_object *attr = attr_offset_array_new("myattrarray", _sym_atom / * matches data size * /, 10 / * max * /,
0 / * no flags * /, (method)OL, (method)OL, calcoffset(t_myobject, atmcount) / * count * /,
calcoffset(t_myobject, atm) / * data * /);
```

Referenced by [ext_main\(\)](#).

36.1.6.15 `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

<code>name</code>	A name for the attribute, as a C-string
<code>type</code>	A <code>t_symbol *</code> representing a valid attribute type. At the time of this writing, the valid type-symbols are: <code>_sym_char</code> (<code>char</code>), <code>_sym_long</code> (<code>long</code>), <code>_sym_float32</code> (32-bit float), <code>↔_sym_float64</code> (64-bit float), <code>_sym_atom</code> (Max <code>t_atom</code> pointer), <code>_sym_symbol</code> (Max <code>t_symbol</code> pointer), <code>_sym_pointer</code> (generic pointer) and <code>_sym_object</code> (Max <code>t_object</code> pointer).
<code>flags</code>	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 <code>e_max_↔attrflags</code> .
<code>mget</code>	The method to use for the attribute's get functionality. If <code>mget</code> is NULL, the default method is used. See the discussion under <code>attribute_new()</code> , for more information.
<code>mset</code>	The method to use for the attribute's set functionality. If <code>mset</code> is NULL, the default method is used. See the discussion under <code>attribute_new()</code> , for more information.
<code>offset</code>	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 <code>calcoffset</code> 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:

```
1 t_object *attr = attr_offset_new("myattr", _sym_float64 / * matches data size * /, 0 / * no flags * /,
                                (method)0L, (method)0L, calcoffset(t_myobject, val));
```

Referenced by `ext_main()`.

36.1.6.16 `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

<i>name</i>	A name for the attribute, as a C-string
<i>type</i>	A <code>t_symbol</code> * representing a valid attribute type. At the time of this writing, the valid type-symbols are: <code>_sym_char</code> (char), <code>_sym_long</code> (long), <code>_sym_float32</code> (32-bit float), <code>_sym_float64</code> (64-bit float), <code>_sym_atom</code> (Max <code>t_atom</code> pointer), <code>_sym_symbol</code> (Max <code>t_symbol</code> pointer), <code>_sym_pointer</code> (generic pointer) and <code>_sym_object</code> (Max <code>t_object</code> pointer).
<i>flags</i>	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 <code>e_max_attrflags</code> .
<i>mget</i>	The method to use for the attribute's get functionality. If <i>mget</i> is NULL, the default method is used.
<i>mset</i>	The method to use for the attribute's set functionality. If <i>mset</i> 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:

```
1 t_max_err myobject_myattr_get(t_myobject *x, void *attr, long *ac, t_atom **av);
1 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:

```
1 t_max_err myobject_myattr_get(t_myobject *x, void *attr, long *ac, t_atom **av)
2 {
3     if (*ac && *av)
4         // memory passed in; use it
5     else {
6         *ac = 1; // size of attr data
7         *av = (t_atom *)getbytes(sizeof(t_atom) * (*ac));
8         if (!(*av)) {
9             *ac = 0;
10            return MAX_ERR_OUT_OF_MEM;
11        }
12    }
13    atom_setlong(*av, x->some_value);
14    return MAX_ERR_NONE;
15 }
16
17 t_max_err myobject_myattr_set(t_myobject *x, void *attr, long ac, t_atom *av)
18 {
19     if (ac && av) {
20         x->some_value = atom_getlong(av);
21     }
22     return MAX_ERR_NONE;
23 }
```

36.1.6.17 `t_max_err object_addattr(void * x, t_object * attr)`

Attaches an attribute directly to an object.

Parameters

<i>x</i>	An object to which the attribute should be attached
<i>attr</i>	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.

36.1.6.18 void* object_attr_get(void * *x*, t_symbol * *attrname*)

Returns the pointer to an attribute, given its name.

Parameters

<i>x</i>	Pointer to the object whose attribute is of interest
<i>attrname</i>	The attribute's name

Returns

This function returns a pointer to the attribute, if successful, or NULL, if unsuccessful.

Referenced by `jit_attr_getchar_array()`, `jit_attr_getdouble_array()`, `jit_attr_getfloat()`, `jit_attr_getfloat_array()`, `jit_attr_getlong()`, `jit_attr_getlong_array()`, `jit_attr_getsym()`, `jit_attr_getsym_array()`, `jit_attr_setchar_array()`, `jit_attr_setdouble_array()`, `jit_attr_setfloat()`, `jit_attr_setfloat_array()`, `jit_attr_setlong()`, `jit_attr_setlong_array()`, `jit_attr_setsym()`, `jit_attr_setsym_array()`, `jit_object_attr_get()`, `jit_object_exportattrs()`, and `max_jit_attr_set()`.

36.1.6.19 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

<i>o</i>	The attribute's parent object
<i>name</i>	The attribute's name
<i>rect</i>	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.

36.1.6.20 long object_attr_getchar_array(void * *x*, t_symbol * *s*, long *max*, t_uint8 * *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

<i>x</i>	The attribute's parent object
<i>s</i>	The attribute's name
<i>max</i>	The number of array elements in <i>vals</i> . The function will take care not to overwrite the bounds of the array.
<i>vals</i>	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.

Referenced by `jit_attr_getchar_array()`.

36.1.6.21 `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

<code>b</code>	The attribute's parent object
<code>attrname</code>	The attribute's name
<code>prgba</code>	The address of a valid <code>t_jrgba</code> 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.

36.1.6.22 `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

<code>x</code>	The attribute's parent object
<code>s</code>	The attribute's name
<code>max</code>	The number of array elements in <code>vals</code> . The function will take care not to overwrite the bounds of the array.
<code>vals</code>	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.

Referenced by `jit_attr_getdouble_array()`.

36.1.6.23 `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

<i>x</i>	Pointer to the object whose attribute is of interest
<i>s</i>	The attribute's name
<i>argc</i>	Unused
<i>argv</i>	Unused

36.1.6.24 *t_atom_float* *object_attr_getfloat*(*void* * *x*, *t_symbol* * *s*)

Retrieves the value of an attribute, given its parent object and name.

Parameters

<i>x</i>	The attribute's parent object
<i>s</i>	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.

Referenced by *jit_attr_getfloat()*.

36.1.6.25 *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

<i>x</i>	The attribute's parent object
<i>s</i>	The attribute's name
<i>max</i>	The number of array elements in <i>vals</i> . The function will take care not to overwrite the bounds of the array.
<i>vals</i>	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.

Referenced by *jit_attr_getfloat_array()*.

36.1.6.26 *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

<i>ob</i>	The attribute's parent object
<i>s</i>	The attribute's name
<i>c</i>	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.

36.1.6.27 t_atom_long object_attr_getlong (void *x, t_symbol *s)

Retrieves the value of an attribute, given its parent object and name.

Parameters

<i>x</i>	The attribute's parent object
<i>s</i>	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.

Referenced by [jit_attr_getlong\(\)](#).

36.1.6.28 long object_attr_getlong_array (void *x, t_symbol *s, long max, t_atom_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

<i>x</i>	The attribute's parent object
<i>s</i>	The attribute's name
<i>max</i>	The number of array elements in <i>vals</i> . The function will take care not to overwrite the bounds of the array.
<i>vals</i>	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.

Referenced by [jit_attr_getlong_array\(\)](#).

36.1.6.29 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

<i>o</i>	The attribute's parent object
<i>name</i>	The attribute's name
<i>pt</i>	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.

36.1.6.30 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

<i>o</i>	The attribute's parent object
<i>name</i>	The attribute's name
<i>size</i>	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.

36.1.6.31 t_symbol* object_attr_getsym(void *x, t_symbol *s)

Retrieves the value of an attribute, given its parent object and name.

Parameters

<i>x</i>	The attribute's parent object
<i>s</i>	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.

Referenced by `jit_attr_getsym()`.

36.1.6.32 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

<i>x</i>	The attribute's parent object
<i>s</i>	The attribute's name
<i>max</i>	The number of array elements in <i>vals</i> . The function will take care not to overwrite the bounds of the array.
<i>vals</i>	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`.

Referenced by `jit_attr_getsym_array()`.

36.1.6.33 `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

<code>x</code>	Pointer to the object whose attribute is of interest
<code>methodname</code>	The Max message used to call the attribute's get or set function. For example, <code>gensym("mode")</code> or <code>gensym("getthresh")</code> .
<code>attr</code>	A pointer to a void *, which will be set to the attribute pointer upon successful completion of the function
<code>get</code>	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.

36.1.6.34 `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

<code>o</code>	The attribute's parent object
<code>name</code>	The attribute's name
<code>rect</code>	The address of a valid <code>t_rect</code> 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.

36.1.6.35 `void object_attr_set_xywh (t_object * o, t_symbol * attr, double x, double y, double w, double h)`

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

<code>o</code>	The attribute's parent object
<code>attr</code>	The attribute's name
<code>x</code>	A double containing the new x position.
<code>y</code>	A double containing the new y position.
<code>w</code>	A double containing the new width.
<code>h</code>	A double containing the new height.

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.

36.1.6.36 t_max_err object_attr_setchar_array (void *x, t_symbol *s, long count, C74_CONST t_uint8 *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

<i>x</i>	The attribute's parent object
<i>s</i>	The attribute's name
<i>count</i>	The number of array elements in <i>vals</i>
<i>vals</i>	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.

Referenced by `jit_attr_setchar_array()`.

36.1.6.37 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

<i>b</i>	The attribute's parent object
<i>attrname</i>	The attribute's name
<i>prgba</i>	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.

36.1.6.38 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

<i>x</i>	The attribute's parent object
<i>s</i>	The attribute's name
<i>count</i>	The number of array elements in <i>vals</i>
<i>vals</i>	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.

Referenced by `jit_attr_setdouble_array()`.

36.1.6.39 `t_max_err object_attr_setfloat(void *x, t_symbol *s, t_atom_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

<code>x</code>	The attribute's parent object
<code>s</code>	The attribute's name
<code>c</code>	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.

Referenced by `jit_attr_setfloat()`.

36.1.6.40 `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

<code>x</code>	The attribute's parent object
<code>s</code>	The attribute's name
<code>count</code>	The number of array elements in <code>vals</code>
<code>vals</code>	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.

Referenced by `jit_attr_setfloat_array()`.

36.1.6.41 `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

<code>ob</code>	The attribute's parent object
<code>s</code>	The attribute's name
<code>c</code>	The address of a <code>t_jrgba</code> 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.

36.1.6.42 `t_max_err object_attr_setlong(void *x, t_symbol *s, t_atom_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

<i>x</i>	The attribute's parent object
<i>s</i>	The attribute's name
<i>c</i>	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.

Referenced by `jit_attr_setlong()`.

36.1.6.43 `t_max_err object_attr_setlong_array (void *x, t_symbol *s, long count, t_atom_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

<i>x</i>	The attribute's parent object
<i>s</i>	The attribute's name
<i>count</i>	The number of array elements in <i>vals</i>
<i>vals</i>	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.

Referenced by `jit_attr_setlong_array()`.

36.1.6.44 `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

<i>x</i>	The object whose attribute will be set.
<i>s</i>	The name of the attribute to set.
<i>parsestr</i>	A C-string to parse into an array of atoms to set the attribute value.

Returns

A Max error code.

See also

[atom_setparse\(\)](#)

36.1.6.45 `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

<i>o</i>	The attribute's parent object
<i>name</i>	The attribute's name
<i>pt</i>	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.

36.1.6.46 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

<i>o</i>	The attribute's parent object
<i>name</i>	The attribute's name
<i>size</i>	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.

36.1.6.47 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

<i>x</i>	The attribute's parent object
<i>s</i>	The attribute's name
<i>c</i>	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.

Referenced by `jit_attr_setsym()`.

36.1.6.48 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

<i>x</i>	The attribute's parent object
----------	-------------------------------

<i>s</i>	The attribute's name
<i>count</i>	The number of array elements in <i>vals</i>
<i>vals</i>	Pointer to the first element of an array of <code>t_symbol</code> * <i>s</i>

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.

Referenced by `jit_attr_setsym_array()`.

36.1.6.49 `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

<i>x</i>	Pointer to the object whose attribute is of interest
<i>s</i>	The attribute's name
<i>argc</i>	The count of arguments in <i>argv</i>
<i>argv</i>	Array of <code>t_atoms</code> ; 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.

Referenced by `jit_object_importattrs()`.

36.1.6.50 `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

<i>x</i>	Pointer to the object whose attribute is of interest
<i>s</i>	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.

Referenced by `jit_object_attr_usercanget()`.

36.1.6.51 `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

<i>x</i>	Pointer to the object whose attribute is of interest
<i>s</i>	The attribute's name

Returns

This function returns 1 if the attribute can be set from the Max interface. Otherwise, it returns 0.

Referenced by [jit_object_attr_usercanset\(\)](#).

36.1.6.52 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

<i>x</i>	The object to which the attribute is attached
<i>attrsym</i>	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.

36.1.6.53 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

<i>x</i>	The object to which the attribute is attached
<i>attrsym</i>	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.

36.1.6.54 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

<i>name_space</i>	The namespace in which to create the instance. Typically this is either CLASS_BOX or CLASS_NOBOX .
<i>classname</i>	The name of the class to instantiate.
<i>parsestr</i>	A C-string to parse into an array of atoms to set the attribute value.

Returns

A pointer to the new instance.

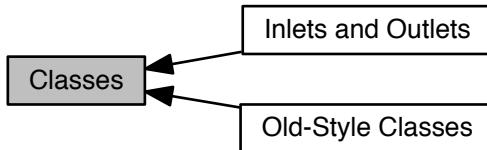
See also

[atom_setparse\(\)](#)
[object_new_typed\(\)](#)

36.2 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 `ext_main()` function.

Collaboration diagram for Classes:



Modules

- [Old-Style Classes](#)
- [Inlets and Outlets](#)

Routines for creating and communicating with inlets and outlets.

Data Structures

- struct [t_class](#)

The data structure for a Max class.

Macros

- `#define CLASS_BOX`

The namespace for all Max object classes which can be instantiated in a box, i.e.

- `#define CLASS_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, CLASS_FLAG_POLYGLOT, CLASS_FLAG_NEWDICTIONARY, CLASS_FLAG_REGISTERED,`
`CLASS_FLAG_UOBJECT, CLASS_FLAG_ALIAS, CLASS_FLAG_DO_NOT_PARSE_ATTR_ARGS, CLASS_FLAG_DO_NOT_ZERO,`
`CLASS_FLAG_NOATTRIBUTES, CLASS_FLAG_OWNATTRIBUTES, CLASS_FLAG_PARAMETER, CLASS_FLAG_RETYPABLE,`
`CLASS_FLAG_OBJECT_METHOD }`

Class flags.

Functions

- **BEGIN_USING_C_LINKAGE void C74_EXPORT ext_main (void *r)**
ext_main() is the entry point for an extern to be loaded, which all externs must implement this shared/common prototype ensures that it will be exported correctly on all platforms.
- **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.
- **t_max_err class_subclass (t_class *superclass, t_class *subclass)**
Define a subclass of an existing class.
- **t_object * class_super_construct (t_class *c,...)**
Call super class constructor.

36.2.1 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 `ext_main()` function.

Thus, Max classes are typically defined in the `ext_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.

36.2.2 Macro Definition Documentation

36.2.2.1 #define CLASS_BOX

The namespace for all Max object classes which can be instantiated in a box, i.e. in a patcher.

Referenced by ext_main().

36.2.3 Enumeration Type Documentation

36.2.3.1 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_UOBJECT 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_DO_NOT_PARSE_ATTR_ARGS override dictionary based constructor attr arg parsing

CLASS_FLAG_DO_NOT_ZERO don't zero the object struct on construction (for efficiency)

CLASS_FLAG_NOATTRIBUTES for efficiency

CLASS_FLAG_OWNATTRIBUTES for classes which support a custom attr interface (e.g. jitter)

CLASS_FLAG_PARAMETER for classes which have a parameter

CLASS_FLAG RETYPEABLE object box can be retyped without recreating the object

CLASS_FLAG_OBJECT_METHOD objects of this class may have object specific methods

36.2.4 Function Documentation

36.2.4.1 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.

Referenced by ext_main(), and jit_class_addattr().

36.2.4.2 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

<i>c</i>	The class pointer
<i>m</i>	Function to be called when the method is invoked
<i>name</i>	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.

Referenced by [ext_main\(\)](#), [jit_class_addmethod\(\)](#), [jit_class_new\(\)](#), and [max_jit_class_ob3d_wrap\(\)](#).

36.2.4.3 t_max_err class_alias (t_class * c, t_symbol * aliasname)

Registers an alias for a previously defined object class.

Parameters

<i>c</i>	The class pointer
<i>aliasname</i>	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.

36.2.4.4 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

<i>c</i>	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.

36.2.4.5 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

<i>name_space</i>	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.
<i>classname</i>	The name of the class to be looked up

Returns

If successful, this function returns the class's data pointer. Otherwise, it returns NULL.

Referenced by [jit_class_findbyname\(\)](#).

36.2.4.6 `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

<i>name_space</i>	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.
<i>classname</i>	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.

36.2.4.7 `t_max_err class_free (t_class * c)`

Frees a previously defined object class.

This function is not typically used by external developers.

Parameters

<i>c</i>	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.

Referenced by [jit_class_free\(\)](#).

36.2.4.8 `long class_is_ui (t_class * c)`

Determine if a class is a user interface object.

Parameters

<i>c</i>	The class pointer.
----------	--------------------

Returns

True is the class defines a user interface object, otherwise false.

36.2.4.9 `t_symbol* class_nameget(t_class * c)`

Retrieves the name of a class, given the class's pointer.

Parameters

<code>c</code>	The class pointer
----------------	-------------------

Returns

If successful, this function returns the name of the class as a `t_symbol *`.

Referenced by `jit_class_nameget()`.

36.2.4.10 `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

<code>name</code>	The class's name, as a C-string
<code>mnew</code>	The instance creation function
<code>mfree</code>	The instance free function
<code>size</code>	The size of the object's data structure in bytes. Usually you use the C <code>sizeof</code> operator here.
<code>mmenu</code>	Obsolete - pass NULL. In Max 4 this was a function pointer for UI objects called when the user created a new object of the class from the Patch window's palette.
<code>type</code>	A standard Max <i>type list</i> 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. <i>Generally, obex objects have a single type argument, A_GIMME, followed by a 0.</i>

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.*

Referenced by `ext_main()`, and `jit_class_new()`.

36.2.4.11 `long class_obexoffset_get(t_class * c)`

Retrieves the byte-offset of the obex member of the class's data structure.

Parameters

<code>c</code>	The class pointer
----------------	-------------------

Returns

This function returns the byte-offset of the obex member of the class's data structure.

36.2.4.12 `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

<i>c</i>	The class pointer
<i>offset</i>	The byte-offset to the obex member of the object's data structure. Conventionally, the macro calcoffset is used to calculate the offset.

Referenced by `ext_main()`.

36.2.4.13 `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

<i>name_space</i>	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.
<i>c</i>	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.

Referenced by `ext_main()`, and `jit_class_register()`.

36.2.4.14 `t_max_err class_subclass (t_class * superclass, t_class * subclass)`

Define a subclass of an existing class.

First call `class_new` on the subclass, then pass it to `class_subclass`. If constructor or destructor are NULL will use the superclass constructor.

Parameters

<i>superclass</i>	The superclass pointer.
<i>subclass</i>	The subclass pointer.

Returns

A Max error code

36.2.4.15 `t_object* class_super_construct (t_class * c, ...)`

Call super class constructor.

Use this instead of `object_alloc` if you want to call the super class constructor, but allocating enough memory for subclass.

Parameters

<i>c</i>	The (sub)class pointer.
----------	-------------------------

...	Args to super class constructor.
-----	----------------------------------

Returns

initialized object instance

36.2.4.16 BEGIN_USING_C_LINKAGE void C74_EXPORT ext_main(void * r)

`ext_main()` is the entry point for an extern to be loaded, which all externs must implement this shared/common prototype ensures that it will be exported correctly on all platforms.

Parameters

r	Pointer to resources for the external, if applicable.
---	---

See also

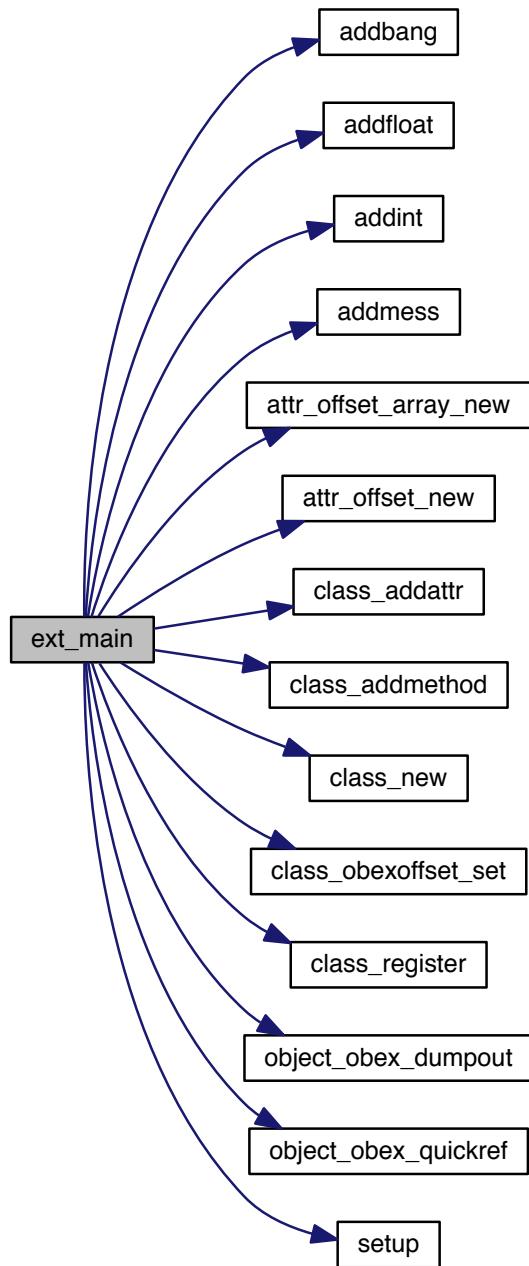
[Anatomy of a Max Object](#)

Version

Introduced in Max 6.1.9

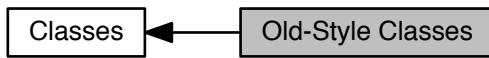
References A_CANT, A_FLOAT, A_GIMME, A_LONG, addbang(), addfloat(), addint(), addmess(), attr_offset_←array_new(), attr_offset_new(), ATTR_SET_OPAQUE, ATTR_SET_OPAQUE_USER, calcoffset, class_addattr(), class_addmethod(), CLASS_BOX, class_new(), class_obexoffset_set(), class_register(), object_obex_dumpout(), object_obex_quickref(), and setup().

Here is the call graph for this function:



36.3 Old-Style Classes

Collaboration diagram for Old-Style Classes:



Functions

- **BEGIN_USING_C_LINKAGE void setup (t_messlist **ident, method makefun, method freefun, t_getbytes→_size 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 address (method f, char *s, short type,...)**

Use [address\(\)](#) 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.

36.3.1 Detailed Description

36.3.2 Function Documentation

36.3.2.1 void addbang (method *f*)

Used to bind a function to the common triggering message bang.

Parameters

<i>f</i>	Function to be the bang method.
----------	---------------------------------

Referenced by ext_main(), max_jit_classex_mop_wrap(), and max_ob3d_setup().

36.3.2.2 void addfloat (method *f*)

Use [addfloat\(\)](#) to bind a function to the float message received in the leftmost inlet.

Parameters

<i>f</i>	Function to be the int method.
----------	--------------------------------

Referenced by ext_main().

36.3.2.3 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

<i>f</i>	Function to be the float method.
<i>n</i>	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](#).

36.3.2.4 void addint (method *f*)

Use [addint\(\)](#) to bind a function to the int message received in the leftmost inlet.

Parameters

<i>f</i>	Function to be the int method.
----------	--------------------------------

Referenced by ext_main().

36.3.2.5 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

<i>f</i>	Function to be the int method.
<i>n</i>	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](#).

36.3.2.6 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

<i>f</i>	Function you want to be the method.
<i>s</i>	C string defining the message.
<i>type</i>	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](#)

Referenced by `ext_main()`, `max_addmethod_defer()`, `max_addmethod_defer_low()`, `max_addmethod_usurp()`, `max_addmethod_usurp_low()`, `max_jit_classex_addattr()`, `max_jit_classex_mop_wrap()`, `max_jit_classex_setup()`, `max_jit_classex_standard_wrap()`, and `max_ob3d_setup()`.

36.3.2.7 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

<i>name</i>	An alternative name for the user to use to make an object of your class.
-------------	--

36.3.2.8 void class_setname (char * *obname*, char * *filename*)

Use [class_setname\(\)](#) to associate your object's name with its filename on disk.

Parameters

<i>obname</i>	A character string with the name of your object class as it appears in Max.
<i>filename</i>	A character string with the name of your external's file as it appears on disk.

36.3.2.9 method egetfn (t_object * *op*, t_symbol * *msg*)

Use [egetfn\(\)](#) to send an untyped message to a Max object that always works.

Parameters

<i>op</i>	Receiver of the message.
<i>msg</i>	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.

36.3.2.10 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, Exprs, etc. should be freed with `freeobject()`.

Parameters

<i>op</i>	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\(\)](#)

Referenced by `max_jit_obex_gimmeback()`, and `max_jit_obex_gimmeback_dumpout()`.

36.3.2.11 method getfn (t_object * *op*, t_symbol * *msg*)

Use `getfn()` to send an untyped message to a Max object with error checking.

Parameters

<i>op</i>	Receiver of the message.
<i>msg</i>	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.

36.3.2.12 void* newinstance (t_symbol * *s*, short *argc*, t_atom * *argv*)

Make a new instance of an existing Max class.

Parameters

<i>s</i>	ClassName Symbol specifying the name of the class of the instance to be created.
<i>argc</i>	Count of arguments in argv.
<i>argv</i>	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.

36.3.2.13 void* newobject (void * *maxclass*)

Use newobject to allocate the space for an instance of your class and initialize its object header.

Parameters

<i>maxclass</i>	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.

Referenced by `max_jit_obex_new()`.

36.3.2.14 BEGIN_USING_C_LINKAGE void setup (t_messlist ** *ident*, method *makefun*, method *freefun*, t_getbytes_size *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

<i>ident</i>	A global variable in your code that points to the initialized class.
<i>makefun</i>	Your instance creation function.
<i>freefun</i>	Your instance free function (see Chapter 7).

<i>size</i>	The size of your objects data structure in bytes. Usually you use the C sizeof operator here.
<i>menufun</i>	No longer used. You should pass NULL for this parameter.
<i>type</i>	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](#)

Referenced by `ext_main()`.

36.3.2.15 void* typedmess (*t_object* * *op*, *t_symbol* * *msg*, short *argc*, *t_atom* * *argp*)

Send a typed message directly to a Max object.

Parameters

<i>op</i>	Max object that will receive the message.
<i>msg</i>	The message selector.
<i>argc</i>	Count of message arguments in <i>argv</i> .
<i>argp</i>	Array of <i>t_atoms</i> ; 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:

```
1 //If you want to send a bang message to the object bang_me...
2 void *bangResult;
3 bangResult = typedmess(bang_me, gensym("bang"), 0, 0L);
```

Referenced by `max_jit_mop_bang()`, and `max_ob3d_bang()`.

36.3.2.16 method zgetfn (*t_object* * *op*, *t_symbol* * *msg*)

Use [zgetfn\(\)](#) to send an untyped message to a Max object without error checking.

Parameters

<i>op</i>	Receiver of the message.
<i>msg</i>	Message selector.

Returns

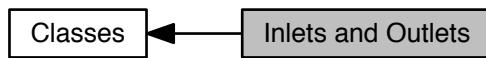
`zgetfn` returns a pointer to the method bound to the message selector *msg* in the receiver's message list. It returns 0 but doesn't print an error message in Max Window if the method can't be found.

Referenced by `max_jit_attr_getdump()`.

36.4 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, t_atom_long n)
Use [outlet_int\(\)](#) to send an int message out an outlet.
- void * [outlet_float](#) (void *o, double f)
Use [outlet_float\(\)](#) to send a float 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.

36.4.1 Detailed Description

Routines for creating and communicating with inlets and outlets.

36.4.2 Function Documentation

36.4.2.1 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.

36.4.2.2 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.

36.4.2.3 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.

36.4.2.4 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

```
1 inlet_new (myObject, "bang");
```

To create an inlet that can receive any message, pass NULL for msg

```
1 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](#).

36.4.2.5 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 have the highest number in- or ft- message (which is usually the case), you should create the highest number message inlet first.

36.4.2.6 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.

36.4.2.7 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.

36.4.2.8 void* outlet_anything (void * o, t_symbol * s, short ac, t_atom * av)

Use [outlet_anything\(\)](#) to send any message out an outlet.

Parameters

<i>o</i>	Outlet that will send the message.
<i>s</i>	The message selector t_symbol *.
<i>ac</i>	Number of elements in the list in argv.
<i>av</i>	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):

```
1 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\(\)](#)):

```
1 t_atom myNumber;
2
3 atom_setlong(&myNumber, 432);
4 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.

Referenced by `max_jit_mop_jit_matrix()`, `max_jit_mop_outputmatrix()`, and `max_jit_obex_dumpout()`.

36.4.2.9 void* outlet_bang (void * o)

Use [outlet_bang\(\)](#) to send a bang message out an outlet.

Parameters

<i>o</i>	Outlet that will send the message.
----------	------------------------------------

Returns

Returns 0 if a stack overflow occurred, otherwise returns 1.

36.4.2.10 void* outlet_float (void * *o*, double *f*)

Use [outlet_float\(\)](#) to send a float message out an outlet.

Parameters

<i>o</i>	Outlet that will send the message.
<i>f</i>	Float value to send.

Returns

Returns 0 if a stack overflow occurred, otherwise returns 1.

36.4.2.11 void* outlet_int (void * *o*, t_atom_long *n*)

Use [outlet_int\(\)](#) to send an int message out an outlet.

Parameters

<i>o</i>	Outlet that will send the message.
<i>n</i>	Integer value to send.

Returns

Returns 0 if a stack overflow occurred, otherwise returns 1.

36.4.2.12 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

<i>o</i>	Outlet that will send the message.
<i>s</i>	Should be NULL, but can be the _sym_list.
<i>ac</i>	Number of elements in the list in argv.
<i>av</i>	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.

```

1 t_atom myList[3];
2 long theNumbers[3];
3 short i;
4
5 theNumbers[0] = 23;
6 theNumbers[1] = 12;
7 theNumbers[2] = 5;
8 for (i=0; i < 3; i++) {
9     atom_setlong(myList+i, theNumbers[i]);
10 }
11 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.

36.4.2.13 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

<code>x</code>	Your object.
<code>s</code>	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.

Referenced by `max_jit_mop_matrixout_new()`, and `max_jit_mop_setup_simple()`.

36.4.2.14 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

<code>master</code>	Your object.
---------------------	--------------

Returns

The index number of the inlet that received the message.

Referenced by `max_jit_obex_inletnumber_get()`.

36.4.2.15 void* proxy_new (void * x, long id, long * stuffloc)

Use `proxy_new` to create a new Proxy object.

Parameters

<i>x</i>	Your object.
<i>id</i>	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.
<i>stuffloc</i>	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\(\)](#).

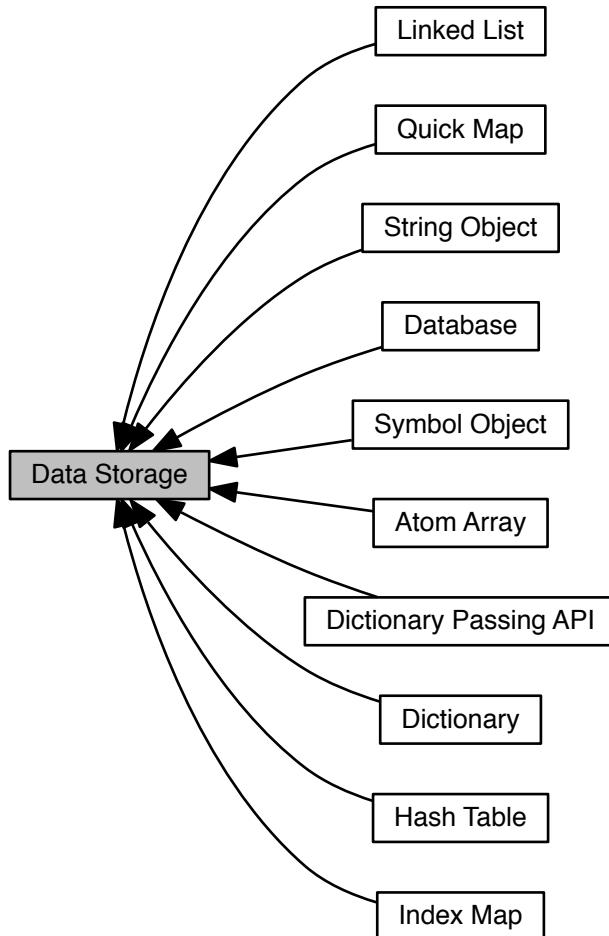
After your method has finished, Proxy sets the stuffLoc location back to 0, since it never sees messages coming in an object's leftmost inlet. You'll know you received a message in the leftmost inlet if the contents of stuffLoc is 0. As of Max 4.3, stuffLoc is not always guaranteed to be a correct indicator of the inlet in which a message was received. Use [proxy_getinlet\(\)](#) to determine the inlet number.

Referenced by [max_jit_obex_proxy_new\(\)](#).

36.5 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 (i.e.

- [Dictionary](#)

Max 5, introduced the [t_dictionary](#) structure/object.

- [Hash Table](#)

A hash table is a data structure that associates some data with a unique key.

- [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](#)

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.

- [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.

- [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.*

- [Dictionary Passing API](#)

The Dictionary Passing API defines a means by which `t_dictionary` instances may be passed between Max objects in a way similar to the way Jitter Matrices are passed between objects.

Typedefs

- [typedef long\(* t_cmpfn\) \(void *, void *\)](#)

Comparison function pointer type.

Enumerations

- [enum e_max_datastore_flags {
OBJ_FLAG_OBJ, OBJ_FLAG_REF, OBJ_FLAG_DATA, OBJ_FLAG_MEMORY,
OBJ_FLAG_SILENT, OBJ_FLAG_INHERITABLE, OBJ_FLAG_ITERATING, OBJ_FLAG_DEBUG }](#)

Flags used in linklist and hashtab objects.

36.5.1 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.

36.5.2 Typedef Documentation

36.5.2.1 [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\(\)](#)

36.5.3 Enumeration Type Documentation

36.5.3.1 [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

OBJ_FLAG_ITERATING used by linklist to signal when is inside iteration

OBJ_FLAG_DEBUG context-dependent flag, used internally for linklist debug code

36.6 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.

Macros

- #define `ATOMARRAY_FLAG_FREECHILDREN`
The atomarray flags.

Functions

- BEGIN_USING_C_LINKAGE `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.
- `t_atom_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_clone (t_atomarray *x)`
Create a new atomarray object which is a full clone 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.

36.6.1 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](#)

36.6.2 Macro Definition Documentation

36.6.2.1 #define ATOMARRAY_FLAG_FREECHILDREN

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.

36.6.3 Function Documentation

36.6.3.1 void atomarray_appendatom (**t_atomarray** **x*, **t_atom** **a*)

Copy a new atom onto the end of the array.

Parameters

<i>x</i>	The atomarray instance.
<i>a</i>	A pointer to the new atom to append to the end of the array.

See also

[atomarray_appendatoms\(\)](#)
[atomarray_setatoms\(\)](#)

36.6.3.2 void atomarray_appendatoms (**t_atomarray** **x*, long *ac*, **t_atom** **av*)

Copy multiple new atoms onto the end of the array.

Parameters

<i>x</i>	The atomarray instance.
<i>ac</i>	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\(\)](#)

36.6.3.3 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.

36.6.3.4 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.

36.6.3.5 void* atomarray_clone (t_atomarray * x)

Create a new atomarray object which is a full clone 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\(\)](#)

36.6.3.6 t_max_err atomarray_copyatoms (t_atomarray * x, long * ac, t_atom ** av)

Retrieve a copy of the atoms in the array.

To retrieve a pointer to the contained atoms use [atomarray_getatoms\(\)](#).

Parameters

<i>x</i>	The atomarray instance.
<i>ac</i>	The address of a long where the number of atoms will be set.
<i>av</i>	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.

```

1 long      ac = 0;
2 t_atom *av = NULL;
3
4 atomarray_copyatoms(anAtomarray, &ac, &av);
5 if(ac && av){
6     // do something with ac and av here...
7     sysmem_freeptr(av);
8 }
```

See also

[atomarray_getatoms\(\)](#)

36.6.3.7 void* atomarray_duplicate (*t_atomarray* * *x*)

Create a new atomarray object which is a copy of another atomarray object.

Parameters

<i>x</i>	The atomarray instance which is to be copied.
----------	---

Returns

A new atomarray which is copied from *x*.

See also

[atomarray_new\(\)](#)

36.6.3.8 void atomarray_flags (*t_atomarray* * *x*, long *flags*)

Set the atomarray flags.

Parameters

<i>x</i>	The atomarray instance.
<i>flags</i>	The new value for the flags.

36.6.3.9 void atomarray_funall (*t_atomarray* * *x*, method *fun*, void * *arg*)

Call the specified function for every item in the atom array.

Parameters

<i>x</i>	The atomarray instance.
<i>fun</i>	The function to call, specified as function pointer cast to a Max method .
<i>arg</i>	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\(\)](#).

```
1 void myFun(t_atom *a, void *myArg)
2 {
3     // do something with a and myArg here
4     // a is the atom in the atom array
5 }
```

See also

[linklist_funall\(\)](#)
[hashtab_funall\(\)](#)

36.6.3.10 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

<i>x</i>	The atomarray instance.
<i>ac</i>	The address of a long where the number of atoms will be set.
<i>av</i>	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\(\)](#)

36.6.3.11 long atomarray_getflags (t_atomarray *x)

Get the atomarray flags.

Parameters

<i>x</i>	The atomarray instance.
----------	-------------------------

Returns

The current value of the atomarray flags.

36.6.3.12 `t_max_err atomarray_getindex (t_atomarray * x, long index, t_atom * av)`

Copy an a specific atom from the array.

Parameters

<i>x</i>	The atomarray instance.
<i>index</i>	The zero-based index into the array from which to retrieve an atom pointer.
<i>av</i>	The address of an atom to contain the copy.

Returns

A Max error code.

Remarks

Example:

```

1 {
2     t_atom a;
3
4     // fetch a copy of the second atom in a previously existing array
5     atomarray_getindex(anAtomarray, 1, &a);
6     // do something with the atom here...
7 }
```

36.6.3.13 t_atom_long atomarray_getsize(t_atomarray * x)

Return the number of atoms in the array.

Parameters

<i>x</i>	The atomarray instance.
----------	-------------------------

Returns

The number of atoms in the array.

36.6.3.14 BEGIN_USING_C_LINKAGE 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

<i>ac</i>	The number of atoms to be initially contained in the atomarray.
<i>av</i>	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.

```
1 object_new(CLASS_NOBOX, gensym("atomarray"), argc, argv);
```

See also

[atomarray_duplicate\(\)](#)

36.6.3.15 `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

<i>x</i>	The atomarray instance.
<i>ac</i>	The number of atoms to be initially contained in the atomarray.
<i>av</i>	A pointer to the first of an array of atoms to initially copy into the atomarray.

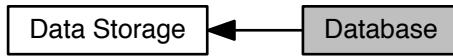
Returns

A Max error code.

36.7 Database

Max's database (i.e.

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_open_ext (t_symbol *dbname, const char *fullpath, t_database **db, long flags)`
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_direct (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)`

- **`t_max_err db_view_create (t_database *db, const char *sql, t_db_view **dbview)`**

Force any open transactions to close.

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.
- **`t_ptr_uint 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, t_ptr_uint *date)`**

A utility to convert from a sql datetime string into seconds.
- **`void db_util_datetostring (const t_ptr_uint date, char *string)`**

A utility to convert from seconds into a sql-ready datetime string.

36.7.1 Detailed Description

Max's database (i.e.

`t_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.

36.7.2 Typedef Documentation

36.7.2.1 `typedef t_object t_database`

A database object.

Use `db_open()` and `db_close()` to create and free database objects.

36.7.2.2 `typedef t_object t_db_result`

A database result object.

This is what the database object returns when a query is executed.

36.7.2.3 `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.

36.7.3 Function Documentation

36.7.3.1 `t_max_err db_close (t_database ** db)`

Close an open database.

Parameters

<code>db</code>	The address of the <code>t_database</code> pointer for your database instance. The pointer will be freed and set NULL upon return.
-----------------	--

Returns

An error code.

36.7.3.2 `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

<code>dbname</code>	The name of the database.
<code>fullpath</code>	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.
<code>db</code>	The address of a <code>t_database</code> 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.

36.7.3.3 `t_max_err db_open_ext (t_symbol * dbname, const char * fullpath, t_database ** db, long flags)`

Create an instance of a database.

Parameters

<i>dbname</i>	The name of the database.
<i>fullpath</i>	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.
<i>db</i>	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.
<i>flags</i>	Any flags to be passed to the database backend while opening the db. At this time, DB_O← PEN_FLAGS_READONLY (0x01) is the only flag available.

Returns

An error code.

36.7.3.4 t_max_err db_query (t_database * db, t_db_result ** dbresult, const char * sql, ...)

Execute a SQL query on the database.

Parameters

<i>db</i>	The t_database pointer for your database instance.
<i>dbresult</i>	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() .
<i>sql</i>	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.

36.7.3.5 t_max_err db_query_direct (t_database * db, t_db_result ** dbresult, const char * sql)

Execute a SQL query on the database.

Parameters

<i>db</i>	The t_database pointer for your database instance.
<i>dbresult</i>	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() .
<i>sql</i>	A C-string containing a valid SQL query.

Returns

An error code.

36.7.3.6 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

<i>db</i>	The t_database pointer for your database instance.
<i>id</i>	The address of a variable to hold the result on return.

Returns

An error code.

36.7.3.7 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

<i>db</i>	The t_database pointer for your database instance.
<i>dbresult</i>	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() .
<i>sql</i>	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.

36.7.3.8 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.

Parameters

<i>db</i>	The t_database pointer for your database instance.
<i>tablename</i>	The name of the table to which the column should be added.
<i>columnname</i>	The name to use for the new column.
<i>columntype</i>	The SQL type for the data that will be stored in the column. For example: "INTEGER" or "VARCHAR"
<i>flags</i>	If you wish to specify any additional information for the column, then pass that here. Otherwise pass NULL.

Returns

An error code.

36.7.3.9 t_max_err db_query_table_new(t_database * db, const char * tablename)

Create a new table in a database.

Parameters

<i>db</i>	The t_database pointer for your database instance.
<i>tablename</i>	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.

36.7.3.10 void db_result_clear ([t_db_result](#) * *result*)

Zero-out a database result.

Parameters

<i>result</i>	The t_db_result pointer for your query results.
---------------	---

36.7.3.11 [t_ptr_uint](#) 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.

The value will be coerced from an expected datetime field into seconds.

Parameters

<i>result</i>	The t_db_result pointer for your query results.
<i>recordindex</i>	The zero-based index number of the record (row) in the result.
<i>fieldindex</i>	The zero-based index number of the field (column) in the result.

Returns

The datetime represented in seconds.

36.7.3.12 char* db_result_fieldname ([t_db_result](#) * *result*, long *fieldindex*)

Return the name of a field specified by its index number.

Parameters

<i>result</i>	The t_db_result pointer for your query results.
<i>fieldindex</i>	The zero-based index number of the field (column) in the result.

Returns

A C-String with the name of the field.

36.7.3.13 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

<i>result</i>	The t_db_result pointer for your query results.
<i>recordindex</i>	The zero-based index number of the record (row) in the result.
<i>fieldindex</i>	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.

36.7.3.14 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

<i>result</i>	The t_db_result pointer for your query results.
<i>recordindex</i>	The zero-based index number of the record (row) in the result.
<i>fieldindex</i>	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.

36.7.3.15 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

<i>result</i>	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\(\)](#).

36.7.3.16 long db_result_numfields (*t_db_result* * *result*)

Return a count of all fields (columns) in the query result.

Parameters

<i>result</i>	The t_db_result pointer for your query results.
---------------	---

Returns

The count of fields in the query result.

36.7.3.17 long db_result_numrecords (*t_db_result* * *result*)

Return a count of all records in the query result.

Parameters

<i>result</i>	The t_db_result pointer for your query results.
---------------	---

Returns

The count of records in the query result.

36.7.3.18 void db_result_reset (t_db_result * result)

Reset the interface for walking a result's record list to the first record.

Parameters

<i>result</i>	The t_db_result pointer for your query results.
---------------	---

36.7.3.19 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

<i>result</i>	The t_db_result pointer for your query results.
<i>recordindex</i>	The zero-based index number of the record (row) in the result.
<i>fieldindex</i>	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.

36.7.3.20 t_max_err db_transaction_end (t_database * db)

Finalize a database transaction.

Parameters

<i>db</i>	The t_database pointer for your database instance.
-----------	--

Returns

An error code.

36.7.3.21 t_max_err db_transaction_flush (t_database * db)

Force any open transactions to close.

Parameters

<i>db</i>	The t_database pointer for your database instance.
-----------	--

Returns

An error code.

36.7.3.22 `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_transaction_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

<code>db</code>	The t_database pointer for your database instance.
-----------------	--

Returns

An error code.

36.7.3.23 `void db_util_datetostring (const t_ptr_uint date, char * string)`

A utility to convert from seconds into a sql-ready datetime string.

Parameters

<code>date</code>	The datetime represented in seconds.
<code>string</code>	The address of a valid C-string whose contents will be set to a SQL-ready string format upon return.

36.7.3.24 `void db_util_stringtodata (const char * string, t_ptr_uint * date)`

A utility to convert from a sql datetime string into seconds.

Parameters

<code>string</code>	A C-string containing a date and time in SQL format.
<code>date</code>	The datetime represented in seconds upon return.

36.7.3.25 `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

<code>db</code>	The t_database pointer for your database instance.
<code>sql</code>	A SQL query that defines the set of results provided by the view.
<code>dbview</code>	The address of a NULL t_db_view pointer which will be set with the new view upon return.

Returns

An error code.

36.7.3.26 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

<i>dbview</i>	The t_db_view pointer for your database view instance.
<i>result</i>	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.

36.7.3.27 t_max_err db_view_remove (t_database * db, t_db_view ** dbview)

Remove a database view created using [db_view_create\(\)](#).

Parameters

<i>db</i>	The t_database pointer for your database instance for which this view was created.
<i>dbview</i>	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.

36.7.3.28 t_max_err db_view_setquery (t_db_view * dbview, char * newquery)

Set the query used by the view.

Parameters

<i>dbview</i>	The t_db_view pointer for your database view instance.
<i>newquery</i>	The SQL string to define a new query for the view, replacing the old query.

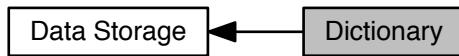
Returns

An error code.

36.8 Dictionary

Max 5, introduced the `t_dictionary` structure/object.

Collaboration diagram for Dictionary:



Data Structures

- struct `t_dictionary_entry`

A dictionary entry.

- struct `t_dictionary`

The dictionary object.

Functions

- `t_dictionary * dictionary_new (void)`

Create a new dictionary object.

- `t_max_err dictionary_appendlong (t_dictionary *d, t_symbol *key, t_atom_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_appendddictionary (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 (C74_CONST t_dictionary *d, t_symbol *key, t_atom_long *value)`

Retrieve a long integer from the dictionary.

- `t_max_err dictionary_getfloat (C74_CONST t_dictionary *d, t_symbol *key, double *value)`

Retrieve a double-precision float from the dictionary.

- `t_max_err dictionary_getsym (C74_CONST t_dictionary *d, t_symbol *key, t_symbol **value)`

Retrieve a `t_symbol` from the dictionary.*

- `t_max_err dictionary_getatom (C74_CONST t_dictionary *d, t_symbol *key, t_atom *value)`

- Copy a `t_atom` from the dictionary.*
- `t_max_err dictionary_getstring (C74_CONST t_dictionary *d, t_symbol *key, const char **value)`
Retrieve a C-string pointer from the dictionary.
 - `t_max_err dictionary_getatoms (C74_CONST 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_getatoms_ext (const t_dictionary *d, t_symbol *key, long stringstosymbols, long *argc, t_atom **argv)`
Retrieve the address of a `t_atom` array in the dictionary.
 - `t_max_err dictionary_copyatoms (C74_CONST 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 (C74_CONST t_dictionary *d, t_symbol *key, t_object **value)`
Retrieve a `t_atomarray` pointer from the dictionary.
 - `t_max_err dictionary_getdictionary (C74_CONST t_dictionary *d, t_symbol *key, t_object **value)`
Retrieve a `t_dictionary` pointer from the dictionary.
 - `t_max_err dictionary_get_ex (t_dictionary *d, t_symbol *key, long *ac, t_atom **av, char *errstr)`
Retrieve the address of a `t_atom` array of in the dictionary within nested dictionaries.
 - `t_max_err dictionary_getobject (C74_CONST t_dictionary *d, t_symbol *key, t_object **value)`
Retrieve a `t_object` pointer from the dictionary.
 - `long dictionary_entryisstring (C74_CONST 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 (C74_CONST 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 (C74_CONST 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 (C74_CONST t_dictionary *d, t_symbol *key)`
Test a key to set if it exists in the dictionary.
 - `t_atom_long dictionary_getentrycount (C74_CONST t_dictionary *d)`
Return the number of keys in a dictionary.
 - `t_max_err dictionary_getkeys (C74_CONST 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.*
 - `void dictionary_entry_getvalues (t_dictionary_entry *x, long *argc, t_atom **argv)`
Given a `t_dictionary_entry`, return the values 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 (const t_dictionary *d, t_symbol *key, t_atom_long *value, t_atom_long def)`
Retrieve a long integer from the dictionary.
 - `t_max_err dictionary_getdeffloat (const t_dictionary *d, t_symbol *key, double *value, double def)`

- `t_max_err dictionary_getdefsym (const t_dictionary *d, t_symbol *key, t_symbol **value, t_symbol *def)`

Retrieve a double-precision float from the dictionary.
- `t_max_err dictionary_getdefatom (const t_dictionary *d, t_symbol *key, t_atom *value, t_atom *def)`

Retrieve a `t_symbol` from the dictionary.*
- `t_max_err dictionary_getdefstring (const 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.
- `t_max_err dictionary_read_yaml (const char *filename, const short path, t_dictionary **d)`

Read the specified YAML file and return a `t_dictionary` object.
- `t_max_err dictionary_write_yaml (const t_dictionary *d, const char *filename, const short path)`

Serialize the specified `t_dictionary` object to a YAML file.
- `void postdictionary (t_object *d)`

Print the contents of a dictionary to the Max window.

36.8.1 Detailed Description

Max 5, introduced the `t_dictionary` structure/object.

This is used for object prototypes, object serialization, object constructors, and many 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).

36.8.2 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.

36.8.2.1 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);
}
```

36.8.2.2 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.

36.8.2.3 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

36.8.3 Reading and Writing Dictionaries as JSON

36.8.3.1 Creating a Dictionary from JSON

The easiest way to get read a `t_dictionary` from a JSON file on disk is to use the `dictionary_read()` function. In some cases you may wish have more control, such as to generate JSON but not write it to disk. For that purpose you can create a `jsonreader` object as demonstrated below.

```
t_dictionary      *d = NULL;
t_max_err        err;
t_atom            result[1];
t_object          *jsonreader = (t_object*)object_new(_sym_nobox, _sym_jsonreader);

// assume we have an argument called 'jsontext' which is a const char* with the JSON
// from which we wish to create a t_dictionary instance

err = (t_max_err)object_method(jsonreader, _sym_parse, jsontext, result);
if (!err) {
    t_object *ro = (t_object*)atom_getobj(result);

    if (ro) {
        if (object_classname_compare(ro, _sym_dictionary))
            d = (t_dictionary*)ro;
        else
            object_free(ro);
    }
}
object_free(jsonreader);

// we now have a t_dictionary in d that can be used as we see fit
```

36.8.3.2 Creating JSON from a Dictionary

The easiest way to get write a `t_dictionary` to disk as JSON is to use the `dictionary_write()` function. In some cases you may wish have more control, such as to generate JSON but not write it to disk. For that purpose you can create a `jsonwriter` object as demonstrated below.

```
t_object    *jsonwriter = (t_object*)object_new(_sym_nobox, _sym_jsonwriter);
t_handle    json;
const char   *str;

object_method(jsonwriter, _sym_writedictionary, d);
object_method(jsonwriter, _sym_getoutput, &json);

str = *json;

// now str contains our JSON serialization of the t_dictionary d

object_free(jsonwriter);
```

36.8.4 Function Documentation

36.8.4.1 `t_max_err dictionary_appendatom(t_dictionary *d, t_symbol *key, t_atom *value)`

Add a `t_atom*` value to the dictionary.

Parameters

<code>d</code>	The dictionary instance.
<code>key</code>	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.
<code>value</code>	The new value to append to the dictionary.

Returns

A Max error code.

36.8.4.2 `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

<code>d</code>	The dictionary instance.
<code>key</code>	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.
<code>value</code>	The new value to append to the dictionary.

Returns

A Max error code.

36.8.4.3 `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

<i>d</i>	The dictionary instance.
<i>key</i>	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.
<i>argc</i>	The number of atoms to append to the dictionary.
<i>argv</i>	The address of the first atom in the array to append to the dictionary.

Returns

A Max error code.

36.8.4.4 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

<i>d</i>	The dictionary instance.
<i>key</i>	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.
<i>value</i>	The new value to append to the dictionary.

Returns

A Max error code.

36.8.4.5 t_max_err dictionary_appendfloat (t_dictionary * *d*, t_symbol * *key*, double *value*)

Add a double-precision float value to the dictionary.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	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.
<i>value</i>	The new value to append to the dictionary.

Returns

A Max error code.

36.8.4.6 t_max_err dictionary_appendlong (t_dictionary * *d*, t_symbol * *key*, t_atom_long *value*)

Add a long integer value to the dictionary.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	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.
<i>value</i>	The new value to append to the dictionary.

Returns

A Max error code.

36.8.4.7 `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

<i>d</i>	The dictionary instance.
<i>key</i>	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.
<i>value</i>	The new value to append to the dictionary.

Returns

A Max error code.

36.8.4.8 `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

<i>d</i>	The dictionary instance.
<i>key</i>	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.
<i>value</i>	The new value to append to the dictionary.

Returns

A Max error code.

36.8.4.9 `t_max_err dictionary_appendsym(t_dictionary * d, t_symbol * key, t_symbol * value)`

Add a `t_symbol*` value to the dictionary.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	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.
<i>value</i>	The new value to append to the dictionary.

Returns

A Max error code.

36.8.4.10 t_max_err dictionary_chuckentry (t_dictionary * *d*, t_symbol * *key*)

Remove a value from the dictionary without freeing it.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to delete.

Returns

A max error code.

See also

[dictionary_deleteentry\(\)](#)

36.8.4.11 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

<i>d</i>	The dictionary instance.
----------	--------------------------

Returns

A max error code.

See also

[dictionary_getkeys\(\)](#)
[dictionary_chuckentry\(\)](#)
[dictionary_deleteentry\(\)](#)

36.8.4.12 t_max_err dictionary_copyatoms (C74_CONST 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

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>argc</i>	The address of a variable to hold the number of atoms in the array.
<i>argv</i>	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\(\)](#)

36.8.4.13 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

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>argc</i>	The address of a variable to hold the number of atoms in the array.
<i>argv</i>	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() .
<i>def</i>	The default values specified as an instance of the t_atomarray object.

Returns

A Max error code.

See also

[dictionary_getdefatoms\(\)](#)
[dictionary_copyatoms\(\)](#)

36.8.4.14 t_max_err dictionary_copyentries (t_dictionary * *src*, t_dictionary * *dst*, t_symbol ** *keys*)

Copy specified entries from one dictionary to another.

Parameters

<i>src</i>	The source dictionary from which to copy entries.
<i>dst</i>	The destination dictionary to which the entries will be copied.
<i>keys</i>	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\(\)](#)

36.8.4.15 `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

<code>d</code>	A dictionary instance. This will be the destination for any values that are copied.
<code>copyfrom</code>	A dictionary instance from which we will copy any values with unique keys.

Returns

A Max error code.

See also

[dictionary_copyentries\(\)](#)

36.8.4.16 `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

<code>d</code>	The dictionary instance.
<code>key</code>	The key associated with the value to delete.

Returns

A max error code.

See also

[dictionary_chuckentry\(\)](#)
[dictionary_clear\(\)](#)

36.8.4.17 `t_max_err dictionary_dump (t_dictionary * d, long recurse, long console)`

Print the contents of a dictionary to the Max window.

Parameters

<code>d</code>	The dictionary instance.
<code>recurse</code>	If non-zero, the dictionary will be recursively unravelled to the Max window. Otherwise it will only print the top level.
<code>console</code>	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.

36.8.4.18 `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\(\)](#)

36.8.4.19 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 <i>t_atom</i> to which the value will be copied.

See also

[dictionary_entry_getkey\(\)](#)
[dictionary_funall\(\)](#)

36.8.4.20 void dictionary_entry_getvalues (*t_dictionary_entry* * *x*, *long* * *argc*, *t_atom* ** *argv*)

Given a [*t_dictionary_entry**](#), return the values associated with that entry.

Parameters

x	The dictionary entry.
argc	The length of the returned <i>t_atom</i> vector.
argv	The address of a <i>t_atom</i> vector to which the values will be copied.

See also

[dictionary_entry_getkey\(\)](#)
[dictionary_funall\(\)](#)

36.8.4.21 long dictionary_entryisatomarray (*C74_CONST 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.

36.8.4.22 long dictionary_entryisdictionary (*C74_CONST t_dictionary* * *d*, *t_symbol* * *key*)

Test a key to set if the data stored with that key contains a [*t_dictionary*](#) object.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to test.

Returns

Returns true if the key contains a [t_dictionary](#), otherwise returns false.

36.8.4.23 long dictionary_entryisstring (C74_CONST t_dictionary * *d*, t_symbol * *key*)

Test a key to set if the data stored with that key contains a [t_string](#) object.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to test.

Returns

Returns true if the key contains a [t_string](#), otherwise returns false.

36.8.4.24 void dictionary_freekeys (t_dictionary * *d*, long *numkeys*, t_symbol ** *keys*)

Free memory allocated by the [dictionary_getkeys\(\)](#) method.

Parameters

<i>d</i>	The dictionary instance.
<i>numkeys</i>	The address of a long where the number of keys retrieved will be set.
<i>keys</i>	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\(\)](#)

36.8.4.25 void dictionary_funall (t_dictionary * *d*, method *fun*, void * *arg*)

Call the specified function for every entry in the dictionary.

Parameters

<i>d</i>	The dictionary instance.
<i>fun</i>	The function to call, specified as function pointer cast to a Max method .
<i>arg</i>	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\(\)](#).

```

1 void my_function(t_dictionary_entry *entry, void* my_arg)
2 {
3     t_symbol    *key;
4     t_atom      value;
5
6     key = dictionary_entry_getkey(entry);
7     dictionary_entry_getvalue(entry, &value);
8
9     // do something with key, value, and my_arg...
10 }

```

See also

[dictionary_entry_getkey\(\)](#)
[dictionary_entry_getvalue\(\)](#)

36.8.4.26 t_max_err dictionary_get_ex (t_dictionary * d, t_symbol * key, long * ac, t_atom ** av, char * errstr)

Retrieve the address of a [t_atom](#) array of in the dictionary within nested dictionaries.

The address can index into nested dictionaries using the '::' operator. For example, the key "field::subfield" will look for the value at key "field" and then look for the value "subfield" in the value found at "field".

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>ac</i>	The number of return values
<i>av</i>	The return values
<i>errstr</i>	An error message if an error code was returned. Optional, pass NULL if you don't want it.

Returns

A Max error code.

36.8.4.27 t_max_err dictionary_getatom (C74_CONST t_dictionary * d, t_symbol * key, t_atom * value)

Copy a [t_atom](#) from the dictionary.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>value</i>	The address of variable to hold the value associated with the key.

Returns

A Max error code.

36.8.4.28 t_max_err dictionary_getatomarray (C74_CONST t_dictionary * d, t_symbol * key, t_object ** value)

Retrieve a [t_atomarray](#) pointer from the dictionary.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>value</i>	The address of variable to hold the value associated with the key.

Returns

A Max error code.

36.8.4.29 t_max_err dictionary_getatoms(C74_CONST 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

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>argc</i>	The address of a variable to hold the number of atoms in the array.
<i>argv</i>	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\(\)](#)
[dictionary_getatoms_ext\(\)](#)

36.8.4.30 t_max_err dictionary_getatoms_ext(const t_dictionary * *d*, t_symbol * *key*, long *stringstosymbols*, long * *argc*, t_atom ** *argv*)

Retrieve the address of a [t_atom](#) array in the dictionary.

The retrieved pointer references the [t_atoms](#) in the dictionary. Optionally convert strings to symbols. To fetch a copy of the [t_atoms](#) from the dictionary, use [dictionary_copyatoms\(\)](#).

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>stringstosymbols</i>	The flag to convert strings to symbols (true/false).
<i>argc</i>	The address of a variable to hold the number of atoms in the array.
<i>argv</i>	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\(\)](#)
[dictionary_getatoms\(\)](#)

36.8.4.31 t_max_err dictionary_getdefatom (const 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

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>value</i>	The address of variable to hold the value associated with the key.
<i>def</i>	The default value to return in the absence of the key existing in the dictionary.

Returns

A Max error code.

See also

[dictionary_getatom\(\)](#)

36.8.4.32 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

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>argc</i>	The address of a variable to hold the number of atoms in the array.
<i>argv</i>	The address of a variable to hold a pointer to the first atom in the array.
<i>def</i>	The default values specified as an instance of the t_atomarray object.

Returns

A Max error code.

See also

[dictionary_getatoms\(\)](#)
[dictionary_copydefatoms\(\)](#)

36.8.4.33 t_max_err dictionary_getdeffloat (const 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

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>value</i>	The address of variable to hold the value associated with the key.
<i>def</i>	The default value to return in the absence of the key existing in the dictionary.

Returns

A Max error code.

See also

[dictionary_getfloat\(\)](#)

36.8.4.34 `t_max_err dictionary_getdeflong (const t_dictionary * d, t_symbol * key, t_atom_long * value, t_atom_long def)`

Retrieve a long integer from the dictionary.

If the named key doesn't exist, then return a specified default value.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>value</i>	The address of variable to hold the value associated with the key.
<i>def</i>	The default value to return in the absence of the key existing in the dictionary.

Returns

A Max error code.

See also

[dictionary_getlong\(\)](#)

36.8.4.35 `t_max_err dictionary_getdefstring (const 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

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>value</i>	The address of variable to hold the value associated with the key.
<i>def</i>	The default value to return in the absence of the key existing in the dictionary.

Returns

A Max error code.

See also

[dictionary_getstring\(\)](#)

36.8.4.36 `t_max_err dictionary_getdefsym (const 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

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>value</i>	The address of variable to hold the value associated with the key.
<i>def</i>	The default value to return in the absence of the key existing in the dictionary.

Returns

A Max error code.

See also

[dictionary_getsym\(\)](#)

36.8.4.37 t_max_err dictionary_getdictionary (C74_CONST t_dictionary * *d*, t_symbol * *key*, t_object ** *value*)

Retrieve a [t_dictionary](#) pointer from the dictionary.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>value</i>	The address of variable to hold the value associated with the key.

Returns

A Max error code.

36.8.4.38 t_atom_long dictionary_getentrycount (C74_CONST t_dictionary * *d*)

Return the number of keys in a dictionary.

Parameters

<i>d</i>	The dictionary instance.
----------	--------------------------

Returns

The number of keys in the dictionary.

36.8.4.39 t_max_err dictionary_getfloat (C74_CONST t_dictionary * *d*, t_symbol * *key*, double * *value*)

Retrieve a double-precision float from the dictionary.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>value</i>	The address of variable to hold the value associated with the key.

Returns

A Max error code.

36.8.4.40 t_max_err dictionary_getkeys (C74_CONST 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

<i>d</i>	The dictionary instance.
<i>numkeys</i>	The address of a long where the number of keys retrieved will be set.
<i>keys</i>	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.

```

1 t_symbol    **keys = NULL;
2 long        numkeys = 0;
3 long        i;
4 t_object    *anItem;
5
6 dictionary_getkeys(d, &numkeys, &keys);
7 for(i=0; i<numkeys; i++){
8     // do something with the keys...
9 }
10 if(keys)
11     dictionary_freekeys(d, numkeys, keys);

```

See also

[dictionary_freekeys\(\)](#)

36.8.4.41 t_max_err dictionary_getlong (C74_CONST t_dictionary * d, t_symbol * key, t_atom_long * value)

Retrieve a long integer from the dictionary.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>value</i>	The address of variable to hold the value associated with the key.

Returns

A Max error code.

36.8.4.42 t_max_err dictionary_getobject (C74_CONST t_dictionary * d, t_symbol * key, t_object ** value)

Retrieve a [t_object](#) pointer from the dictionary.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>value</i>	The address of variable to hold the value associated with the key.

Returns

A Max error code.

36.8.4.43 t_max_err dictionary_getstring (C74_CONST 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

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>value</i>	The address of variable to hold the value associated with the key.

Returns

A Max error code.

36.8.4.44 t_max_err dictionary_getsym (C74_CONST t_dictionary * *d*, t_symbol * *key*, t_symbol ** *value*)

Retrieve a [t_symbol*](#) from the dictionary.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to lookup.
<i>value</i>	The address of variable to hold the value associated with the key.

Returns

A Max error code.

36.8.4.45 long dictionary_hasentry (C74_CONST t_dictionary * *d*, t_symbol * *key*)

Test a key to set if it exists in the dictionary.

Parameters

<i>d</i>	The dictionary instance.
<i>key</i>	The key associated with the value to test.

Returns

Returns true if the key exists, otherwise returns false.

36.8.4.46 `t_dictionary* dictionary_new(void)`

Create a new dictionary object.

You can free the dictionary 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\(\)](#)

36.8.4.47 `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

<i>filename</i>	The name of the file.
<i>path</i>	The path of the file.
<i>d</i>	The address of a t_dictionary pointer that will be set to the newly created dictionary.

Returns

A Max error code

36.8.4.48 `t_max_err dictionary_read_yaml(const char * filename, const short path, t_dictionary ** d)`

Read the specified YAML 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

<i>filename</i>	The name of the file.
<i>path</i>	The path of the file.
<i>d</i>	The address of a t_dictionary pointer that will be set to the newly created dictionary.

Returns

A Max error code

36.8.4.49 `t_dictionary* dictionary_sprintf(C74_CONST char * fmt, ...)`

Create a new dictionary populated with values using a combination of attribute and sprintf syntax.

Parameters

<i>fmt</i>	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,

```
1 "@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.

```
1 t_dictionary *d;
2 char text[4];
3
4 strncpy_zero(text, "foo", 4);
5
6 d = dictionary_sprintf("@maxclass comment @varname _name \
7 @text \"%s\" @patching_rect %.2f %.2f %.2f %.2f \
8 @fontsize %f @textcolor %f %f %f 1.0 \
9 @fontname %s @bgcolor 0.001 0.001 0.001 0.", \
10 text, 20.0, 20.0, 200.0, 24.0,
11 18, 0.9, 0.9, 0.9, "Arial");
12
13 // do something with the dictionary here.
14
15 object_free(d);
```

See also

[newobject_sprintf\(\)](#)
[newobject_fromdictionary\(\)](#)
[atom_setparse\(\)](#)

36.8.4.50 t_max_err dictionary_write(t_dictionary * d, char * filename, short path)

Serialize the specified [t_dictionary](#) object to a JSON file.

Parameters

<i>d</i>	The dictionary to serialize into JSON format and write to disk.
<i>filename</i>	The name of the file to write.
<i>path</i>	The path to which the file should be written.

Returns

A Max error code.

36.8.4.51 t_max_err dictionary_write_yaml(const t_dictionary * d, const char * filename, const short path)

Serialize the specified [t_dictionary](#) object to a YAML file.

Parameters

<i>d</i>	The dictionary to serialize into YAML format and write to disk.
<i>filename</i>	The name of the file to write.
<i>path</i>	The path to which the file should be written.

Returns

A Max error code.

36.8.4.52 void postdictionary (t_object * d)

Print the contents of a dictionary to the Max window.

Parameters

<i>d</i>	A pointer to a dictionary object.
----------	-----------------------------------

36.9 Hash Table

A hash table is a data structure that associates some data with a unique key.

Collaboration diagram for Hash Table:



Data Structures

- struct `t_hashtab_entry`
A hashtab entry.
- struct `t_hashtab`
The hashtab object.

Macros

- `#define HASH_DEFSLOTS`
Default number of slots in the hash table.

Functions

- `BEGIN_USING_C_LINKAGE 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_storelong (t_hashtab *x, t_symbol *key, t_atom_long val)`
Store a `t_atom_long` value in a hashtab with an associated key.
- `t_max_err hashtab_storesym (t_hashtab *x, t_symbol *key, t_symbol *val)`
Store a `t_symbol` value 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_lookuplong (t_hashtab *x, t_symbol *key, t_atom_long *val)`
Return a `t_atom_long` value stored in a hashtab with the specified key.
- `t_max_err hashtab_lookupsym (t_hashtab *x, t_symbol *key, t_symbol **val)`
Return a `t_symbol` value 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 hashtable_clear (t_hashtable *x)`
Delete all items stored in a hashtable.
- `t_max_err hashtable_chuckkey (t_hashtable *x, t_symbol *key)`
Remove an item from a hashtable associated with a given key.
- `t_max_err hashtable_chuck (t_hashtable *x)`
Free a hashtable, but don't free the items it contains.
- `t_max_err hashtable_findfirst (t_hashtable *x, void **o, long cmpfn(void *, void *), void *cmpdata)`
Search the hash table for the first item meeting defined criteria.
- `t_max_err hashtable_methodall (t_hashtable *x, t_symbol *s,...)`
Call the named message on every object in the hashtable.
- `t_max_err hashtable_funall (t_hashtable *x, method fun, void *arg)`
Call the specified function for every item in the hashtable.
- `t_atom_long hashtable_getsize (t_hashtable *x)`
Return the number of items stored in a hashtable.
- `void hashtable_print (t_hashtable *x)`
Post a hashtable's statistics to the Max window.
- `void hashtable_READONLY (t_hashtable *x, long readonly)`
Set the hashtable's readonly bit.
- `void hashtable_flags (t_hashtable *x, long flags)`
Set the hashtable's datastore flags.
- `t_atom_long hashtable_getflags (t_hashtable *x)`
Get the hashtable's datastore flags.
- `t_max_err hashtable_keyflags (t_hashtable *x, t_symbol *key, long flags)`
Change the flags for an item stored in the hashtable with a given key.
- `t_atom_long hashtable_getkeyflags (t_hashtable *x, t_symbol *key)`
Retrieve the flags for an item stored in the hashtable with a given key.
- `t_max_err hashtable_getkeys (t_hashtable *x, long *kc, t_symbol ***kv)`
Retrieve all of the keys stored in a hashtable.

36.9.1 Detailed Description

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_hashtable` 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_hashtable`, you use `hashtable_new()`. To add items, use `hashtable_store()`. To find items that have been added, use `hashtable_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 `hashtable_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 hashtable, shows how to add some data (in this case, just a number), look it up, and delete the hashtable.

```
t_hashtable *tab = (t_hashtable *)hashtable_new(0);
long result, value;

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

result = hashtable_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 internally using both a [t_hashtab](#) and a [t_linklist](#) in parallel. The [t_hashtab](#) provides fast access, and the [t_linklist](#) provides sorting.

See also

http://en.wikipedia.org/wiki/Hash_table
[Linked List](#)

36.9.2 Macro Definition Documentation

36.9.2.1 #define HASH_DEFSLOTS

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.

36.9.3 Function Documentation

36.9.3.1 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](#)

36.9.3.2 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

<i>x</i>	The hashtab instance.
<i>key</i>	The key of the item to delete.

Returns

A Max error code.

See also

[hashtab_delete\(\)](#)

36.9.3.3 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\(\)](#)

36.9.3.4 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

<i>x</i>	The hashtab instance.
<i>key</i>	The key of the item to delete.

Returns

A Max error code.

See also

[hashtab_chuckkey\(\)](#)
[hashtab_clear\(\)](#)
[hashtab_flags\(\)](#)

36.9.3.5 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

<i>x</i>	The hashtab instance.
<i>o</i>	The address to pointer that will be set with the matching item.
<i>cmpfn</i>	The function used to determine a match in the list.
<i>cmpdata</i>	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](#)

36.9.3.6 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

<i>x</i>	The hashtab instance.
<i>flags</i>	A valid value from the e_max_datastore_flags . The default is OBJ_FLAG_OBJ .

36.9.3.7 t_max_err hashtab_funall (t_hashtab *x, method fun, void *arg)

Call the specified function for every item in the hashtab.

Parameters

<i>x</i>	The hashtab instance.
<i>fun</i>	The function to call, specified as function pointer cast to a Max method .
<i>arg</i>	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\(\)](#).

```
1 void myFun(t_hashtab_entry *e, void *myArg)
2 {
3     if (e->key && e->value) {
4         // do something with e->key, e->value, and myArg here as appropriate
5     }
6 }
```

Referenced by [jit_class_addinterface\(\)](#).

36.9.3.8 t_atom_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](#).

36.9.3.9 t_atom_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\(\)](#)

36.9.3.10 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.

```

1 t_symbol    **keys = NULL;
2 long        numKeys = 0;
3 long        i;
4 t_object    *anItem;
5
6 hashtab_getkeys(aHashtab, &numKeys, &keys);
7 for(i=0; i<numKeys; i++){
8     hashtab_lookup(aHashtab, keys[i], &anItem);
9     // Do something with anItem here...
10 }
11 if(keys)
12     sysmem_freeptr(keys);

```

36.9.3.11 t_atom_long hashtab_getsize (t_hashtab * x)

Return the number of items stored in a hashtab.

Parameters

<i>x</i>	The hashtab instance.
----------	-----------------------

Returns

The number of items in the hash table.

36.9.3.12 `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

<i>x</i>	The hashtab instance.
<i>key</i>	The key in the hashtab whose flags will be changed.
<i>flags</i>	One of the values listed in e_max_datastore_flags .

Returns

A Max error code.

See also

[hashtab_store_flags\(\)](#)

36.9.3.13 `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

<i>x</i>	The hashtab instance.
<i>key</i>	The key in the hashtab to fetch.
<i>val</i>	The address of a pointer to which the fetched value will be assigned.

Returns

A Max error code.

See also

[hashtab_store\(\)](#), [hashtab_lookuplong\(\)](#), [hashtab_lookupsym\(\)](#)

36.9.3.14 `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

<i>x</i>	The hashtab instance.
<i>key</i>	The key in the hashtab to fetch.

<i>val</i>	The address of a pointer to which the fetched value will be assigned.
<i>flags</i>	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\(\)](#)

36.9.3.15 t_max_err hashtab_lookuplong (t_hashtab *x, t_symbol *key, t_atom_long *val)

Return a t_atom_long value stored in a hashtab with the specified key.

Parameters

<i>x</i>	The hashtab instance.
<i>key</i>	The key in the hashtab to fetch.
<i>val</i>	A pointer to a t_atom_long to which the fetched value will be assigned.

Returns

A Max error code.

See also

[hashtab_storelong\(\)](#), [hashtab_lookup\(\)](#), [hashtab_lookupsym\(\)](#)

36.9.3.16 t_max_err hashtab_lookupsym (t_hashtab *x, t_symbol *key, t_symbol **val)

Return a t_symbol value stored in a hashtab with the specified key.

Parameters

<i>x</i>	The hashtab instance.
<i>key</i>	The key in the hashtab to fetch.
<i>val</i>	A pointer to the address of a t_symbol to which the fetched value will be assigned.

Returns

A Max error code.

See also

[hashtab_storesym\(\)](#), [hashtab_lookup\(\)](#), [hashtab_lookuplong\(\)](#)

36.9.3.17 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

<i>x</i>	The hashtab instance.
<i>s</i>	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.

36.9.3.18 BEGIN_USING_C_LINKAGE 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

<i>slotcount</i>	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\(\)](#)

Referenced by [jit_class_addinterface\(\)](#).

36.9.3.19 void hashtab_print (t_hashtab * x)

Post a hashtab's statistics to the Max window.

Parameters

<i>x</i>	The hashtab instance.
----------	-----------------------

36.9.3.20 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

<i>x</i>	The hashtab instance.
<i>readonly</i>	A 1 or 0 for setting the readonly bit.

36.9.3.21 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

<i>x</i>	The hashtab instance.
<i>key</i>	The key in the hashtab with which to associate the value.
<i>val</i>	The value to store.

Returns

A Max error code.

See also

[hashtab_lookup\(\)](#), [hashtab_storesafe\(\)](#), [hashtab_storelong\(\)](#), [hashtab_storesym\(\)](#)

Referenced by [jit_class_addinterface\(\)](#).

36.9.3.22 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_safe\(\)](#) 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

<i>x</i>	The hashtab instance.
<i>key</i>	The key in the hashtab with which to associate the value.
<i>val</i>	The value to store.

Returns

A Max error code.

See also

[hashtab_store\(\)](#)

36.9.3.23 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

<i>x</i>	The hashtab instance.
<i>key</i>	The key in the hashtab with which to associate the value.
<i>val</i>	The value to store.
<i>flags</i>	One of the values listed in e_max_datastore_flags .

Returns

A Max error code.

See also

[hashtab_store\(\)](#)

36.9.3.24 t_max_err hashtab_storelong (t_hashtab *x, t_symbol *key, t_atom_long val)

Store a *t_atom_long* value in a hashtab with an associated key.

Parameters

<i>x</i>	The hashtab instance.
<i>key</i>	The key in the hashtab with which to associate the value.
<i>val</i>	The <i>t_atom_long</i> value to store.

Returns

A Max error code.

See also

[hashtab_lookuplong\(\)](#), [hashtab_store\(\)](#), [hashtab_storesafe\(\)](#), [hashtab_storesym\(\)](#)

36.9.3.25 t_max_err hashtab_storesym (t_hashtab *x, t_symbol *key, t_symbol *val)

Store a *t_symbol* value in a hashtab with an associated key.

Parameters

<i>x</i>	The hashtab instance.
<i>key</i>	The key in the hashtab with which to associate the value.
<i>val</i>	The <i>t_symbol</i> pointer to store.

Returns

A Max error code.

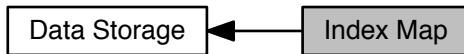
See also

[hashtab_lookupsym\(\)](#), [hashtab_store\(\)](#), [hashtab_storesafe\(\)](#), [hashtab_storelong\(\)](#)

36.10 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.

36.10.1 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 hashtab 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 hashtab are updated with new indices.

36.10.2 Function Documentation

36.10.2.1 void indexmap_append (*t_indexmap* * *x*, *void* * *data*)

Add an item to an indexmap.

Parameters

<i>x</i>	The indexmap instance.
<i>data</i>	The item to add.

36.10.2.2 void indexmap_clear (*t_indexmap* * *x*)

Delete all items in an indexmap.

Parameters

<i>x</i>	The indexmap instance.
----------	------------------------

36.10.2.3 void* indexmap_datafromindex (*t_indexmap* * *x*, long *index*)

Get an item from an indexmap by index.

Parameters

<i>x</i>	The indexmap instance.
<i>index</i>	The index from which to fetch a stored item.

Returns

The item stored at the specified index.

36.10.2.4 t_max_err indexmap_delete (*t_indexmap* * *x*, *void* * *data*)

Delete a specified item from an indexmap.

Parameters

<i>x</i>	The indexmap instance.
<i>data</i>	The item pointer to remove from the indexmap.

Returns

A Max error code.

36.10.2.5 `t_max_err indexmap_delete_index(t_indexmap *x, long index)`

Delete an item from the indexmap by index.

Parameters

<i>x</i>	The indexmap instance.
<i>index</i>	The index of the item to remove from the indexmap.

Returns

A Max error code.

36.10.2.6 *t_max_err indexmap_delete_index_multi (t_indexmap *x, long count, long *indices)*

Delete multiple items from an indexmap by index.

Parameters

<i>x</i>	The indexmap instance.
<i>count</i>	The number of items to remove from the indexmap.
<i>indices</i>	The address of the first of an array of index numbers to remove the indexmap.

Returns

A Max error code.

36.10.2.7 *t_max_err indexmap_delete_multi (t_indexmap *x, long count, void **pdata)*

Delete multiple specified items from an indexmap.

Parameters

<i>x</i>	The indexmap instance.
<i>count</i>	The number of items to remove from the indexmap.
<i>pdata</i>	The address of the first of an array of item pointers to remove from the indexmap.

Returns

A Max error code.

36.10.2.8 *long indexmap_getsize (t_indexmap *x)*

Return the number of items in an indexmap.

Parameters

<i>x</i>	The indexmap instance.
----------	------------------------

Returns

The number of items in the indexmap.

36.10.2.9 *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

<i>x</i>	The indexmap instance.
<i>data</i>	The item whose index you wish to look up.
<i>index</i>	The address of a variable to hold the retrieved index.

Returns

A Max error code.

36.10.2.10 `t_max_err indexmap_move(t_indexmap *x, void *data, long newindex)`

Move an item to a different position in an indexmap.

Parameters

<i>x</i>	The indexmap instance.
<i>data</i>	The item in the indexmap to move.
<i>newindex</i>	The new index to which to move the item.

Returns

A Max error code.

36.10.2.11 `t_indexmap* indexmap_new(void)`

Create a new indexmap object.

Returns

Pointer to the new indexmap object.

36.10.2.12 `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

<i>x</i>	The indexmap instance.
<i>fn</i>	The function used to sort the list.

See also

[linklist_sort\(\)](#)

36.11 Linked List

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.

Collaboration diagram for Linked List:



Data Structures

- struct `t_llitem`
A linklist element.
- struct `t_linklist`
The linklist object.

Functions

- `BEGIN_USING_C_LINKAGE 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.
- `t_atom_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.
- `t_atom_long linklist_objptr2index (t_linklist *x, void *p)`
Return an item's index, given the item itself.
- `t_atom_long linklist_append (t_linklist *x, void *o)`
Add an item to the end of the list.
- `t_atom_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.
- `t_atom_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.
- long `linklist_chuckobject` (`t_linklist` **x*, void **o*)

Remove the specified item from the list.
- long `linklist_deleteobject` (`t_linklist` **x*, void **o*)

Delete the specified item from the list.
- void `linklist_clear` (`t_linklist` **x*)

Remove and free all items in the list.
- `t_atom_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.
- `t_atom_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*, `t_atom_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.
- `t_atom_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.
- `t_atom_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.

36.11.1 Detailed Description

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")); // find 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);
}
```

See also

http://en.wikipedia.org/wiki/Linked_list

36.11.2 Function Documentation

36.11.2.1 `t_atom_long linklist_append(t_linklist *x, void *o)`

Add an item to the end of the list.

Parameters

<code>x</code>	The linklist instance.
<code>o</code>	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.

Referenced by `jit_linklist_append()`.

36.11.2.2 `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

<code>x</code>	The linklist object to be freed.
----------------	----------------------------------

See also

[object_free](#)

Referenced by `jit_linklist_chuck()`, and `jit_linklist_findcount()`.

36.11.2.3 `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

<code>x</code>	The linklist instance.
<code>index</code>	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](#)

Referenced by `jit_linklist_chuckindex()`.

36.11.2.4 long 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

<i>x</i>	The linklist instance.
<i>o</i>	The pointer to the item to remove.

See also

[linklist_deleteindex](#)
[linklist_chuckindex](#)
[linklist_deleteobject](#)

36.11.2.5 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

<i>x</i>	The linklist instance.
----------	------------------------

Referenced by `jit_linklist_clear()`, and `jit_linklist_free()`.

36.11.2.6 t_atom_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

<i>x</i>	The linklist instance.
<i>index</i>	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](#)

Referenced by `jit_linklist_deleteindex()`.

36.11.2.7 long linklist_deleteobject (t_linklist * x, void * o)

Delete the specified item from the list.

The object is removed from the list and deleted. The deletion is done with respect to any flags passed to linklist_← flags.

Parameters

x	The linklist instance.
o	The pointer to the item to delete.

See also

[linklist_deleteindex](#)
[linklist_chuckindex](#)
[linklist_chuckobject](#)

36.11.2.8 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.

```
1 t_linklist *results = NULL;
2
3 linklist_findall(myLinkList, &results, myCmpFunction, (void *)someCriteria);
4 // do something here with the 'results' linklist
5 // then free the results linklist
6 linklist_chuck(results);
```

See also

[linklist_match](#)
[t_cmpfn](#)
[linklist_findfirst](#)

Referenced by [jit_linklist_findall\(\)](#), and [jit_linklist_findcount\(\)](#).

36.11.2.9 t_atom_long linklist_findfirst (t_linklist * x, void ** o, long cmpfnvoid *, 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

<i>x</i>	The linklist instance.
<i>o</i>	The address to pointer that will be set with the matching item.
<i>cmpfn</i>	The function used to determine a match in the list.
<i>cmpdata</i>	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.

```
1 void *obj;
2 long index;
3
4 index = linklist_findfirst(x, &obj, #linklist_match, o);
5 if(index != -1)
6     linklist_chuckindex(x, index);
```

See also

[linklist_match](#)
[t_cmpfn](#)
[linklist_findall](#)

Referenced by [jit_linklist_findfirst\(\)](#).

36.11.2.10 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

<i>x</i>	The linklist instance.
<i>flags</i>	A valid value from the e_max_datastore_flags . The default is OBJ_FLAG_OBJ .

36.11.2.11 void linklist_funall (t_linklist *x, method fun, void *arg)

Call the specified function for every item in the linklist.

Parameters

<i>x</i>	The linklist instance.
<i>fun</i>	The function to call, specified as function pointer cast to a Max method .
<i>arg</i>	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\(\)](#).

```
1 void myFun(t_object *myObj, void *myArg)
2 {
3     // do something with myObj and myArg here
4     // myObj is the item in the linklist
5 }
```

36.11.2.12 `t_atom_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

<i>x</i>	The linklist instance.
<i>fun</i>	The function to call, specified as function pointer cast to a Max method .
<i>arg</i>	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\(\)](#).

```

1 long myFun(t_symbol *myListSymbol, void *myArg)
2 {
3     // this function is called by a linklist that contains symbols for its items
4     if(myListSymbol == gensym(""))
5         error("empty symbol -- aborting linklist traversal")
6     return 1;
7 }
8 else{
9     // do something with the symbol
10    return 0;
11 }
12 }
```

36.11.2.13 `void* linklist_funindex (t_linklist *x, long i, method fun, void *arg)`

Call the specified function for an item specified by index.

Parameters

<i>x</i>	The linklist instance.
<i>i</i>	The index of the item to which to send the message.
<i>fun</i>	The function to call, specified as function pointer cast to a Max method .
<i>arg</i>	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\(\)](#).

```

1 void myFun(t_object *myObj, void *myArg)
2 {
3     // do something with myObj and myArg here
4     // myObj is the item in the linklist
5 }
```

36.11.2.14 `t_atom_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](#).

36.11.2.15 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

Referenced by [jit_linklist_getindex\(\)](#).

36.11.2.16 t_atom_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.

Referenced by [jit_linklist_findcount\(\)](#), and [jit_linklist_getsize\(\)](#).

36.11.2.17 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.

36.11.2.18 t_llitem* linklist_insertafterobjptr (t_linklist *x, void *o, void *objptr)

Insert an item into the list after another specified item.

Parameters

<i>x</i>	The linklist instance.
<i>o</i>	The item pointer to insert.
<i>objptr</i>	The item pointer after which to insert in the list.

Returns

An opaque linklist element.

36.11.2.19 `t_llitem* linklist_insertbeforeobjptr (t_linklist *x, void *o, void *objptr)`

Insert an item into the list before another specified item.

Parameters

<i>x</i>	The linklist instance.
<i>o</i>	The item pointer to insert.
<i>objptr</i>	The item pointer before which to insert in the list.

Returns

An opaque linklist element.

36.11.2.20 `t_atom_long linklist_insertindex (t_linklist *x, void *o, long index)`

Insert an item into the list at the specified index.

Parameters

<i>x</i>	The linklist instance.
<i>o</i>	The item pointer to insert.
<i>index</i>	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.

Referenced by `jit_linklist_insertindex()`.

36.11.2.21 `void* linklist_last (t_linklist *x, void **item)`

Return the last item (the tail) in the linked-list.

Parameters

<i>x</i>	The linklist instance.
<i>item</i>	The address of pointer in which to store the last item in the linked-list.

Returns

always returns NULL

36.11.2.22 `t_atom_long linklist_makearray (t_linklist *x, void **a, long max)`

Retrieve linklist items as an array of pointers.

Parameters

<i>x</i>	The linklist instance.
<i>a</i>	The address of the first pointer in the array to fill.
<i>max</i>	The number of pointers in the array.

Returns

The number of items from the list actually returned in the array.

Referenced by [jit_linklist_makearray\(\)](#).

36.11.2.23 long linklist_match (void * *a*, void * *b*)

A linklist comparison method that determines if two item pointers are equal.

Parameters

<i>a</i>	The first item to compare.
<i>b</i>	The second item to compare.

Returns

Returns 1 if the items are equal, otherwise 0.

See also

[t_cmpfn](#)

36.11.2.24 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

<i>x</i>	The linklist instance.
<i>s</i>	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.

Referenced by [jit_linklist_methodall\(\)](#).

36.11.2.25 void* linklist_methodindex (t_linklist * *x*, t_atom_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

<i>x</i>	The linklist instance.
<i>i</i>	The index of the item to which to send the message.
<i>s</i>	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.

Referenced by [jit_linklist_methodindex\(\)](#).

36.11.2.26 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

<i>x</i>	The linklist instance.
<i>o</i>	The item pointer to insert.
<i>objptr</i>	The item pointer after which to move o in the list.

Returns

An opaque linklist element.

36.11.2.27 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

<i>x</i>	The linklist instance.
<i>o</i>	The item pointer to insert.
<i>objptr</i>	The item pointer before which to move o in the list.

Returns

An opaque linklist element.

36.11.2.28 BEGIN_USING_C_LINKAGE 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\(\)](#)

Referenced by [jit_linklist_new\(\)](#).

36.11.2.29 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

<i>x</i>	The linklist instance.
<i>p</i>	An item in the list.
<i>next</i>	The address of a pointer to set with the next item in the list.

36.11.2.30 *t_atom_long* linklist_objptr2index (*t_linklist* * *x*, void * *p*)

Return an item's index, given the item itself.

Parameters

<i>x</i>	The linklist instance.
<i>p</i>	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.

Referenced by [jit_linklist_objptr2index\(\)](#).

36.11.2.31 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

<i>x</i>	The linklist instance.
<i>p</i>	An item in the list.
<i>prev</i>	The address of a pointer to set with the previous item in the list.

36.11.2.32 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

<i>x</i>	The linklist instance.
<i>readonly</i>	A 1 or 0 for setting the readonly bit.

36.11.2.33 void linklist_reverse (*t_linklist* * *x*)

Reverse the order of items in the linked-list.

Parameters

x	The linklist instance.
---	------------------------

Referenced by jit_linklist_reverse().

36.11.2.34 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.

Referenced by jit_linklist_rotate().

36.11.2.35 void linklist_shuffle (t_linklist *x)

Randomize the order of items in the linked-list.

Parameters

x	The linklist instance.
---	------------------------

Referenced by jit_linklist_shuffle().

36.11.2.36 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 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\(\)](#).

```

1 long myAlphabeticalCmpfn(void *a, void *b)
2 {
3     t_symbol *s1 = (t_symbol *)a;
4     t_symbol *s2 = (t_symbol *)b;
5
6     if(s1->s_name[0] < s2->s_name[0])
7         return true;
8     else
9         return false;
10 }
11
12 void mySortMethod(t_myobj *x)
13 {
14     // the linklist was already created and filled with items previously
15     linklist_sort(x->myLinkList, myAlphabeticalCmpfn);
16 }
```

Referenced by jit_linklist_sort().

36.11.2.37 void* linklist_substitute (t_linklist *x, void *p, void *newp)

Given an item in the list, replace it with a different value.

Parameters

<i>x</i>	The linklist instance.
<i>p</i>	An item in the list.
<i>newp</i>	The new value.

Returns

Always returns NULL.

36.11.2.38 void linklist_swap (t_linklist *x, long a, long b)

Swap the position of two items in the linked-list, specified by index.

Parameters

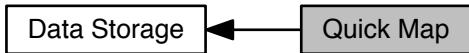
<i>x</i>	The linklist instance.
<i>a</i>	The index of the first item to swap.
<i>b</i>	The index of the second item to swap.

Referenced by jit_linklist_swap().

36.12 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.

36.12.1 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.

36.12.2 Function Documentation

36.12.2.1 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

<i>x</i>	The quickmap instance.
<i>p1</i>	The (first) key.
<i>p2</i>	The value (or the second key).

Returns

A Max error code.

36.12.2.2 void quickmap_drop (t_quickmap * *x*, void * *p1*, void * *p2*)

Drop a pair of keys mapped to each other in the quickmap.

Parameters

<i>x</i>	The quickmap instance.
<i>p1</i>	The first key.
<i>p2</i>	The second key.

Returns

A Max error code.

36.12.2.3 long quickmap_lookup_key1 (t_quickmap * *x*, void * *p1*, void ** *p2*)

Given a (first) key, lookup the value (the second key).

Parameters

<i>x</i>	The quickmap instance.
<i>p1</i>	The (first) key.
<i>p2</i>	The address of a pointer which will hold the resulting key upon return.

Returns

A Max error code.

36.12.2.4 long quickmap_lookup_key2 (t_quickmap * *x*, void * *p1*, void ** *p2*)

Given a (second) key, lookup the value (the first key).

Parameters

<i>x</i>	The quickmap instance.
<i>p1</i>	The (second) key.
<i>p2</i>	The address of a pointer which will hold the resulting key upon return.

Returns

A Max error code.

36.12.2.5 BEGIN_USING_C_LINKAGE void* quickmap_new (void)

Create a new quickmap object.

Returns

Pointer to the new quickmap object.

36.12.2.6 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

<i>x</i>	The quickmap instance.
<i>way</i>	Set to true to make the quickmap readonly (disable thread protection) or false (the default) to enable thread protection.

36.13 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 \(t_string *x\)](#)
Fetch a pointer to a string object's internal C-string.
- [void string_reserve \(t_string *x, long numbytes\)](#)
Reserve additional memory for future string growth.
- [void string_append \(t_string *x, const char *s\)](#)
Append a C-string onto the existing string maintained by a [t_string](#) object.
- [void string_chop \(t_string *x, long numchars\)](#)
Shorten a string by eliminating N characters from the end.

36.13.1 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](#)

36.13.2 Function Documentation

36.13.2.1 void string_append (t_string * x, const char * s)

Append a C-string onto the existing string maintained by a [t_string](#) object.

Memory allocation for the string will grow as needed to hold the concatenated string.

Parameters

<i>x</i>	The string object instance.
<i>s</i>	A string to append/concatenate with the existing string.

36.13.2.2 void string_chop (*t_string* * *x*, long *numchars*)

Shorten a string by eliminating N characters from the end.

Parameters

<i>x</i>	The string object instance.
<i>numchars</i>	The number of characters to chop from the end of the string.

36.13.2.3 const char* string_getptr (*t_string* * *x*)

Fetch a pointer to a string object's internal C-string.

Parameters

<i>x</i>	The string object instance.
----------	-----------------------------

Returns

A pointer to the internally maintained C-string.

36.13.2.4 *t_string string_new (const char * *psz*)**

Create a new string object.

Parameters

<i>psz</i>	Pointer to a C-string that will be copied to memory internal to this string object instance.
------------	--

Returns

The new string object instance pointer.

36.13.2.5 void string_reserve (*t_string* * *x*, long *numbytes*)

Reserve additional memory for future string growth.

Parameters

<i>x</i>	The string object instance.
<i>numbytes</i>	The total number of bytes to allocate for this string object.

36.14 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.

36.14.1 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

36.14.2 Function Documentation

36.14.2.1 long `symobject_linklist_match (void * a, void * b)`

Utility for searching a linklist containing symobjects.

Parameters

<code>a</code>	(opaque)
<code>b</code>	(opaque)

Returns

Returns true if a match is found, otherwise returns false.

Remarks

The following example shows one common use of the this method.

```
1 t_symobject *item = NULL;
2 long      index;
3 t_symbol   *textsym;
4
5 textsym = gensym("something to look for");
6
7 // search for a symobject with the symbol 'something to look for'
8 index = linklist_findfirst(s_ll_history, (void **)&item, symobject_linklist_match, textsym);
9 if(index == -1){
10     // symobject not found.
11 }
12 else{
13     do something with the symobject, or with the index of the symobject in the linklist
14 }
```

36.14.2.2 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.

36.15 Dictionary Passing API

The Dictionary Passing API defines a means by which `t_dictionary` instances may be passed between Max objects in a way similar to the way Jitter Matrices are passed between objects.

Collaboration diagram for Dictionary Passing API:



Functions

- `BEGIN_USING_C_LINKAGE t_dictionary * dictobj_register (t_dictionary *d, t_symbol **name)`
Register a `t_dictionary` with the dictionary passing system and map it to a unique name.
- `t_max_err dictobj_unregister (t_dictionary *d)`
Unregister a `t_dictionary` with the dictionary passing system.
- `t_dictionary * dictobj_findregistered_clone (t_symbol *name)`
Find the `t_dictionary` for a given name, and return a copy of that dictionary. When you are done, do not call `dictobj->release()` on the dictionary, because you are working on a copy rather than on a retained pointer.
- `t_dictionary * dictobj_findregistered_retain (t_symbol *name)`
Find the `t_dictionary` for a given name, return a pointer to that `t_dictionary`, and increment its reference count.
- `t_max_err dictobj_release (t_dictionary *d)`
For a `t_dictionary`/name that was previously retained with `dictobj_findregistered_retain()`, release it (decrement its reference count).
- `t_symbol * dictobj_namefromptr (t_dictionary *d)`
Find the name associated with a given `t_dictionary`.
- `void dictobj_outlet_atoms (void *out, long argc, t_atom *argv)`
Send atoms to an outlet in your Max object, handling complex datatypes that may be present in those atoms.
- `long dictobj_atom_safety (t_atom *a)`
Ensure that an atom is safe for passing.
- `long dictobj_atom_safety_flags (t_atom *a, long flags)`
Ensure that an atom is safe for passing.
- `long dictobj_validate (const t_dictionary *schema, const t_dictionary *candidate)`
Validate the contents of a `t_dictionary` against a second `t_dictionary` containing a schema.
- `t_max_err dictobj_jsonfromstring (long *jsonsize, char **json, const char *str)`
Convert a C-string of `Dictionary Syntax` into a C-string of JSON.
- `t_max_err dictobj_dictionaryfromstring (t_dictionary **d, const char *str, int str_is_already_json, char *errorstring)`
Create a new `t_dictionary` from `Dictionary Syntax` which is passed in as a C-string.
- `t_max_err dictobj_dictionaryfromatoms (t_dictionary **d, const long argc, const t_atom *argv)`
Create a new `t_dictionary` from `Dictionary Syntax` which is passed in as an array of atoms.
- `t_max_err dictobj_dictionaryfromatoms_extended (t_dictionary **d, const t_symbol *msg, long argc, const t_atom *argv)`
Create a new `t_dictionary` from an array of atoms that use Max dictionary syntax, JSON, or compressed JSON.
- `t_max_err dictobj_dictionarytoatoms (const t_dictionary *d, long *argc, t_atom **argv)`
Serialize the contents of a `t_dictionary` into `Dictionary Syntax`.

- `t_max_err dictobj_key_parse (t_object *x, t_dictionary *d, t_atom *akey, t_bool create, t_dictionary **targetdict, t_symbol **targetkey, t_int32 *index)`

Given a complex key (one that includes potential heirarchy or array-member access), return the actual key and the dictionary in which the key should be referenced.

36.15.1 Detailed Description

The Dictionary Passing API defines a means by which `t_dictionary` instances may be passed between Max objects in a way similar to the way Jitter Matrices are passed between objects.

There are important differences, however, between Jitter matrix passing and dictionary passing. Many of these differences are documented in Max's documentation on dictionaries and structured data.

Every dictionary instance in this system is mapped to a unique name that identifies the dictionary. Dictionaries are passed between objects using the "dictionary" message with a single argument, which is the name of the dictionary.

36.15.2 Registration and Access

The C-API for working with these dictionaries is composed of 5 primary registration/access methods:

- `dictobj_register()` : register a `t_dictionary` instance with the system, and map the instance to a name
- `dictobj_unregister()` : unregister a `t_dictionary` from the system
- `dictobj_findregistered_clone()` : find the `t_dictionary` for a given name, and return a *copy* of that dictionary
- `dictobj_findregistered_retain()` : find the `t_dictionary` for a given name, return a pointer to that `t_dictionary`, and increment its reference count.
- `dictobj_release()` : for a `t_dictionary`/name that was previously retained with `dictobj_findregistered_retain()`, release it (decrement its reference count).

It is useful to think of objects in the dictionary system as "nouns" and "verbs".

A "noun" is an object that possess or owns a dictionary. These objects are servers whose dictionary will accessed by other object that are clients. An example of a "noun" is the dict.pack object that creates a dictionary that is passed to other objects.

A "verb" is an object that does not maintain its own dictionary (it is not a thing) but merely does something to any dictionaries it receives. This object is a client rather than a server. An example of a "verb" is the dict.strip object, which removes entries from an existing dictionary but possesses no dictionary of its own.

Any object which is a dictionary "noun", can keep and rely on their dictionary pointer. Because of the way `object->register()` works, there should be no possibility for this pointer to change behind the scenes. They each need to call `object_free()` on their respective object pointer, however. A call to `object_free()` also calls `object_unregister()` once, so there's technically not a need to unregister from the owner itself. They work like jit.matrix (and similar to buffer~), and use `object_register()` to increment a server reference count. If an object has already registered an object with the given name, the pointer passed in to register is freed and the existing one is returned from the registration function.

Dictionary "verbs" on the other hand should just call `dict_findregistered_retain()` and `dict_release()` when done. They are not incrementing the server reference count. They increment a reference count with regards to object freeing, which is compatible with and complementary to the server reference count.

36.15.3 Dictionary Syntax

Dictionaries may be represented in a variety of textual formats including JSON. Max also supports a compact Y \leftarrow AML-like dictionary notation which is useful for proving data structure contents as lists of atoms in object boxes. This format is documented in Max's documentation of the dictionary features. The following functions are used for formatting and parsing the dictionary syntax.

- [dictobj_jsonfromstring\(\)](#)
- [dictobj_dictionaryfromstring\(\)](#)
- [dictobj_dictionaryfromatoms\(\)](#)
- [dictobj_dictionarytoatoms\(\)](#)

36.15.4 Utilities

There are several utility functions available to assist in coding objects that pass dictionaries.

- [dictobj_outlet_atoms\(\)](#)
- [dictobj_atom_safety\(\)](#)
- [dictobj_validate\(\)](#)

The [dictobj_validate\(\)](#) object is a utility routine for validating a dictionary against "schema" dictionary. This enables a behavior somewhat analogous to Objective-C or Smalltalk prototypes. Dictionary validation can be useful to implement a kind of dictionary polymorphism. For a multiple-inheritance behavior, simply validate a dictionary against multiple schemas to verify the presence of required keys and values.

36.15.5 Limitations

The [dict_outlet_atoms\(\)](#) function will not output [A_OBJ](#) atoms directly (nor should any other object) and as such it will also not output [t_atomarray](#) instances containing objects, thus atomarrays are *not* hierarchical in the dictionary passing implementation.

It will output an atom array if provided a single [A_OBJ](#) atom with class atomarray. If there is an array of atoms which contain [A_OBJ](#) atoms, they are converted to the *symbols* <dictionary-object>, <atomarray-object>, <string-object>, <other-object> respectively. Ideally such a case should never be reached if everything which inserts values into a dictionary is well behaved—i.e.

- a key may be a single atom
- a key may be an atomarray (but no [A_OBJ](#) atoms)
- a key may be a dictionary

Version

6.0

36.15.6 Function Documentation

36.15.6.1 long dictobj_atom_safety (t_atom * a)

Ensure that an atom is safe for passing.

Atoms are allowed to be [A_LONG](#), [A_FLOAT](#), or [A_SYM](#), but not [A_OBJ](#). If the atom is an [A_OBJ](#), it will be converted into something that will be safe to pass.

Parameters

<i>a</i>	An atom to check, and potentially modify, to ensure safety in the dictionary-passing system.
----------	--

Returns

If the atom was changed then 1 is returned. Otherwise 0 is returned.

36.15.6.2 long dictobj_atom_safety_flags (t_atom * *a*, long *flags*)

Ensure that an atom is safe for passing.

Atoms are allowed to be [A_LONG](#), [A_FLOAT](#), or [A_SYM](#), but not [A_OBJ](#). If the atom is an [A_OBJ](#), it will be converted into something that will be safe to pass.

Parameters

<i>a</i>	An atom to check, and potentially modify, to ensure safety in the dictionary-passing system.
<i>flags</i>	Pass DICTOBJ_ATOM_FLAGS_REGISTER to have dictionary atoms registered/retained.

Returns

If the atom was changed then 1 is returned. Otherwise 0 is returned.

36.15.6.3 t_max_err dictobj_dictionaryfromatoms (t_dictionary ** *d*, const long *argc*, const t_atom * *argv*)

Create a new [t_dictionary](#) from [Dictionary Syntax](#) which is passed in as an array of atoms.

Unlike many [t_dictionary](#) calls to create dictionaries, this function does not take over ownership of the atoms you pass in.

Parameters

<i>d</i>	The address of a dictionary variable, which will hold a pointer to the new dictionary upon return. Should be initialized to NULL.
<i>argc</i>	The number of atoms in <i>argv</i> .
<i>argv</i>	Pointer to the first of an array of atoms to be interpreted as Dictionary Syntax .

Returns

A Max error code.

See also

[dictobj_dictionaryfromatoms_extended\(\)](#) [dictobj_dictionarytoatoms\(\)](#)

36.15.6.4 t_max_err dictobj_dictionaryfromatoms_extended (t_dictionary ** *d*, const t_symbol * *msg*, long *argc*, const t_atom * *argv*)

Create a new [t_dictionary](#) from from an array of atoms that use Max dictionary syntax, JSON, or compressed JSON.

This function is the C analog to the dict.deserialize object in Max. Unlike many [t_dictionary](#) calls to create dictionaries, this function does not take over ownership of the atoms you pass in.

Parameters

<i>d</i>	The address of a dictionary variable, which will hold a pointer to the new dictionary upon return. Should be initialized to NULL.
<i>msg</i>	Ignored.
<i>argc</i>	The number of atoms in argv.
<i>argv</i>	Pointer to the first of an array of atoms to be interpreted as Dictionary Syntax , JSON, or compressed JSON.

Returns

A Max error code.

See also

[dictobj_dictionaryfromatoms\(\)](#) [dictobj_dictionaryfromstring\(\)](#)

36.15.6.5 t_max_err dictobj_dictionaryfromstring (t_dictionary ** d, const char * str, int str_is_already_json, char * errorstring)

Create a new [t_dictionary](#) from [Dictionary Syntax](#) which is passed in as a C-string.

Parameters

<i>d</i>	The address of a dictionary variable, which will hold a pointer to the new dictionary upon return. Should be initialized to NULL.
<i>str</i>	A NULL-terminated C-string containing Dictionary Syntax .
<i>str_is_already_json</i>	
<i>errorstring</i>	

Returns

A Max error code.

See also

[dictobj_dictionarytoatoms\(\)](#)

36.15.6.6 t_max_err dictobj_dictionarytoatoms (const t_dictionary * d, long * argc, t_atom ** argv)

Serialize the contents of a [t_dictionary](#) into [Dictionary Syntax](#).

Parameters

<i>d</i>	The dictionary to serialize.
<i>argc</i>	The address of a variable to hold the number of atoms allocated upon return.
<i>argv</i>	The address of a t_atom pointer which will point to the first atom (of an array of argc atoms) upon return.

Returns

A Max error code.

See also

[dictobj_dictionaryfromatoms\(\)](#)

36.15.6.7 `t_dictionary* dictobj_findregistered_clone(t_symbol * name)`

Find the [t_dictionary](#) for a given name, and return a *copy* of that dictionary When you are done, do *not* call [dictobj->release\(\)](#) on the dictionary, because you are working on a copy rather than on a retained pointer.

Parameters

<i>name</i>	The name associated with the dictionary for which you wish to obtain a copy.
-------------	--

Returns

The dictionary cloned from the existing dictionary. Returns NULL if no dictionary is associated with name.

See also

[dictobj_findregistered_retain\(\)](#)

36.15.6.8 t_dictionary* dictobj_findregistered_retain (t_symbol * name)

Find the [t_dictionary](#) for a given name, return a pointer to that [t_dictionary](#), and increment its reference count.

When you are done you should call [dictobj_release\(\)](#) on the dictionary.

Parameters

<i>name</i>	The name associated with the dictionary for which you wish to obtain a pointer.
-------------	---

Returns

A pointer to the dictionary associated with name. Returns NULL if no dictionary is associated with name.

See also

[dictobj_release\(\)](#)
[dictobj_findregistered_clone\(\)](#)

36.15.6.9 t_max_err dictobj_jsonfromstring (long * jsonsize, char ** json, const char * str)

Convert a C-string of [Dictionary Syntax](#) into a C-string of JSON.

Parameters

<i>jsonsize</i>	The address of a variable to be filled-in with the number of chars in json upon return.
<i>json</i>	The address of a char pointer to point to the JSON C-string upon return. Should be initialized to NULL. You are responsible for freeing the string with sysmem_freeptr() when you are done with it.
<i>str</i>	A NULL-terminated C-string containing Dictionary Syntax .

Returns

A Max error code.

See also

[dictobj_dictionarytoatoms\(\)](#)

36.15.6.10 t_max_err dictobj_key_parse (t_object * x, t_dictionary * d, t_atom * akey, t_bool create, t_dictionary ** targetdict, t_symbol ** targetkey, t_int32 * index)

Given a complex key (one that includes potential heirarchy or array-member access), return the actual key and the dictionary in which the key should be referenced.

Parameters

<i>x</i>	Your calling object. If there is an error this will be used by the internal call to object_error() .
<i>d</i>	The dictionary you are querying.
<i>akey</i>	The complex key specifying the query.
<i>create</i>	If true, create the intermediate dictionaries in the hierarchy specified in akey.
<i>targetdict</i>	Returns the t_dictionary that for the (sub)dictionary specified by akey.
<i>targetkey</i>	Returns the name of the key in targetdict that to which akey is referencing.
<i>index</i>	Returns the index requested if array-member access is specified. Pass NULL if you are not interested in this.

Returns

A Max error code.

36.15.6.11 t_symbol* dictobj_namefromptr (t_dictionary * d)

Find the name associated with a given [t_dictionary](#).

Parameters

<i>d</i>	A dictionary, whose name you wish to determine.
----------	---

Returns

The symbol associated with the dictionary, or NULL if the dictionary is not registered.

See also

[dictobj_register\(\)](#)

36.15.6.12 void dictobj_outlet_atoms (void * out, long argc, t_atom * argv)

Send atoms to an outlet in your Max object, handling complex datatypes that may be present in those atoms.

This is particularly when sending the contents of a dictionary entry out of an outlet as in the following example code.

```

1 long      ac = 0;
2 t_atom    *av = NULL;
3 t_max_err err;
4
5 err = dictionary_copyatoms(d, key, &ac, &av);
6 if (!err && ac && av) {
7     // handles singles, lists, symbols, atomarrays, dictionaries, etc.
8     dictobj_outlet_atoms(x->outlets[i],ac,av);
9 }
10
11 if (av)
12     sysmem_freeptr(av);

```

Parameters

<i>out</i>	The outlet through which the atoms should be sent.
<i>argc</i>	The count of atoms in argv.
<i>argv</i>	Pointer to the first of an array of atoms to send to the outlet.

36.15.6.13 BEGIN_USING_C_LINKAGE t_dictionary* dictobj_register (t_dictionary * d, t_symbol ** name)

Register a [t_dictionary](#) with the dictionary passing system and map it to a unique name.

Parameters

<i>d</i>	A valid dictionary object.
<i>name</i>	The address of a t_symbol pointer to the name you would like mapped to this dictionary. If the t_symbol pointer has a NULL value then a unique name will be generated and filled-in upon return.

Returns

The dictionary mapped to the specified name.

36.15.6.14 t_max_err dictobj_release (t_dictionary * d)

For a [t_dictionary](#)/name that was previously retained with [dictobj_findregistered_retain\(\)](#), release it (decrement its reference count).

Parameters

<i>d</i>	A valid dictionary object retained by dictobj_findregistered_retain() .
----------	---

Returns

A Max error code.

See also

[dictobj_findregistered_retain\(\)](#)

36.15.6.15 t_max_err dictobj_unregister (t_dictionary * d)

Unregister a [t_dictionary](#) with the dictionary passing system.

Generally speaking you should not need to call this method. Calling [object_free\(\)](#) on the [t_dictionary](#) automatically unregisters it.

Parameters

<i>d</i>	A valid dictionary object.
----------	----------------------------

Returns

A Max error code.

36.15.6.16 long dictobj_validate (const t_dictionary * schema, const t_dictionary * candidate)

Validate the contents of a [t_dictionary](#) against a second [t_dictionary](#) containing a schema.

The schema dictionary contains keys and values, like any dictionary. [dictobj_validate\(\)](#) checks to make sure that all keys in the schema dictionary are present in the candidate dictionary. If the keys are all present then the candidate passes and the function returns true. Otherwise the the candidate fails the validation and the function returns false.

Generally speaking, the schema dictionary with contain values with the symbol "*", indicating a wildcard, and thus only the key is used to validate the dictionary (all values match the wildcard). However, if the schema dictionary contains non-wildcard values for any of its keys, those keys in the candidate dictionary must also contain matching values in order for the candidate to successfully validate.

An example of this in action is the dict.route object in Max, which simply wraps this function.

Parameters

<i>schema</i>	The dictionary against which to validate candidate.
<i>candidate</i>	A dictionary to test against the schema.

Returns

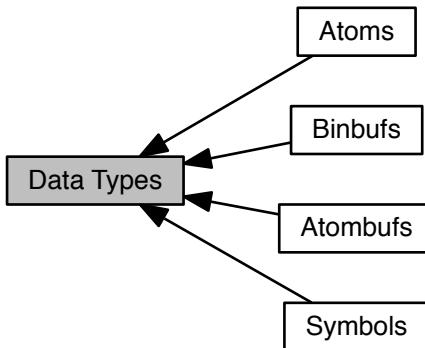
Returns true if the candidate validates against the schema, otherwise returns false.

See also

[dictobj_dictionarytoatoms\(\)](#)

36.16 Data Types

Collaboration diagram for Data Types:



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.

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.

Typedefs

- [typedef void *\(* method\)](#) (void *,...)
Function pointer type for generic methods.
- [typedef long\(* t_intmethod\)](#) (void *,...)
Function pointer type for methods returning a long.
- [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.

36.16.1 Detailed Description

36.17 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.

Macros

- #define `ATOM_MAX_STRLEN`
Defines the largest possible string size for an atom.

Enumerations

- enum `e_max_atomtypes` {
 `A_NOTHING`, `A_LONG`, `A_FLOAT`, `A_SYM`,
 `A_OBJ`, `A_DEFLONG`, `A_DEFFLOAT`, `A_DEFSYM`,
 `A_GIMME`, `A_CANT`, `A_SEMI`, `A_COMM`,
 `A_DOLLAR`, `A_DOLLSYM`, `A_GIMMEBACK`, `A_DEFER`,
 `A_USURP`, `A_DEFER_LOW`, `A_USURP_LOW` }

the list of officially recognized types, including pseudotypes for commas and semicolons.
- enum `e_max_atom_gettext_flags` {
 `OBEX_UTIL_ATOM_GETTEXT_DEFAULT`, `OBEX_UTIL_ATOM_GETTEXT_TRUNCATE_ZEROS`, `OBEX_UTIL_ATOM_GETTEXT_SYM_NO_QUOTE`, `OBEX_UTIL_ATOM_GETTEXT_SYM_FORCE_QUOTE`,
 `OBEX_UTIL_ATOM_GETTEXT_COMMAS_DELIM`, `OBEX_UTIL_ATOM_GETTEXT_FORCE_ZEROS`, `OBEX_UTIL_ATOM_GETTEXT_NUM_HI_RES`, `OBEX_UTIL_ATOM_GETTEXT_NUM_LO_RES` }

Flags that determine how functions convert atoms into text (C-strings).

Functions

- `t_max_err atom_setlong (t_atom *a, t_atom_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`.

- `t_atom_long atom_getlong (const t_atom *a)`
Retrieves a long integer value from a `t_atom`.
- `t_atom_float atom_getfloat (const t_atom *a)`
Retrieves a floating point value from a `t_atom`.
- `t_symbol * atom_getsym (const t_atom *a)`
*Retrieves a `t_symbol` * value from a `t_atom`.*
- `void * atom_getobj (const t_atom *a)`
Retrieves a generic pointer value from a `t_atom`.
- `long atom_getcharfix (const t_atom *a)`
Retrieves an unsigned integer value between 0 and 255 from a `t_atom`.
- `long atom_gettype (const t_atom *a)`
Retrieves type from a `t_atom`.
- `t_max_err atom_arg_getlong (t_atom_long *c, long idx, long ac, const 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, const 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, const 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, const 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, t_atom_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, t_atom_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` (const `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` (long argc1, `t_atom` *argv1, `t_atom` *argv2)

Copy an array of atoms.
 - void `postargs` (long 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.

36.17.1 Detailed Description

36.17.2 Enumeration Type Documentation

36.17.2.1 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
- `OBEX_UTIL_ATOM_GETTEXT_NUM_LO_RES`** // print fewer decimal places (HI_RES will win though)

36.17.2.2 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.

36.17.3 Function Documentation

36.17.3.1 `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

<i>ac</i>	The address of a variable that will contain the number of atoms allocated (1).
<i>av</i>	The address of a pointer that will be set with the new allocated memory for the atom.
<i>alloc</i>	Address of a variable that will be set true if memory is allocated, otherwise false.

Returns

A Max error code.

36.17.3.2 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

<i>minsize</i>	The minimum number of atoms that this array will need to contain. This determines the amount of memory allocated.
<i>ac</i>	The address of a variable that will contain the number of atoms allocated.
<i>av</i>	The address of a pointer that will be set with the new allocated memory for the atoms.
<i>alloc</i>	Address of a variable that will be set true if memory is allocated, otherwise false.

Returns

A Max error code.

36.17.3.3 long atom_arg_getdouble (double * *c*, long *idx*, long *ac*, const 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

<i>c</i>	Pointer to a double variable to receive the atom's data if the function is successful. Otherwise the value is left unchanged.
<i>idx</i>	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, <i>idx</i> should be set to 2.
<i>ac</i>	Count of <i>av</i> .
<i>av</i>	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.

Referenced by [jit_atom_arg_getdouble\(\)](#).

36.17.3.4 long atom_arg_getfloat (float * *c*, long *idx*, long *ac*, const t_atom * *av*)

Retrieves the floating point value of a particular [t_atom](#) from an atom list, if the atom exists.

Parameters

<i>c</i>	Pointer to a float variable to receive the atom's data if the function is successful. Otherwise, the value is left unchanged.
<i>idx</i>	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, <i>idx</i> should be set to 2.
<i>ac</i>	Count of av.
<i>av</i>	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.

Referenced by [jit_atom_arg_getfloat\(\)](#).

36.17.3.5 t_max_err atom_arg_getlong (t_atom_long * c, long idx, long ac, const t_atom * av)

Retrieves the integer value of a particular [t_atom](#) from an atom list, if the atom exists.

Parameters

<i>c</i>	Pointer to a long variable to receive the atom's data if the function is successful.
<i>idx</i>	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, <i>idx</i> should be set to 2.
<i>ac</i>	Count of av.
<i>av</i>	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:

```

1 void myobject_mymessage(t_myobject *x, t_symbol *s, long ac, t_atom *av)
2 {
3     t_atom_long var = -1;
4
5     // here, we are expecting a value of 0 or greater
6     atom_arg_getlong(&var, 0, ac, av);
7     if (val == -1) // i.e. unchanged
8         post("it is likely that the user did not provide a valid argument");
9     else {
10         ...
11     }
12 }
```

Referenced by [jit_atom_arg_getlong\(\)](#).

36.17.3.6 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

<i>x</i>	The address of a pointer to the object contained in <i>av</i> if it is of the specified class upon return. Otherwise NULL upon return.
<i>idx</i>	The index of the atom in the array from which to get the object pointer.
<i>argc</i>	The count of atoms in <i>argv</i> .
<i>argv</i>	The address to the first of an array of atoms.
<i>cls</i>	A symbol containing the class name of which the object should be an instance.

Returns

A Max error code.

36.17.3.7 long atom_arg_getsym (t_symbol ** c, long idx, long ac, const t_atom * av)

Retrieves the *t_symbol* * value of a particular *t_atom* from an atom list, if the atom exists.

Parameters

<i>c</i>	Pointer to a <i>t_symbol</i> * variable to receive the atom's data if the function is successful. Otherwise, the value is left unchanged.
<i>idx</i>	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, <i>idx</i> should be set to 2.
<i>ac</i>	Count of <i>av</i> .
<i>av</i>	Pointer to the first <i>t_atom</i> 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:

```

1 void myobject_open(t_myobject *x, t_symbol *s, long ac, t_atom *av)
2 {
3     t_symbol *filename = _sym_nothing;
4
5     // here, we are expecting a file name.
6     // if we don't get it, open a dialog box
7     atom_arg_getsym(&filename, 0, ac, av);
8     if (filename == _sym_nothing) { // i.e. unchanged
9         // open the file dialog box,
10        // get a value for filename
11    }
12    // do something with the filename
13 }
```

Referenced by [jit_atom_arg_getsym\(\)](#).

36.17.3.8 BEGIN_USING_C_LINKAGE void atom_copy (long argc1, t_atom * argv1, t_atom * argv2)

Copy an array of atoms.

Parameters

<i>argc1</i>	The count of atoms in argv1.
<i>argv1</i>	The address to the first of an array of atoms that is the source for the copy.
<i>argv2</i>	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 <code>sysmem_newptr()</code> or <code>atom_alloc()</code> .

36.17.3.9 `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

<i>ac</i>	The number of atoms allocated in the <i>av</i> parameter.
<i>av</i>	The address to the first of an array of allocated atoms.
<i>count</i>	The number of values to fetch from the array specified by <i>vals</i> .
<i>vals</i>	The address of the array to which is copied the values from <i>av</i> .

Returns

A Max error code.

36.17.3.10 `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

<i>ac</i>	The number of atoms allocated in the <i>av</i> parameter.
<i>av</i>	The address to the first of an array of allocated atoms.
<i>count</i>	The number of values to fetch from the array specified by <i>vals</i> .
<i>vals</i>	The address of the array to which is copied the values from <i>av</i> .

Returns

A Max error code.

36.17.3.11 `long atom_getcharfix(const t_atom * a)`

Retrieves an unsigned integer value between 0 and 255 from a `t_atom`.

Parameters

<i>a</i>	Pointer to a <code>t_atom</code> 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.

Referenced by `jit_atom_getcharfix()`.

36.17.3.12 `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

<code>ac</code>	The number of atoms allocated in the <code>av</code> parameter.
<code>av</code>	The address to the first of an array of allocated atoms.
<code>count</code>	The number of values to fetch from the array specified by <code>vals</code> .
<code>vals</code>	The address of the array to which is copied the values from <code>av</code> .

Returns

A Max error code.

36.17.3.13 `t_atom_float atom_getfloat(const t_atom * a)`

Retrieves a floating point value from a `t_atom`.

Parameters

<code>a</code>	Pointer to a <code>t_atom</code> 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.

Referenced by `jit_atom_getfloat()`.

36.17.3.14 `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

<code>ac</code>	The number of atoms allocated in the <code>av</code> parameter.
<code>av</code>	The address to the first of an array of allocated atoms.
<code>count</code>	The number of values to fetch from the array specified by <code>vals</code> .
<code>vals</code>	The address of the array to which is copied the values from <code>av</code> .

Returns

A Max error code.

36.17.3.15 `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

<i>ac</i>	The number of atoms to parse in <i>av</i> .
<i>av</i>	The address of the first t_atom pointer in an array to parse.
<i>fmt</i>	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 <i>fmt</i> string.

Returns

A Max error code.

See also

[atom_setformat\(\)](#)

36.17.3.16 t_atom_long atom_getlong (const t_atom * *a*)

Retrieves a long integer value from a [t_atom](#).

Parameters

<i>a</i>	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.

Referenced by [jit_atom_getlong\(\)](#).

36.17.3.17 t_max_err atom_getlong_array (long *ac*, t_atom * *av*, long *count*, t_atom_long * *vals*)

Fetch an array of long integer values from an array of atoms.

Parameters

<i>ac</i>	The number of atoms allocated in the <i>av</i> parameter.
<i>av</i>	The address to the first of an array of allocated atoms.
<i>count</i>	The number of values to fetch from the array specified by <i>vals</i> .
<i>vals</i>	The address of the array to which is copied the values from <i>av</i> .

Returns

A Max error code.

36.17.3.18 void* atom_getobj (const t_atom * *a*)

Retrieves a generic pointer value from a [t_atom](#).

Parameters

<i>a</i>	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.

Referenced by `jit_atom_getobj()`.

36.17.3.19 [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

<i>ac</i>	The number of atoms allocated in the <i>av</i> parameter.
<i>av</i>	The address to the first of an array of allocated atoms.
<i>count</i>	The number of values to fetch from the array specified by <i>vals</i> .
<i>vals</i>	The address of the array to which is copied the values from <i>av</i> .

Returns

A Max error code.

36.17.3.20 [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

<i>av</i>	A pointer to the atom from which to get the t_object .
<i>cls</i>	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.

36.17.3.21 [t_symbol* atom_getsym \(const t_atom * a \)](#)

Retrieves a [t_symbol](#) * value from a [t_atom](#).

Parameters

<i>a</i>	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.

Referenced by `jit_atom_getsym()`.

36.17.3.22 `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

<i>ac</i>	The number of atoms allocated in the <i>av</i> parameter.
<i>av</i>	The address to the first of an array of allocated atoms.
<i>count</i>	The number of values to fetch from the array specified by <i>vals</i> .
<i>vals</i>	The address of the array to which is copied the values from <i>av</i> .

Returns

A Max error code.

36.17.3.23 *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

<i>ac</i>	The number of atoms to fetch in <i>av</i> .
<i>av</i>	The address of the first <i>t_atom</i> pointer in an array to retrieve.
<i>textSize</i>	The size of the string to which the atoms will be formatted and copied.
<i>text</i>	The address of the string to which the text will be written.
<i>flags</i>	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\(\)](#)

36.17.3.24 long atom_gettype(const t_atom * a)

Retrieves type from a *t_atom*.

Parameters

<i>a</i>	Pointer to a <i>t_atom</i> whose type is of interest
----------	--

Returns

This function returns the type of the specified *t_atom* as defined in [e_max_atomtypes](#)

36.17.3.25 *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

<i>ac</i>	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 <i>av</i> .
-----------	--

<i>av</i>	The address to the first of an array of allocated atoms.
<i>count</i>	The number of values in the array specified by <i>vals</i> .
<i>vals</i>	The array from which to copy the values into the array of atoms at <i>av</i> .

Returns

A Max error code.

36.17.3.26 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

<i>ac</i>	The number of atoms to try to fetch from the array of chars. You should have at least this number of atoms allocated in <i>av</i> .
<i>av</i>	The address to the first of an array of allocated atoms.
<i>count</i>	The number of values in the array specified by <i>vals</i> .
<i>vals</i>	The array from which to copy the values into the array of atoms at <i>av</i> .

Returns

A Max error code.

36.17.3.27 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

<i>ac</i>	The number of atoms to try to fetch from the array of doubles. You should have at least this number of atoms allocated in <i>av</i> .
<i>av</i>	The address to the first of an array of allocated atoms.
<i>count</i>	The number of values in the array specified by <i>vals</i> .
<i>vals</i>	The array from which to copy the values into the array of atoms at <i>av</i> .

Returns

A Max error code.

36.17.3.28 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

<i>a</i>	Pointer to a t_atom whose value and type will be modified
<i>b</i>	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.

Referenced by [jit_atom_setfloat\(\)](#).

36.17.3.29 `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

<i>ac</i>	The number of atoms to try to fetch from the array of floats. You should have at least this number of atoms allocated in <i>av</i> .
<i>av</i>	The address to the first of an array of allocated atoms.
<i>count</i>	The number of values in the array specified by <i>vals</i> .
<i>vals</i>	The array from which to copy the values into the array of atoms at <i>av</i> .

Returns

A Max error code.

36.17.3.30 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

<i>ac</i>	The address of a variable to hold the number of returned atoms.
<i>av</i>	The address of a t_atom pointer to which memory may be allocated and atoms copied.
<i>fmt</i>	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\(\)](#)

36.17.3.31 t_max_err atom_setlong(t_atom * a, t_atom_long b)

Inserts an integer into a [t_atom](#) and change the [t_atom](#)'s type to [A_LONG](#).

Parameters

<i>a</i>	Pointer to a t_atom whose value and type will be modified
<i>b</i>	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.

Referenced by [jit_atom_setlong\(\)](#).

36.17.3.32 t_max_err atom_setlong_array(long ac, t_atom * av, long count, t_atom_long * vals)

Assign an array of long integer values to an array of atoms.

Parameters

<i>ac</i>	The number of atoms to try to fetch from the array of longs. You should have at least this number of atoms allocated in <i>av</i> .
<i>av</i>	The address to the first of an array of allocated atoms.
<i>count</i>	The number of values in the array specified by <i>vals</i> .
<i>vals</i>	The array from which to copy the values into the array of atoms at <i>av</i> .

Returns

A Max error code.

36.17.3.33 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

<i>a</i>	Pointer to a t_atom whose value and type will be modified
<i>b</i>	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.

Referenced by [jit_atom_setobj\(\)](#).

36.17.3.34 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

<i>ac</i>	The number of atoms to try to fetch from the array of objects. You should have at least this number of atoms allocated in <i>av</i> .
<i>av</i>	The address to the first of an array of allocated atoms.
<i>count</i>	The number of values in the array specified by <i>vals</i> .
<i>vals</i>	The array from which to copy the values into the array of atoms at <i>av</i> .

Returns

A Max error code.

36.17.3.35 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

<i>ac</i>	The address of a variable to hold the number of returned atoms.
<i>av</i>	The address of a t_atom pointer to which memory may be allocated and atoms copied.
<i>parsestr</i>	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.

```

1 t_atom *av = NULL;
2 long ac = 0;
3 t_max_err err = MAX_ERR_NONE;
4
5 err = atom_setparse(&ac, &av, "foo bar 1 2 3.0");

```

36.17.3.36 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

<i>a</i>	Pointer to a t_atom whose value and type will be modified
<i>b</i>	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.

Referenced by [jit_atom_setsym\(\)](#).

36.17.3.37 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

<i>ac</i>	The number of atoms to try to fetch from the array of symbols. You should have at least this number of atoms allocated in <i>av</i> .
<i>av</i>	The address to the first of an array of allocated atoms.
<i>count</i>	The number of values in the array specified by <i>vals</i> .
<i>vals</i>	The array from which to copy the values into the array of atoms at <i>av</i> .

Returns

A Max error code.

36.17.3.38 long atomisatomarray(t_atom *a)

Determines whether or not an atom represents a [t_atomarray](#) object.

Parameters

<i>a</i>	The address of the atom to test.
----------	----------------------------------

Returns

Returns true if the `t_atom` contains a valid `t_atomarray` object.

36.17.3.39 long atomisdictionary (`t_atom * a`)

Determines whether or not an atom represents a `t_dictionary` object.

Parameters

<i>a</i>	The address of the atom to test.
----------	----------------------------------

Returns

Returns true if the `t_atom` contains a valid `t_dictionary` object.

36.17.3.40 long atomisstring (`const t_atom * a`)

Determines whether or not an atom represents a `t_string` object.

Parameters

<i>a</i>	The address of the atom to test.
----------	----------------------------------

Returns

Returns true if the `t_atom` contains a valid `t_string` object.

36.17.3.41 void postargs (`long argc, t_atom * argv`)

Print the contents of an array of atoms to the Max window.

Parameters

<i>argc</i>	The count of atoms in argv.
<i>argv</i>	The address to the first of an array of atoms.

36.18 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

- **BEGIN_USING_C_LINKAGE 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.

36.18.1 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).

36.18.2 Function Documentation

36.18.2.1 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.
---	------------------------

36.18.2.2 BEGIN_USING_C_LINKAGE void* atombuf_new (long argc, t_atom * argv)

Use [atombuf_new\(\)](#) to create a new Atombuf from an array of t_atoms.

Parameters

<i>argc</i>	Number of t_atoms in the argv array. May be 0.
<i>argv</i>	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.

36.18.2.3 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)`.

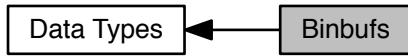
Parameters

<i>x</i>	Pointer to existing atombuf variable. The variable will be replaced by a new Atombuf containing the converted text.
<i>text</i>	Handle to the text to be converted. It need not be zero-terminated.
<i>size</i>	Number of characters in the text.

36.19 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 `t_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, t_ptr_size *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 `t_atom` from a text buffer.

36.19.1 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.

36.19.2 Function Documentation

36.19.2.1 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

<i>x</i>	Binbuf to receive the items.
<i>s</i>	Ignored. Pass NULL.
<i>argc</i>	Count of items in the argv array.
<i>argv</i>	Array of atoms to add to the Binbuf.

36.19.2.2 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

<i>x</i>	Binbuf containing the message.
<i>ac</i>	Count of items in the argv array.
<i>av</i>	Array of t_atoms as the arguments to the message.
<i>to</i>	Receiver of the message.

Returns

The result of sending the message.

36.19.2.3 short binbuf_getatom (void * *x*, long * *p1*, long * *p2*, t_atom * *ap*)

Use binbuf_getatom to retrieve a single [t_atom](#) from a Binbuf.

Parameters

<i>x</i>	Binbuf containing the desired t_atom .
<i>p1</i>	Offset into the Binbuf's array of types. Modified to point to the next t_atom .
<i>p2</i>	Offset into the Binbuf's array of data. Modified to point to the next t_atom .
<i>ap</i>	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:

```

1 t_atom holder;
2 long to, so;
3
4 to = 0;
5 so = 0;
6 while (!binbuf_getatom(x, &to, &so, &holder)) {
7     // do something with the t_atom
8 }
```

36.19.2.4 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

<i>x</i>	Binbuf to receive the items.
<i>s</i>	Ignored. Pass NULL.
<i>argc</i>	Count of items in the argv array.
<i>argv</i>	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.

36.19.2.5 void* binbuf_new (void)

Use [binbuf_new\(\)](#) to create and initialize a Binbuf.

Returns

Returns a new binbuf if successful, otherwise NULL.

36.19.2.6 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

<i>x</i>	Binbuf to receive the items.
<i>s</i>	Ignored. Pass NULL.
<i>argc</i>	Count of items in the argv array.
<i>argv</i>	Array of t_atoms to put in the Binbuf.

36.19.2.7 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

<i>x</i>	Binbuf to contain the converted text. It must have already been created with binbuf_new. Its previous contents are destroyed.
<i>srcText</i>	Handle to the text to be converted. It need not be terminated with a 0.
<i>n</i>	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.

36.19.2.8 `short binbuf_totext(void * x, char ** dstText, t_ptr_size * 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

<code>x</code>	Binbuf with data to convert to text.
<code>dstText</code>	Pre-existing handle where the text will be placed. <code>dstText</code> will be resized to accomodate the text.
<code>sizep</code>	Where <code>binbuf_totext()</code> 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.

36.19.2.9 `void binbuf_vinsert(void * x, char * fmt, ...)`

Use `binbuf_vinsert()` to append a Max message to a Binbuf adding a semicolon.

Parameters

<code>x</code>	Binbuf containing the desired <code>t_atom</code> .
<code>fmt</code>	A C-string containing one or more letters corresponding to the types of each element of the message. s for <code>t_symbol*</code> , l for long, or f for float.
<code>...</code>	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.

```

1 void myobject_save (myObject **x, Binbuf *dstBuf)
2 {
3     binbuf_vinsert(dstBuf, "ss1", gensym("#N"),
4                     gensym("myobject"),
5                     x->m_val1, x->m_val2);
6 }

```

Suppose that such an object had written this data into a file. If you opened the file as text, you would see the following:

```

1 #N myobject 10 20;
2 #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.

36.19.2.10 short readatom (char * *outstr*, char ** *text*, long * *n*, long *e*, t_atom * *ap*)

Use [readatom\(\)](#) to read a single [t_atom](#) from a text buffer.

Parameters

<i>outstr</i>	C-string of 256 characters that will receive the next text item read from the buffer.
<i>text</i>	Handle to the text buffer to be read.
<i>n</i>	Starts at 0, and is modified by readatom to point to the next item in the text buffer.
<i>e</i>	Number of characters in text.
<i>ap</i>	Where the resulting t_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.

```

1 long index = 0;
2 t_atom dst;
3 char outstr[256];
4
5 while (readatom(outstr, textHandle, &index, textLength, &dst))
6 {
7     // do something with the resulting t_atom
8 }

```

An alternative to using readatom is to turn your text into a Binbuf using [binbuf_text\(\)](#), then call [binbuf_getatom\(\)](#) in a loop.

36.20 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.

- [t_symbol * gensym_tr \(const char *s\)](#)

Given a C-string, fetch the matching [t_symbol](#) pointer from the symbol table, generating and translating the symbol if necessary.

36.20.1 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;
```

36.20.2 Function Documentation

36.20.2.1 `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 neccessary.

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.

Referenced by jit_attr_offset_array_new(), jit_attr_offset_new(), jit_attribute_new(), jit_bin_read_matrix(), jit_bin←
write_matrix(), jit_class_addinterface(), jit_class_method_addargsafe(), jit_class_method_argsafe_get(), jit_mop←
_input_nolink(), jit_mop_io_new(), jit_mop_new(), jit_mop_newcopy(), jit_mop_output_nolink(), jit_mop_single←
_planeCount(), jit_mop_single_type(), jit_ob3d_free(), jit_ob3d_new(), jit_ob3d_setup(), jit_object_exportattrs(),
jit_object_importattrs(), max_jit_attr_args(), max_jit_attr_get(), max_jit_attr_getdump(), max_jit_mop_variable←
addinputs(), max_jit_mop_variable_addoutputs(), max_jit_obex_gimmeback_dumpout(), and max_jit_obex_jitob←
_set().

36.20.2.2 t_symbol* gensym_tr (const char * s)

Given a C-string, fetch the matching [t_symbol](#) pointer from the symbol table, generating and translating the symbol if necessary.

Parameters

s	A C-string to be looked up in Max's symbol table and then translated
---	--

Returns

A pointer to the [t_symbol](#) in the symbol table.

36.21 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.

Macros

- #define [MAX_PATH_CHARS](#)
The size you should use when allocating strings for full paths.
- #define [MAX_FILENAME_CHARS](#)
The size you should use when allocating strings for filenames.

Typedefs

- typedef [t_filestruct](#) * [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, 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, PATH_TYPE_ABSOLUTE, PATH_TYPE_RELATIVE, PATH_TYPE_BOOT,
 PATH_TYPE_C74, PATH_TYPE_PATH, PATH_TYPE_DESKTOP, PATH_TYPE_TILDE,
 PATH_TYPE_TEMPFOLDER, PATH_TYPE_MAXDB }
Constants that determine the output of [path_nameconform\(\)](#).
- enum [e_max_fileinfo_flags](#) { PATH_FILEINFO_ALIAS, PATH_FILEINFO_FOLDER, PATH_FILEINFO_PAKAGE }
Flags used to represent properties of a file in a [t_fileinfo](#) struct.
- enum [e_max_path_folder_flags](#) { PATH_REPORTPACKAGEASFOLDER, PATH_FOLDER_SNIFF, PATH_NOALIASRESOLUTION }
Flags used by functions such as [path_foldernextfile\(\)](#) and [path_openfolder\(\)](#).
- enum [e_max_openfile_permissions](#) { PATH_READ_PERM, PATH_WRITE_PERM, PATH_RW_PERM }
Permissions or mode with which to open a file.
- enum [t_sysfile_pos_mode](#) { SYSFILE_ATMARK, SYSFILE_FROMSTART, SYSFILE_FROMEOF, SYSFILE_FROMMARK }
Modes used by [sysfile_setpos\(\)](#)
- enum [t_sysfile_text_flags](#) {
 TEXT_LB_NATIVE, TEXT_LB_MAC, TEXT_LB_PC, TEXT_LB_UNIX ,
 TEXT_ENCODING_USE_FILE, TEXT_NULL_TERMINATE }
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 `locate filetype` (C74_CONST char *name, short *outvol, `t_fourcc` filetype, `t_fourcc` creator)

Find a Max document by name in the search path.
- short `locatefile_extended` (char *name, short *outvol, `t_fourcc` *outtype, C74_CONST `t_fourcc` *filenamelist, short numtypes)

Find a file by name.
- 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, `t_fileinfo` *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, `t_ptr_uint` *date)

Determine the modification date of the selected path.
- short `path_getfilemoddate` (C74_CONST char *filename, C74_CONST short path, `t_ptr_uint` *date)

Determine the modification date of the selected file.
- void * `path_openfolder` (short path)

Prepare a directory for iteration.
- short `path_foldernextfile` (void **xx, `t_fourcc` *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 `path_createsysfile` (C74_CONST char *name, short path, `t_fourcc` 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.
- `t_max_err path_toabsolutesystempath` (const short in_path, const char *in_filename, char *out_filepath)

Translates a Max path+filename combo into a correct POSIX absolute path that can be used to pass to libraries and also handles multiple volumes correctly.
- `t_max_err path_absolutepath` (`t_symbol` **returned_path, const `t_symbol` *s, const `t_fourcc` *filenamelist, short numtypes)

Convert a path to an absolutepath as done by the absolutepath object in Max.
- 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.
 - `void fileusage_addpackage (void *w, C74_CONST char *name, t_object *subfoldernames)`
Add a package to a standalone.
 - `short open_dialog (char *name, short *volptr, t_fourcc *typeptr, t_fourcc *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, t_fourcc *type, t_fourcc *typelist, short numtypes)`
Present the user with the standard save file dialog with your own list of file types.
 - `BEGIN_USING_C_LINKAGE t_max_err sysfile_close (t_filehandle f)`
Close a file opened with `sysfile_open()`.
 - `t_max_err sysfile_read (t_filehandle f, t_ptr_size *count, void *bufptr)`
Read a file from disk.
 - `t_max_err sysfile_readtohandle (t_filehandle f, char ***h)`
Read the contents of a file into a handle.
 - `t_max_err sysfile_readtoptr (t_filehandle f, char **p)`
Read the contents of a file into a pointer.
 - `t_max_err sysfile_write (t_filehandle f, t_ptr_size *count, const void *bufptr)`
Write part of a file to disk.
 - `t_max_err sysfile_seteof (t_filehandle f, t_ptr_size logeof)`
Set the size of a file handle.
 - `t_max_err sysfile_geteof (t_filehandle f, t_ptr_size *logeof)`
Get the size of a file handle.
 - `t_max_err sysfile_setpos (t_filehandle f, t_sysfile_pos_mode mode, t_ptr_int offset)`
Set the current file position of a file handle.
 - `t_max_err sysfile_getpos (t_filehandle f, t_ptr_size *filepos)`
Get the current file position of a file handle.
 - `t_max_err sysfile_spoolcopy (t_filehandle src, t_filehandle dst, t_ptr_size size)`
Copy the contents of one file handle to another file handle.
 - `t_max_err sysfile_readtextfile (t_filehandle f, t_handle htext, t_ptr_size maxlen, t_sysfile_text_flags flags)`
Read a text file from disk.
 - `t_max_err sysfile_writetextfile (t_filehandle f, t_handle htext, t_sysfile_text_flags flags)`
Write a text file to disk.
 - `t_max_err sysfile_openhandle (char **h, t_sysfile_flags flags, t_filehandle *fh)`
Create a `t_filehandle` from a pre-existing handle.
 - `t_max_err sysfile_openptrsize (char *p, t_ptr_size length, t_sysfile_flags flags, t_filehandle *fh)`
Create a `t_filehandle` from a pre-existing pointer.

36.21.1 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.

36.21.2 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.

36.21.3 Example: filebyte (notes from the IRCAM workshop)

36.21.3.1 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

36.21.3.2 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

36.21.3.3 File Names

- C string
- Max 5 filenames are UTF-8
- Max 5 supports long (unicode) filenames on both Mac and Windows

36.21.3.4 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`

36.21.4 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.

36.21.5 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.

36.21.6 Macro Definition Documentation

36.21.6.1 #define MAX_FILENAME_CHARS

The size you should use when allocating strings for filenames.

At the time of this writing it supports up to 256 UTF chars

36.21.7 Typedef Documentation

36.21.7.1 typedef t_filestruct* 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`.

36.21.8 Enumeration Type Documentation

36.21.8.1 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)

36.21.8.2 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.

36.21.8.3 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

PATH_NOALIASRESOLUTION no alias resolution

36.21.8.4 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

36.21.8.5 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
PATH_TYPE_DESKTOP desktop
PATH_TYPE_TILDE "home"
PATH_TYPE_TEMPFOLDER /tmp
PATH_TYPE_MAXDB combi: try PATH_TYPE_C74, PATH_TYPE_TILDE, PATH_TYPE_RELATIVE, PATH_TYPE_ABSOLUTE in that order

36.21.8.6 enum t_sysfile_pos_mode

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_FROMMARK Calculate the file position from the current file position.

36.21.8.7 enum t_sysfile_text_flags

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.

36.21.9 Function Documentation

36.21.9.1 void fileusage_addfile (void * w, long flags, C74_CONST char * name, C74_CONST short path)

Add a file to a collective.

Parameters

<i>w</i>	Handle for the collective builder.
<i>flags</i>	If flags == 1, copy this file to support folder of an app instead of to the collective in an app.
<i>name</i>	The name of the file.
<i>path</i>	The path of the file to add.

36.21.9.2 void fileusage_addpackage (void * *w*, C74_CONST char * *name*, t_object * *subfoldernames*)

Add a package to a standalone.

Parameters

<i>w</i>	Handle for the standalone builder
<i>name</i>	The name of the package
<i>subfoldernames</i>	A t_atomarray containing symbols, each of which is a foldername in the package to include. Pass NULL to include the entire package contents.

Version

Introduced in Max 7.0.4

36.21.9.3 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

<i>owner</i>	Your object. This object will receive the message "filechanged" when the watcher sees a change in the file or folder.
<i>path</i>	The path in which the file being watched resides, or the path of the folder being watched.
<i>filename</i>	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:

```
1 void myObject_filechanged(t_myObject **x, char *filename, short path);
```

36.21.9.4 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

<i>name</i>	A C string that is the name of the file to look for.
<i>outvol</i>	The Path ID containing the location of the file if it is found.
<i>binflag</i>	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\(\)](#)

36.21.9.5 short locatefile_extended (char * *name*, short * *outvol*, t_fourcc * *outtype*, C74_CONST t_fourcc * *filetypelist*, short *numtypes*)

Find a file by name.

If a complete path is not specified, search for the name in the search path. This is the preferred method for file searching since its introduction in Max version 4.

Version

4.0

Parameters

<i>name</i>	The file name for the search, receives actual filename upon return.
<i>outvol</i>	The Path ID of the file (if found).
<i>outtype</i>	The file type of the file (if found).
<i>filetypelist</i>	The file type(s) for which you are searching for, or NULL if any type is acceptable.
<i>numtypes</i>	The number of file types in the typelist array (1 if a single entry, 0 if any type is acceptable).

Returns

If a file is found with the name specified by filename, locatefile returns 0, otherwise it returns non-zero.

See also

[path_getdefault\(\)](#).

Remarks

The old file search routines `locatefile()` and `locate filetype()` 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 `locate filetype()` that may require some rewriting of your code. *It modifies its name parameter* in certain cases, while `locatefile()` and `locate filetype()` 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:

```
1 char filename[MAX_FILENAME_CHARS];
2 strncpy_zero(filename,str->s_name, MAX_FILENAME_CHARS);
3 result = locatefile_extended(filename,&path,&type,typelist,1);
```

36.21.9.6 short locatefiletype (C74_CONST char * *name*, short * *outvol*, t_fourcc *filetype*, t_fourcc *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

<i>name</i>	A C string that is the name of the file to look for.
<i>outvol</i>	The Path ID containing the location of the file if it is found.
<i>filetype</i>	The filetype of the file to look for. If you pass 0L, files of all filetypes are considered.
<i>creator</i>	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\(\)](#)

36.21.9.7 short open_dialog (char * *name*, short * *volptr*, t_fourcc * *typeptr*, t_fourcc * *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

<i>name</i>	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 <code>MAX_FILENAME_CHARS</code> .
<i>volptr</i>	Receives the Path ID of the file the user wants to open.
<i>typeptr</i>	The file type of the file the user wants to open.
<i>types</i>	A list of file types to display. This is not limited to 4 types as in the <code>SFGetFile()</code> trap. Pass NULL to display all types.
<i>ntypes</i>	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\(\)](#)

36.21.9.8 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\(\)](#)

36.21.9.9 t_max_err path_absolutepath (t_symbol ** returned_path, const t_symbol * s, const t_fourcc * filetypeplist, short numtypes)

Convert a path to an absolutepath as done by the absolutepath object in Max.

Parameters

<i>returned_path</i>	Address to a symbol pointer which will be filled-in upon successful return.
<i>s</i>	Incoming (non-absolute) path.
<i>filetypeplist</i>	The first of an array of file types to limit the search.
<i>numtypes</i>	The number of entries in the filetypeplist parameter.

Returns

A Max error code.

See also

[path_topathname\(\)](#)
[locatefile_extended\(\)](#)

36.21.9.10 void path_closefolder (void * x)

Complete a directory iteration.

Parameters

x	The "folder state" value originally returned by path_openfolder() .
---	---

36.21.9.11 short path_createsysfile (C74_CONST char * name, short path, t_fourcc type, t_filehandle * ref)

Create a file given a type code, a filename, and a Path ID.

Parameters

<i>name</i>	The name of the file to be opened.
<i>path</i>	The Path ID of the file to be opened.
<i>type</i>	The file type of the created file.
<i>ref</i>	A t_filehandle reference to the opened file will be returned in this parameter.

Returns

An error code.

36.21.9.12 short path_fileinfo (C74_CONST char * *name*, C74_CONST short *path*, t_fileinfo * *info*)

Retrive a [t_fileinfo](#) structure from a file/path combination.

Parameters

<i>name</i>	The file name to be queried.
<i>path</i>	The Path ID of the file.
<i>info</i>	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.

36.21.9.13 short path_foldernextfile (void * *xx*, t_fourcc * *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

<i>xx</i>	The "folder state" value returned by path_openfolder() .
<i>filetype</i>	Contains the file type of the file type on return.
<i>name</i>	Contains the file name of the next file on return.
<i>descend</i>	Unused.

Returns

Returns non-zero if successful, and zero when there are no more files.

See also

[e_max_path_folder_flags](#)

36.21.9.14 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

<i>name</i>	The extended file path to be converted.
<i>path</i>	Contains the Path ID on return.
<i>filename</i>	Contains the file name on return.

Returns

Returns 0 if successful, otherwise it returns an OS-specific error code.

36.21.9.15 short path_getapppath (void)

Retrieve the Path ID of the Max application.

Returns

The path id.

36.21.9.16 short path_getdefault (void)

Retrieve the Path ID of the default search path.

Returns

The path id of the default search path.

36.21.9.17 short path_getfilemoddate (C74_CONST char * filename, C74_CONST short path, t_ptr_uint * date)

Determine the modification date of the selected file.

Parameters

<i>filename</i>	The name of the file to query.
<i>path</i>	The Path ID of the file.
<i>date</i>	The last modification date of the file upon return.

Returns

An error code.

36.21.9.18 short path_getmoddate (short path, t_ptr_uint * date)

Determine the modification date of the selected path.

Parameters

<i>path</i>	The Path ID of the directory to check.
<i>date</i>	The last modification date of the directory.

Returns

An error code.

36.21.9.19 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

<i>src</i>	A pointer to source character string to be converted.
<i>dst</i>	A pointer to destination character string.
<i>style</i>	The destination filepath style, as defined in e_max_path_styles
<i>type</i>	The destination filepath type, as defined in e_max_path_types

Returns

An error code.

See also

[MAX_PATH_CHARS](#)

36.21.9.20 void* path_openfolder (short *path*)

Prepare a directory for iteration.

Parameters

<i>path</i>	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.

36.21.9.21 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

<i>name</i>	The name of the file to be opened.
<i>path</i>	The Path ID of the file to be opened.
<i>ref</i>	A t_filehandle reference to the opened file will be returned in this parameter.
<i>perm</i>	The permission for the opened file as defined in e_max_openfile_permissions .

Returns

An error code.

36.21.9.22 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

<i>name</i>	A file name (which may be fully or partially qualified), will contain the file name on return.
<i>path</i>	The Path ID to be resolved.
<i>outpath</i>	The Path ID of the returned file name.

Returns

Returns 0 if successful.

36.21.9.23 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

<i>path</i>	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).
<i>recursive</i>	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.

36.21.9.24 t_max_err path_toabsolutesystempath(const short *in_path*, const char * *in_filename*, char * *out_filepath*)

Translates a Max path+filename combo into a correct POSIX absolute path that can be used to pass to libraries and also handles multiple volumes correctly.

Parameters

<i>in_path</i>	The Max path reference
<i>in_filename</i>	The name of the file in that path.
<i>out_filepath</i>	A string that is MAX_PATH_CHARS in length, which will receive the formatted absolute path upon return.

Returns

Returns 0 if successful.

See also

[path_topotentialname\(\)](#)
[path_nameconform\(\)](#)

```
1 // example for getting a windows-formatted path for a folder path:
2 path_toabsolutesystempath(pathid, "", filestring);
3 path_nameconform(filestring, sNativeQualifiedPathname, PATH_STYLE_NATIVE, PATH_TYPE_PATH);
```

36.21.9.25 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

<i>path</i>	The path to be used.
<i>file</i>	The file name to be used.
<i>name</i>	Loaded with the fully extended file name on return.

Returns

Returns 0 if successful, otherwise it returns an OS-specific error code.

36.21.9.26 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

<i>path</i>	The path to be used.
<i>file</i>	The file name to be used.
<i>name</i>	Loaded with the fully extended file name on return.
<i>check</i>	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\(\)](#)

36.21.9.27 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

<i>filename</i>	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_FILE_NAME_CHARS .
<i>path</i>	If the user decides to save the file, the Path ID of the location chosen is returned here.
<i>binptr</i>	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\(\)](#)

36.21.9.28 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\(\)](#)

36.21.9.29 short saveasdialog_extended(char * name, short * vol, t_fourcc * type, t_fourcc * 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

<i>name</i>	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_FILE_NAME_CHARS .
<i>vol</i>	If the user decides to save the file, the Path ID of the location chosen is returned here.
<i>type</i>	Returns the type of file chosen by the user.
<i>typelist</i>	The list of types provided to the user.
<i>numtypes</i>	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\(\)](#)

36.21.9.30 BEGIN_USING_C_LINKAGE t_max_err 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

<i>f</i>	The t_filehandle structure of the file the user wants to close.
----------	---

Returns

An error code.

36.21.9.31 t_max_err sysfile_geteof(t_filehandle f, t_ptr_size * 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

<i>f</i>	The file's t_filehandle structure.
<i>logeof</i>	The file size in bytes is returned to this value.

Returns

An error code.

36.21.9.32 t_max_err sysfile_getpos(t_filehandle f, t_ptr_size * 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

<i>f</i>	The file's t_filehandle structure.
<i>filepos</i>	The address of a variable to hold the current file position of file handle in bytes.

Returns

An error code.

36.21.9.33 t_max_err sysfile_openhandle (char ** *h*, t_sysfile_flags *flags*, t_filehandle * *fh*)

Create a [t_filehandle](#) from a pre-existing handle.

Parameters

<i>h</i>	A handle for some data, data is <i>*not*</i> copied and <i>*not*</i> freed on file close.
<i>flags</i>	Pass 0 (additional flags are private).
<i>fh</i>	The address of a t_filehandle which will be allocated.

Returns

An error code.

36.21.9.34 t_max_err sysfile_openptrsize (char * *p*, t_ptr_size *length*, t_sysfile_flags *flags*, t_filehandle * *fh*)

Create a [t_filehandle](#) from a pre-existing pointer.

Parameters

<i>p</i>	A pointer to some data. Data is <i>*not*</i> copied and <i>*not*</i> freed on file close.
<i>length</i>	The size of <i>p</i> .
<i>flags</i>	Pass 0 (additional flags are private).
<i>fh</i>	The address of a t_filehandle which will be allocated.

Returns

An error code.

36.21.9.35 t_max_err sysfile_read (t_filehandle *f*, t_ptr_size * *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

<i>f</i>	The t_filehandle structure of the file the user wants to open.
<i>count</i>	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.

<i>bufptr</i>	Pointer to the buffer that the data will be read into.
---------------	--

Returns

An error code.

Referenced by `jit_bin_read_chunk_info()`, `jit_bin_read_header()`, and `jit_bin_read_matrix()`.

36.21.9.36 *t_max_err sysfile_readtextfile(t_filehandle f, t_handle htext, t_ptr_size maxlen, t_sysfile_text_flags 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

<i>f</i>	The t_filehandle structure of the text file the user wants to open.
<i>htext</i>	Handle that the data will be read into.
<i>maxlen</i>	The maximum length in bytes to be read into the handle. Passing the value 0L indicates no maximum (i.e. read the entire file).
<i>flags</i>	Flags to set linebreak translation as defined in t_sysfile_text_flags .

Returns

An error code.

36.21.9.37 *t_max_err sysfile_readtohandle(t_filehandle f, char * h)***

Read the contents of a file into a handle.

Parameters

<i>f</i>	The open t_filehandle structure to read into the handle.
<i>h</i>	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\(\)](#).

36.21.9.38 *t_max_err sysfile_readtoptr(t_filehandle f, char ** p)*

Read the contents of a file into a pointer.

Parameters

<i>f</i>	The open t_filehandle structure to read into the handle.
----------	--

<i>p</i>	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_freeptr\(\)](#).

36.21.9.39 t_max_err sysfile_seteof(t_filehandle *f*, t_ptr_size *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

<i>f</i>	The file's t_filehandle structure.
<i>logeof</i>	The file size in bytes.

Returns

An error code.

36.21.9.40 t_max_err sysfile_setpos(t_filehandle *f*, t_sysfile_pos_mode *mode*, t_ptr_int *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 [t_sysfile_pos_mode](#).

Parameters

<i>f</i>	The file's t_filehandle structure.
<i>mode</i>	Mode from which the offset will be calculated, as defined in t_sysfile_pos_mode .
<i>offset</i>	The offset in bytes relative to the mode.

Returns

An error code.

Referenced by [jit_bin_read_chunk_info\(\)](#), [jit_bin_read_header\(\)](#), and [jit_bin_write_header\(\)](#).

36.21.9.41 t_max_err sysfile_spoolcopy(t_filehandle *src*, t_filehandle *dst*, t_ptr_size *size*)

Copy the contents of one file handle to another file handle.

Parameters

<i>src</i>	The file handle from which to copy.
<i>dst</i>	The file handle to which the copy is written.

<code>size</code>	The number of bytes to copy. If 0 the size of <code>src</code> will be used.
-------------------	--

Returns

An error code.

36.21.9.42 `t_max_err sysfile_write (t_filehandle f, t_ptr_size * 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

<code>f</code>	The <code>t_filehandle</code> structure of the file to which the user wants to write.
<code>count</code>	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.
<code>bufptr</code>	Pointer to the buffer that the data will be read from.

Returns

An error code.

Referenced by `jit_bin_write_header()`, and `jit_bin_write_matrix()`.

36.21.9.43 `t_max_err sysfile_writetextfile (t_filehandle f, t_handle htext, t_sysfile_text_flags 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

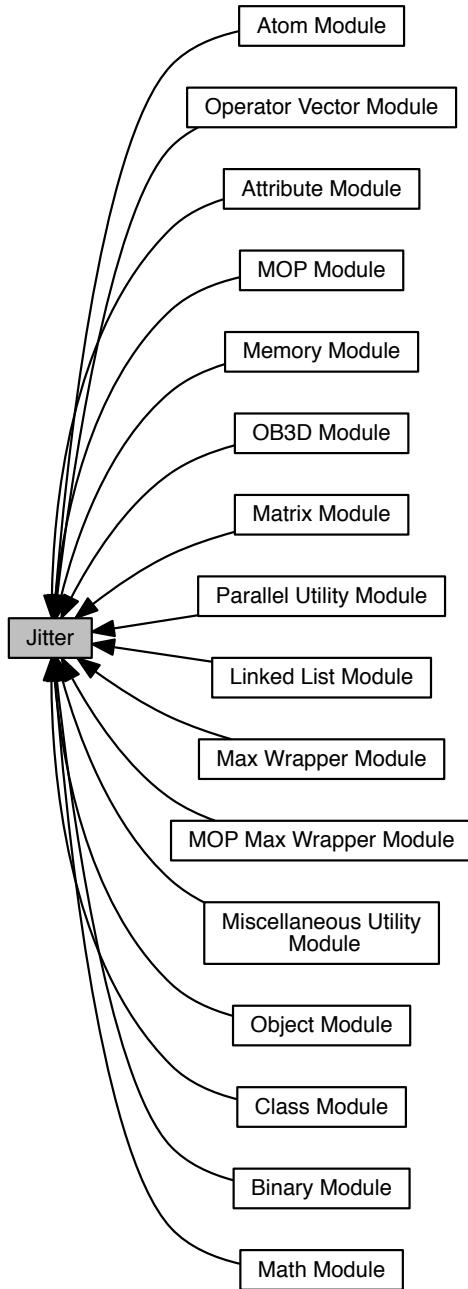
<code>f</code>	The <code>t_filehandle</code> structure of the text file to which the user wants to write.
<code>htext</code>	Handle that the data that will be read from.
<code>flags</code>	Flags to set linebreak translation as defined in <code>t_sysfile_text_flags</code> .

Returns

An error code.

36.22 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

Macros

- `#define JIT_ATTR_GET_OPAQUE`
private getter (all)
- `#define JIT_ATTR_SET_OPAQUE`
private setter (all)
- `#define JIT_ATTR_GET_OPAQUE_USER`
private getter (user)
- `#define JIT_ATTR_SET_OPAQUE_USER`
private setter (user)
- `#define JIT_ATTR_GET_DEFER`
defer getter (deprecated)
- `#define JIT_ATTR_GET_USURP`
usurp getter (deprecated)
- `#define JIT_ATTR_GET_DEFER_LOW`
defer getter
- `#define JIT_ATTR_GET_USURP_LOW`
usurp getter
- `#define JIT_ATTR_SET_DEFER`
defer setter (deprecated)
- `#define JIT_ATTR_SET_USURP`
usurp setter (deprecated)
- `#define JIT_ATTR_SET_DEFER_LOW`
defer setter
- `#define JIT_ATTR_SET_USURP_LOW`
usurp setter
- `#define JIT_MATRIX_DATA_HANDLE`
data is handle
- `#define JIT_MATRIX_DATA_REFERENCE`
data is reference to outside memory
- `#define JIT_MATRIX_DATA_PACK_TIGHT`
data is tightly packed (doesn't use standard 16 byte alignment)
- `#define JIT_MATRIX_DATA_FLAGS_USE`
necessary if using handle/reference data flags when creating jit_matrix, however, it is never stored in matrix
- `#define JIT_MATRIX_MAX_DIMCOUNT`
maximum dimension count
- `#define JIT_MATRIX_MAX_PLANECOUNT`

- `#define JIT_MATRIX_CONVERT_CLAMP`
maximum plane count
not currently used
- `#define JIT_MATRIX_CONVERT_INTERP`
use interpolation
- `#define JIT_MATRIX_CONVERT_SRCDIM`
use source dimensions
- `#define JIT_MATRIX_CONVERT_DSTDIM`
use destination dimensions
- `#define JIT_OB3D_NO_ROTATION_SCALE`
ob3d flag
- `#define JIT_OB3D_NO_POLY_VARS`
ob3d flag
- `#define JIT_OB3D_NO_BLEND`
ob3d flag
- `#define JIT_OB3D_NO_TEXTURE`
ob3d flag
- `#define JIT_OB3D_NO_MATRIXOUTPUT`
ob3d flag
- `#define JIT_OB3D_AUTO_ONLY`
ob3d flag
- `#define JIT_OB3D_DOES_UI`
ob3d flag
- `#define JIT_OB3D_NO_DEPTH`
ob3d flag
- `#define JIT_OB3D_NO_ANTIALIAS`
ob3d flag
- `#define JIT_OB3D_NO_FOG`
ob3d flag
- `#define JIT_OB3D_NO_LIGHTING_MATERIAL`
ob3d flag
- `#define JIT_OB3D_HAS_LIGHTS`
ob3d flag
- `#define JIT_OB3D_HAS_CAMERA`
ob3d flag
- `#define JIT_OB3D_IS_RENDERER`
ob3d flag
- `#define JIT_OB3D_NO_COLOR`
ob3d flag
- `#define JIT_OB3D_IS_SLAB`
ob3d flag
- `#define MAX_JIT_MOP_FLAGS_NONE`
mop flag
- `#define MAX_JIT_MOP_FLAGS_OWN_ALL`
mop flag
- `#define MAX_JIT_MOP_FLAGS_OWN_JIT_MATRIX`
mop flag
- `#define MAX_JIT_MOP_FLAGS_OWN_BANG`
mop flag
- `#define MAX_JIT_MOP_FLAGS_OWN_OUTPUTMATRIX`
mop flag

- #define MAX JIT MOP FLAGS OWN NAME
mop flag
- #define MAX JIT MOP FLAGS OWN TYPE
mop flag
- #define MAX JIT MOP FLAGS OWN DIM
mop flag
- #define MAX JIT MOP FLAGS OWN PLANECOUNT
mop flag
- #define MAX JIT MOP FLAGS OWN CLEAR
mop flag
- #define MAX JIT MOP FLAGS OWN NOTIFY
mop flag
- #define MAX JIT MOP FLAGS OWN ADAPT
mop flag
- #define MAX JIT MOP FLAGS OWN OUTPUTMODE
mop flag
- #define MAX JIT MOP FLAGS ONLY MATRIX PROBE
mop flag
- #define JIT MOP INPUT
mop flag
- #define JIT MOP OUTPUT
mop flag

Typedefs

- typedef t_object t jit object
object header

Variables

- 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
- JIT_EX_DATA `t_symbol` * `_jit_sym_position`
 cached t_symbol
- JIT_EX_DATA `t_symbol` * `_jit_sym_rotatexyz`
 cached t_symbol
- JIT_EX_DATA `t_symbol` * `_jit_sym_scale`
 cached t_symbol
- JIT_EX_DATA `t_symbol` * `_jit_sym_quat`
 cached t_symbol
- JIT_EX_DATA `t_symbol` * `_jit_sym_direction`
 cached t_symbol
- JIT_EX_DATA `t_symbol` * `_jit_sym_lookat`
 cached t_symbol
- JIT_EX_DATA `t_symbol` * `_jit_sym_anim`

- cached t_symbol*
 - JIT_EX_DATA *t_symbol* * *_jit_sym_bounds*
 - cached t_symbol*
 - JIT_EX_DATA *t_symbol* * *_jit_sym_boundcalc*
 - cached t_symbol*
 - JIT_EX_DATA *t_symbol* * *_jit_sym_calcbounds*
 - cached t_symbol*

36.22.1 Detailed Description

36.23 Memory Management

In the past, Max has provided two separate APIs for memory management.

Macros

- `#define MM_UNIFIED`

This macro being defined means that getbytes and sysmem APIs for memory management are unified.

Functions

- `char * getbytes (t_getbytes_size size)`

Allocate small amounts of non-relocatable memory.
- `void freebytes (void *b, t_getbytes_size size)`

Free memory allocated with getbytes().
- `char * getbytes16 (t_getbytes_size 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, t_getbytes_size 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 disphandle (char **h)`

Free the memory used by a handle you no longer need.
- `BEGIN_USING_C_LINKAGE 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.

36.23.1 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.

36.23.2 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).

36.23.3 Macro Definition Documentation

36.23.3.1 #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 older max targets.

36.23.4 Function Documentation

36.23.4.1 void disposhandle (char ** h)

Free the memory used by a handle you no longer need.

Parameters

<i>h</i>	The handle to dispose.
----------	------------------------

See also

[sysmem_freehandle\(\)](#)

36.23.4.2 void freebytes (void * *b*, t_getbytes_size *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

<i>b</i>	A pointer to the block of memory previously allocated that you want to free.
<i>size</i>	The size the block specified (as parameter <i>b</i>) in bytes.

Referenced by `jit_object_exportattrs()`, `jit_object_importattrs()`, `max_jit_attr_getdump()`, `max_jit_obex_gimmeback()`, and `max_jit_obex_gimmeback_dumpout()`.

36.23.4.3 void freebytes16 (char * *mem*, t_getbytes_size *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

<i>mem</i>	A pointer to the block of memory previously allocated that you want to free.
<i>size</i>	The size the block specified (as parameter <i>b</i>) 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\(\)](#).

36.23.4.4 char* getbytes (t_getbytes_size *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

<i>size</i>	The size to allocate in bytes (up to 32767 bytes).
-------------	--

Returns

A pointer to the allocated memory.

36.23.4.5 char* getbytes16 (t_getbytes_size *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

<i>size</i>	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\(\)](#).

36.23.4.6 short growhandle (void * *h*, long *size*)

Change the size of a handle.

Parameters

<i>h</i>	The handle to resize.
<i>size</i>	The new size to allocate in bytes.

Returns

Ignored.

See also

[sysmem_resizehandle\(\)](#)

36.23.4.7 char newhandle (long size)**

Allocate relocatable memory.

Parameters

<i>size</i>	The size to allocate in bytes.
-------------	--------------------------------

Returns

The allocated handle.

See also

[sysmem_newhandle\(\)](#)

36.23.4.8 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

<i>src</i>	A pointer to the memory whose bytes will be copied.
<i>dst</i>	A pointer to the memory where the data will be copied.
<i>bytes</i>	The size in bytes of the data to be copied.

Referenced by jit_copy_bytes().

36.23.4.9 void sysmem_freehandle (t_handle handle)

Free memory allocated with [sysmem_newhandle\(\)](#).

Parameters

<i>handle</i>	The handle whose memory will be freed.
---------------	--

Referenced by jit_handle_free().

36.23.4.10 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

<i>ptr</i>	The pointer whose memory will be freed.
------------	---

Referenced by jit_disposeptr(), jit_freebytes(), and jit_gl_begincapture().

36.23.4.11 long sysmem_handlesize (t_handle handle)

Find the size of a handle.

This function is similar to GetHandleSize().

Parameters

<i>handle</i>	The handle whose size will be queried.
---------------	--

Returns

The number of bytes allocated to the specified handle.

Referenced by jit_handle_size_get().

36.23.4.12 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

<i>handle</i>	The handle that will be locked.
<i>lock</i>	The new lock state of the handle.

Returns

The previous lock state.

Referenced by jit_handle_lock().

36.23.4.13 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

<i>size</i>	The size of the handle in bytes that will be allocated.
-------------	---

Returns

A new [t_handle](#).

Referenced by jit_handle_new().

36.23.4.14 t_handle sysmem_newhandleclear (unsigned long size)

Allocate a handle (a pointer to a pointer) whose memory is set to zero.

Parameters

<code>size</code>	The size of the handle in bytes that will be allocated.
-------------------	---

Returns

A new [t_handle](#).

See also

[sysmem_newhandle\(\)](#)

36.23.4.15 BEGIN_USING_C_LINKAGE 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

<code>size</code>	The amount of memory to allocate.
-------------------	-----------------------------------

Returns

A pointer to the allocated memory, or NULL if the allocation fails.

Referenced by [jit_getbytes\(\)](#), [jit_gl_begincapture\(\)](#), and [jit_newptr\(\)](#).

36.23.4.16 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

<code>size</code>	The amount of memory to allocate.
-------------------	-----------------------------------

Returns

A pointer to the allocated memory, or NULL if the allocation fails.

36.23.4.17 long sysmem_nullterminatehandle (t_handle h)

Add a null terminator to a handle.

Parameters

<code>h</code>	A handle to null terminate.
----------------	-----------------------------

Returns

An error code.

36.23.4.18 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

<i>p</i>	The existing pointer whose data will be copied into the resized handle.
<i>h</i>	The handle which will be enlarged by the size of the pointer.
<i>size</i>	The size in bytes that will be added to the handle.

Returns

The number of bytes allocated to the specified handle.

36.23.4.19 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

<i>p</i>	The existing pointer whose data will be copied into the resized handle.
<i>h</i>	The handle which will be enlarged by the size of the pointer.
<i>size</i>	The size in bytes that will be added to the handle.

Returns

An error code.

36.23.4.20 long sysmem_ptrsize (void * *ptr*)

Find the size of a pointer.

This function is similar to [_msize\(\)](#).

Parameters

<i>ptr</i>	The pointer whose size will be queried
------------	--

Returns

The number of bytes allocated to the pointer specified.

36.23.4.21 long sysmem_resizehandle (t_handle *handle*, long *newsize*)

Resize an existing handle.

This function is similar to [SetHandleSize\(\)](#). It resizes an existing handle to the size specified.

Parameters

<i>handle</i>	The handle that will be resized.
<i>newsize</i>	The new size of the handle in bytes.

Returns

The number of bytes allocated to the specified handle.

Referenced by [jit_handle_size_set\(\)](#).

36.23.4.22 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

<i>ptr</i>	The pointer to the memory that will be resized.
<i>newsize</i>	The new size of the pointer in bytes.

Returns

A pointer to the resized memory, or NULL if the allocation fails.

36.23.4.23 t_ptr sysmem_resizeptrclear(void * ptr, long newsize)

Resize an existing pointer and clear it.

Parameters

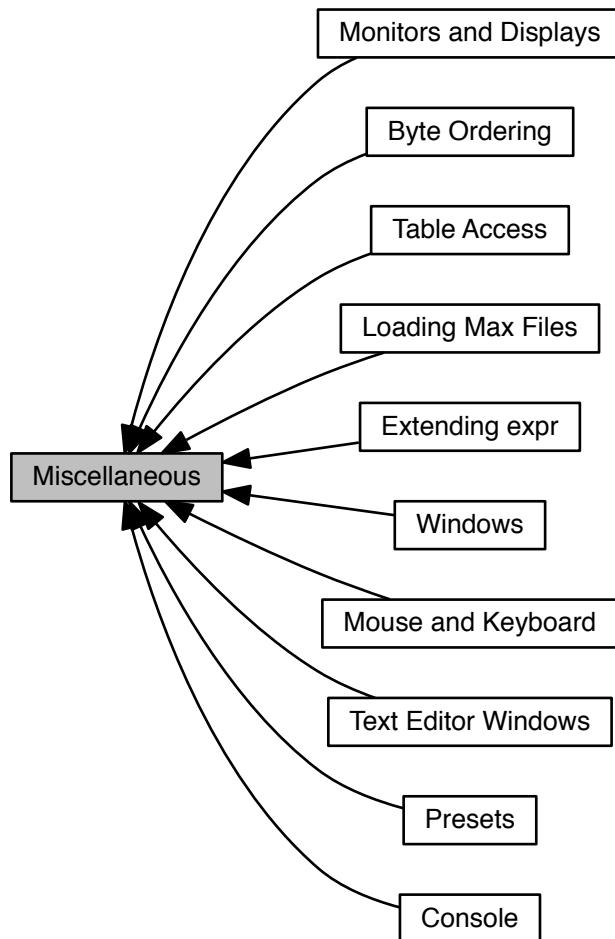
<i>ptr</i>	The pointer to the memory that will be resized.
<i>newsize</i>	The new size of the pointer in bytes.

Returns

A pointer to the resized memory, or NULL if the allocation fails.

36.24 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.

- **Loading Max Files**

Several high-level functions permit you to load patcher files.

- **Monitors and Displays**

Functions for finding out information about the environment.

- **Windows**

- **Mouse and Keyboard**

Macros

- **#define INRANGE(v, lo, hi)**

If a value is within the specified range, then return true.

- **#define MAX(a, b)**

Return the higher of two values.

- **#define MIN(a, b)**

Return the lower of two values.

- **#define CLAMP(a, lo, hi)**

Limit values to within a specified range.

- **#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`.

- **#define calcoffset(x, y)**

Find byte offset of a named member of a struct, relative to the beginning of that struct.

- **#define structmembersize(structname, membername)**

Find size of a named member of a struct.

Typedefs

- **typedef unsigned int t_uint**

an unsigned int as defined by the architecture / platform

- **typedef char t_int8**

a 1-byte int

- **typedef unsigned char t_uint8**

an unsigned 1-byte int

- **typedef short t_int16**

a 2-byte int

- **typedef unsigned short t_uint16**

an unsigned 2-byte int

- **typedef int t_int32**

a 4-byte int

- **typedef unsigned int t_uint32**

an unsigned 4-byte int

- **typedef long long t_int64**

an 8-byte int

- **typedef unsigned long long t_uint64**

an unsigned 8-byte int

- **typedef t_uint32 t_fourcc**

an integer of suitable size to hold a four char code / identifier

- **typedef unsigned long t_ptr_uint**

- `t_ptr_int`
an unsigned pointer-sized int
- `t_atom_float`
the type that is an A_FLOAT in a `t_atom`
- `t_getbytes_size`
like size_t but for `getbytes()`
- `t_ptr_int t_int`
an integer
- `t_ptr_uint t_ptr_size`
unsigned pointer-sized value for counting (like size_t)
- `t_ptr_int t_atom_long`
the type that is an A_LONG in a `t_atom`
- `t_atom_long t_max_err`
an integer value suitable to be returned as an error code
- `t_handle`
a handle (address of a pointer)
- `t_ptr`
a pointer
- `t_uint8 t_bool`
a true/false variable
- `t_int16 t_filepath`
i.e. path/vol in file APIs identifying a folder

Enumerations

- `e_max_errorcodes` {

MAX_ERR_NONE, MAX_ERR_GENERIC, MAX_ERR_INVALID_PTR, MAX_ERR_DUPLICATE,
MAX_ERR_OUT_OF_MEM }

Standard values returned by function calls with a return type of `t_max_err`.
- `e_max_wind_advise_result` { aaYes, aaNo, aaCancel }

Returned values from `wind_advise()`

Functions

- `BEGIN_USING_C_LINKAGE 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`.
- `t_atom_long method_true (void *x)`

A method that always returns true.
- `t_atom_long method_false (void *x)`

A method that always returns false.
- `t_symbol * symbol_unique (void)`

*Generates a unique `t_symbol` *.*
- `t_symbol * symbol_stripquotes (t_symbol *s)`

- Strip quotes from the beginning and end of a symbol if they are present.*
- 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.
 - [BEGIN_USING_C_LINKAGE](#) 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.
 - void [classname_openrefpage_ext](#) ([t_symbol](#) *classnamespace, char *classname)

Show the refpage for a given class.

36.24.1 Detailed Description

36.24.2 Macro Definition Documentation

36.24.2.1 #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.

36.24.2.2 #define calcoffset(x, y)

Find byte offset of a named member of a struct, relative to the beginning of that struct.

Parameters

<i>x</i>	The name of the struct
<i>y</i>	The name of the member

Returns

A pointer-sized integer representing the number of bytes into the struct where the member begins.

Referenced by ext_main(), and jit_ob3d_setup().

36.24.2.3 #define CLAMP(*a*, *lo*, *hi*)

Limit values to within a specified range.

Parameters

<i>a</i>	The value to constrain. NB: CLIP_ASSIGN modifies param 'a' but CLAMP only returns limited value
<i>lo</i>	The low bound for the range.
<i>hi</i>	The high bound for the range.

Returns

Returns the value *a* constrained to the range specified by *lo* and *hi*.

36.24.2.4 #define INRANGE(*v*, *lo*, *hi*)

If a value is within the specified range, then return true.

Otherwise return false.

Parameters

<i>v</i>	The value to test.
<i>lo</i>	The low bound for the range.
<i>hi</i>	The high bound for the range.

Returns

Returns true if within range, otherwise false.

36.24.2.5 #define MAX(*a*, *b*)

Return the higher of two values.

Parameters

<i>a</i>	The first value to compare.
<i>b</i>	The second value to compare.

Returns

Returns the higher of *a* or *b*.

Referenced by jit_gl_texcoord2f(), jit_matrix_frommatrix(), and jit_mop_io_restrict_dim().

36.24.2.6 #define MIN(*a*, *b*)

Return the lower of two values.

Parameters

<i>a</i>	The first value to compare.
<i>b</i>	The second value to compare.

Returns

Returns the lower of a or b.

Referenced by jit_gl_texcoord1f(), jit_gl_texcoord1fv(), jit_gl_texcoord2f(), jit_gl_texcoord2fv(), jit_gl_texcoord3f(), and jit_gl_texcoord3fv().

36.24.2.7 #define structmembersize(*structname*, *membername*)

Find size of a named member of a struct.

Parameters

<i>structname</i>	The name of the struct
<i>membername</i>	The name of the member

Returns

The size of the member of the struct.

36.24.3 Enumeration Type Documentation**36.24.3.1 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.

36.24.3.2 enum e_max_wind_advise_result

Returned values from [wind_advise\(\)](#)

Enumerator

aaYes Yes button was choosen.

aaNo No button was choosen.

aaCancel Cancel button was choosen.

36.24.4 Function Documentation**36.24.4.1 void classname_openrefpage_ext(*t_symbol* * *classnamespace*, char * *classname*)**

Show the refpage for a given class.

Parameters

<i>classnamespace</i>	The namespace for the class, e.g. "box".
<i>classname</i>	The name of the class.

36.24.4.2 void error_subscribe (t_object * x)

Receive messages from the error handler.

Parameters

<i>x</i>	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:

```
1 addmess( (method)myobject_error, "error", A_GIMME, 0 );
```

Your error handling routine should be declared as follows:

```
1 void myobject_error(t_myobject *x, t_symbol *s, short argc, t_atom *argv);
```

36.24.4.3 void error_sym (void * x, t_symbol * s)

Posts an error message to the Max window.

This function is interrupt safe.

Parameters

<i>x</i>	The object's pointer
<i>s</i>	Symbol to be posted as an error in the Max window

36.24.4.4 void error_unsubscribe (t_object * x)

Remove an object as an error message recipient.

Parameters

<i>x</i>	The object to unsubscribe.
----------	----------------------------

36.24.4.5 t_max_err globalsymbol_bind (t_object * x, C74_CONST char * name, long flags)

Bind an object to a [t_symbol](#).

Parameters

<i>x</i>	The object to bind to the t_symbol .
----------	--

<i>name</i>	The name of the t_symbol to which the object will be bound.
<i>flags</i>	Pass 0.

Returns

A Max error code.

36.24.4.6 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

<i>x</i>	The object that is getting the reference to the symbol.
<i>name</i>	The name of the symbol to reference.
<i>classname</i>	The name of the class of which the object we are referencing should be an instance.

See also

[globalsymbol_reference\(\)](#)

36.24.4.7 BEGIN_USING_C_LINKAGE 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

<i>x</i>	The object that is getting the reference to the symbol.
<i>name</i>	The name of the symbol to reference.
<i>classname</i>	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.

```

1 // the struct of our object
2 typedef struct _myobject {
3     t_object      obj;
4     t_symbol      *buffer_name;
5     t_buffer      *buffer_object;
6 } t_myobject;
7
8 void myobject_setbuffer(t_myobject *x, t_symbol *s, long argc, t_atom *argv)
9 {
10    if(s != x->buffer_name) {
11        // Reference the buffer associated with the incoming name
12        x->buffer_object = (t_buffer *)globalsymbol_reference((t_object *)x, s->s_name, "buffer~");
13
14        // If we were previously referencing another buffer, we should not longer reference it.
15        globalsymbol_dereference((t_object *)x, x->buffer_name->s_name, "buffer~");
16
17        x->buffer_name = s;
18    }
19 }
```

36.24.4.8 void globalsymbol_unbind (t_object * x, C74_CONST char * name, long flags)

Remove an object from being bound to a [t_symbol](#).

Parameters

<i>x</i>	The object from which to unbind the t_symbol .
<i>name</i>	The name of the t_symbol from which the object will be unbound.
<i>flags</i>	Pass 0.

36.24.4.9 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.

36.24.4.10 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.

Referenced by [ext_main\(\)](#).

36.24.4.11 void post_sym(void *x, t_symbol *s)

Posts a message to the Max window.

This function is interrupt safe.

Parameters

<i>x</i>	The object's pointer
<i>s</i>	Symbol to be posted in the Max window

36.24.4.12 void quittask_install(method m, void *a)

Register a function that will be called when Max exits.

Parameters

<i>m</i>	A function that will be called on Max exit.
<i>a</i>	Argument to be used with method <i>m</i> .

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.

36.24.4.13 void quittask_remove (method *m*)

Unregister a function previously registered with [quittask_install\(\)](#).

Parameters

<i>m</i>	Function to be removed as a shutdown method.
----------	--

36.24.4.14 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

<i>buffer</i>	The destination string (already allocated) for the copy.
<i>count</i>	The number of chars allocated to the buffer string.
<i>format</i>	The source string that will be copied, which may include sprintf() formatting codes for substitutions.
<i>...</i>	An array of arguments to be substituted into the format string.

36.24.4.15 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

<i>dst</i>	The destination string onto whose end the src string will be appended.
<i>src</i>	The source string that will be copied.
<i>size</i>	The number of chars allocated to the dst string.

36.24.4.16 BEGIN_USING_C_LINKAGE 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

<i>dst</i>	The destination string (already allocated) for the copy.
<i>src</i>	The source string that will be copied.
<i>size</i>	The number of chars allocated to the dst string.

36.24.4.17 t_symbol* symbol_stripquotes (*t_symbol * s*)

Strip quotes from the beginning and end of a symbol if they are present.

Parameters

<i>s</i>	The symbol to be stripped.
----------	----------------------------

Returns

Symbol with any leading/trailing quote pairs removed.

36.24.4.18 `t_symbol* symbol_unique(void)`

Generates a unique `t_symbol *`.

The symbol will be formatted somewhat like "u123456789".

Returns

This function returns a unique `t_symbol *`.

36.24.4.19 `t_max_err symbolarray_sort(long ac, t_symbol ** av)`

Performs an ASCII sort on an array of `t_symbol *`s.

Parameters

<code>ac</code>	The count of <code>t_symbol *</code> s in <code>av</code>
<code>av</code>	An array of <code>t_symbol *</code> 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.

Referenced by `jit_object_exportattrs()`.

36.24.4.20 `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

<code>w</code>	The window with which this dialog is associated.
<code>s</code>	A string with any sprintf()-like formatting to be displayed.
<code>...</code>	Any variables that should be substituted in the string defined by <code>s</code> .

Returns

One of the values defined in `e_max_wind_advise_result`, depending on what the user selected.

36.24.4.21 `void wind_setcursor(short which)`

Change the cursor.

Parameters

<code>which</code>	One of the following predefined cursors: 1 #define C_ARROW 1 2 #define C_WATCH 2 3 #define C_IBEAM 3 4 #define C_HAND 4 5 #define C_CROSS 5 6 #define C_PENCIL 6 7 #define C_GROW 8
--------------------	--

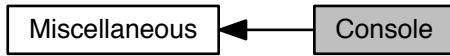
Remarks

[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 your own cursor directly.

36.25 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, 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), and grab the user's attention by displaying a banner in the patcher window.

36.25.1 Detailed Description

36.25.2 Function Documentation

36.25.2.1 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

<i>fmt</i>	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\(\)](#)

36.25.2.2 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

<i>fmt</i>	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\(\)](#)

Referenced by [jit_gl_begincapture\(\)](#), and [jit_gl_report_error\(\)](#).

36.25.2.3 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\(\)](#)

36.25.2.4 void object_error_obtrusive (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), 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\(\)](#)

36.25.2.5 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:

```
1 void myMethod(myObject **x, long someArgument)
2 {
3     object_post((t_object*)x, "This is my argument: %ld", someArgument);
4 }
```

See also

[object_error\(\)](#)

36.25.2.6 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

<i>x</i>	A pointer to your object.
<i>s</i>	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\(\)](#)

36.25.2.7 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

<i>s</i>	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\(\)](#)

36.25.2.8 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

<i>fmt</i>	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:

```
1 short whatIsIt;
2
3 whatIsIt = 999;
4 post ("the variable is %ld", (long)whatIsIt);
```

Remarks

The Max Window output when this code is executed.

```
1 the variable is 999
```

See also

[object_post\(\)](#)
[error\(\)](#)
[cpost\(\)](#)

Referenced by [jit_gl_report_error\(\)](#).

36.25.2.9 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.
----	---

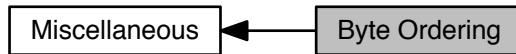
See also

[object_post\(\)](#)
[post\(\)](#)
[cpost\(\)](#)

36.26 Byte Ordering

Utilities for swapping the order of bytes to match the Endianness of the required platform.

Collaboration diagram for Byte Ordering:



Macros

- `#define BYTEORDER_SWAPW16(x)`
Switch the byte ordering of a short integer.
- `#define BYTEORDER_SWAPW32(x)`
Switch the byte ordering of an integer.
- `#define BYTEORDER_SWAPW64(x)`
Switch the byte ordering of an integer.
- `#define BYTEORDER_SWAPF32`
Switch the byte ordering of a float.
- `#define BYTEORDER_SWAPF64`
Switch the byte ordering of a double.

36.26.1 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.

36.26.2 Macro Definition Documentation

36.26.2.1 `#define BYTEORDER_SWAPF32`

Switch the byte ordering of a float.

Parameters

x	A float.
---	----------

Returns

A float with the byte-ordering swapped.

36.26.2.2 #define BYTEORDER_SWAPF64

Switch the byte ordering of a double.

Parameters

x	A double.
---	-----------

Returns

A double.

36.26.2.3 #define BYTEORDER_SWAPW16(x)

Switch the byte ordering of a short integer.

Parameters

x	A short integer.
---	------------------

Returns

A short integer with the byte-ordering swapped.

36.26.2.4 #define BYTEORDER_SWAPW32(x)

Switch the byte ordering of an integer.

Parameters

x	An integer.
---	-------------

Returns

An integer with the byte-ordering swapped.

36.26.2.5 #define BYTEORDER_SWAPW64(x)

Switch the byte ordering of an integer.

Parameters

x	An integer.
---	-------------

Returns

An integer with the byte-ordering swapped.

36.27 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 `t_ex_ex`
ex_ex.
- struct `t_expr`
Struct for an instance of expr.

Enumerations

- enum `e_max_expr_types` {
 `ET_INT`, `ET_FLT`, `ET_OP`, `ET_STR`,
`ET_TBL`, `ET_FUNC`, `ET_SYM`, `ET_VSYM`,
`ET_LP`, `ET_LB`, `ET_II`, `ET_FI`,
`ET_SI` }

Defines for ex_type.

Functions

- void * `expr_new` (long argc, `t_atom` *argv, `t_atom` *types)
Create a new expr object.
- short `expr_eval` (`t_expr` *x, long argc, `t_atom` *argv, `t_atom` *result)
Evaluate an expression in an expr object.

36.27.1 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.

36.27.2 Enumeration Type Documentation

36.27.2.1 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
- ET_FLT** a float
- ET_OP** operator
- ET_STR** string
- ET_TBL** a table, the content is a pointer
- ET_FUNC** a function
- ET_SYM** symbol ("string")
- ET_VSYM** variable symbol ("\$s?")
- ET_LP** left parenthesis
- ET_LB** left bracket
- ET_II** and integer inlet
- ET_FI** float inlet
- ET_SI** string inlet

36.27.3 Function Documentation

36.27.3.1 short expr_eval (*t_expr* * *x*, long *argc*, *t_atom* * *argv*, *t_atom* * *result*)

Evaluate an expression in an expr object.

Parameters

<i>x</i>	The expr object to evaluate.
<i>argc</i>	Count of arguments in argv.
<i>argv</i>	Array of nine <i>t_atom</i> values that will be substituted for variable arguments (such as \$i1) in the expression. Unused arguments should be of type <i>A NOTHING</i> .
<i>result</i>	A pre-existing <i>t_atom</i> 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 *t_atom* values. 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*.

36.27.3.2 void* expr_new (long *argc*, *t_atom* * *argv*, *t_atom* * *types*)

Create a new expr object.

Parameters

<i>argc</i>	Count of arguments in argv.
<i>argv</i>	Arguments that are used to create the expr. See the example below for details.
<i>types</i>	A pre-existing array of nine <i>t_atom</i> , that will hold the types of any variable arguments created in the expr. The types are returned in the <i>a_type</i> field of each <i>t_atom</i> . If an argument was not present, <i>A NOTHING</i> 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 *t_atom* values, 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:

```

1 $i1 (A_SYM)
2 + (A_SYM)
3 $f3 (A_SYM)
4 + (A_SYM)
5 3 (A_LONG)

```

After calling `expr_new`, types would contain the following:

Index	Argument	Type	Value
0	\$i1	A_LONG	0
1	+	A NOTHING	0
2	\$f3	A_FLOAT	0.0
3	+	A NOTHING	0
4		A NOTHING	0
5		A NOTHING	0
6		A NOTHING	0
7		A NOTHING	0
8		A NOTHING	0
9		A NOTHING	0

36.28 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.

36.28.1 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."

36.28.2 Function Documentation

36.28.2.1 short `table_dirty (t_symbol * s)`

Mark a table object as having changed data.

Parameters

<code>s</code>	Symbol containing the name of a table object.
----------------	---

Returns

If no table is associated with `tableName`, `table_dirty` returns a non-zero result.

36.28.2.2 short `table_get (t_symbol * s, long *** hp, long * sp)`

Get a handle to the data in a named table object.

Parameters

<i>s</i>	Symbol containing the name of the table object to find.
<i>hp</i>	Address of a handle where the table's data will be returned if the named table object is found.
<i>sp</i>	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:

```
1 long **storage,size,value;
2 if (!table_get(gensym("somename"),&storage,&size)) {
3     if (size > 40)
4         value = *((*storage)+40);
5 }
```

36.29 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](#)

36.30 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, [t_atom_long](#) val)
Restore the state of your object with a set message.
- void [preset_int](#) ([t_object](#) *x, [t_atom_long](#) n)
Restore the state of your object with an int message.

36.30.1 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);
    }
}
```

36.30.2 Function Documentation

36.30.2.1 void preset_int (**t_object** * *x*, **t_atom_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

<i>x</i>	Your object.
<i>n</i>	Current value of your object.

36.30.2.2 void preset_set (**t_object** * *obj*, **t_atom_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

<i>obj</i>	Your object.
<i>val</i>	Current value of your object.

36.30.2.3 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.

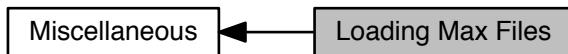
Parameters

<i>fmt</i>	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.

36.31 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.

36.31.1 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.

36.31.2 Function Documentation

36.31.2.1 void* [fileload](#) (C74_CONST char * name, short vol)

Load a patcher file by name and volume reference number.

Parameters

<i>name</i>	Filename of the patcher file to load (C string).
<i>vol</i>	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.

36.31.2.2 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()` was 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

<i>name</i>	Name of the file to open.
<i>volume</i>	Path ID specifying the location of the file.
<i>s</i>	A symbol.
<i>ac</i>	Count of t_atoms in av. To properly open a patcher file, ac should be 9.
<i>av</i>	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.
<i>couldedit</i>	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.

36.31.2.3 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

<i>name</i>	Name of the patcher file to load.
<i>volume</i>	Path ID specifying the location of the file.
<i>h</i>	Pointer to a handle variable that will receive the handle that contains the data in the file.
<i>sizep</i>	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.

36.31.2.4 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

<i>name</i>	Filename of the patcher file to load (C string).
-------------	--

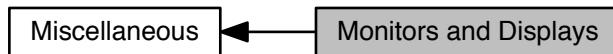
Returns

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.

36.32 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.

36.32.1 Detailed Description

Functions for finding our information about the environment.

36.32.2 Function Documentation

36.32.2.1 void `jmonitor_getdisplayrect (long workarea, long displayindex, t_rect * rect)`

Return the `t_rect` for a given display.

Parameters

<code>workarea</code>	Set workarea non-zero to clip out things like dock / task bar.
<code>displayindex</code>	The index number for a monitor. The primary monitor has an index of 0.
<code>rect</code>	The address of a valid <code>t_rect</code> whose values will be filled-in upon return.

36.32.2.2 void `jmonitor_getdisplayrect_foralldisplays (long workarea, t_rect * rect)`

Return a union of all display rects.

Parameters

<code>workarea</code>	Set workarea non-zero to clip out things like dock / task bar.
-----------------------	--

<i>rect</i>	The address of a valid t_rect whose values will be filled-in upon return.
-------------	---

36.32.2.3 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

<i>workarea</i>	Set workarea non-zero to clip out things like dock / task bar.
<i>pt</i>	A point, for which the monitor will be determined and the rect returned.
<i>rect</i>	The address of a valid t_rect whose values will be filled-in upon return.

36.32.2.4 long jmonitor_getnumdisplays ()

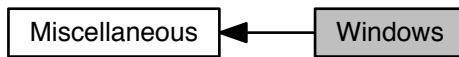
Return the number of monitors on which can be displayed.

Returns

The number of monitors.

36.33 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.

36.33.1 Detailed Description

36.33.2 Function Documentation

36.33.2.1 `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.

36.33.2.2 `t_object* jwind_getat (long index)`

Return a pointer to the window with a given index.

Parameters

<code>index</code>	Get window at index (0 to count-1).
--------------------	-------------------------------------

Returns

A pointer to a window object.

36.33.2.3 `long jwind_getcount (void)`

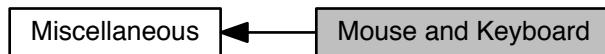
Determine how many windows exist.

Returns

The number of windows.

36.34 Mouse and Keyboard

Collaboration diagram for Mouse and Keyboard:



Enumerations

- enum `t_modifiers` {
 `eCommandKey, eShiftKey, eControlKey, eAltKey,`
`eLeftButton, eRightButton, eMiddleButton, ePopupMenu,`
`eCapsLock, eAutoRepeat` }
- 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_RESIZE_LEFTRIGHT, JMOUSE_CURSOR_RESIZE_UPDOWN, JMOUSE_CURSOR_RESIZE_FOURWAY,`
`JMOUSE_CURSOR_RESIZE_DRAGGINGHAND,`
`JMOUSE_CURSOR_RESIZE_LEFTEdge, JMOUSE_CURSOR_RESIZE_UPEdge,`
`JMOUSE_CURSOR_RESIZE_BOTTOMEDGE, JMOUSE_CURSOR_RESIZE_BOTTOMLEFTCORNER,`
`JMOUSE_CURSOR_RESIZE_TOPRIGHTCORNER, 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.

36.34.1 Detailed Description

36.34.2 Enumeration Type Documentation

36.34.2.1 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.

36.34.2.2 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.

36.34.3 Function Documentation

36.34.3.1 `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.

36.34.3.2 `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.

36.34.3.3 `void jmouse_getposition_global(int *x, int *y)`

Get the position of the mouse cursor in screen coordinates.

Parameters

<code>x</code>	The address of a variable to hold the x-coordinate upon return.
<code>y</code>	The address of a variable to hold the y-coordinate upon return.

36.34.3.4 `void jmouse_setcursor(t_object *patcherview, t_object *box, t_jmouse_cursortype type)`

Set the mouse cursor.

Parameters

<code>patcherview</code>	The patcherview for which the cursor should be applied.
<code>box</code>	The box for which the cursor should be applied.
<code>type</code>	The type of cursor for the mouse to use.

36.34.3.5 `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

<code>patcherview</code>	The patcherview containing the box upon which the mouse coordinates are based.
<code>box</code>	The box upon which the mouse coordinates are based.
<code>bx</code>	The new x-coordinate of the mouse cursor position.
<code>by</code>	The new y-coordinate of the mouse cursor position.

36.34.3.6 `void jmouse_setposition_global(int x, int y)`

Set the position of the mouse cursor in screen coordinates.

Parameters

<i>x</i>	The new x-coordinate of the mouse cursor position.
<i>y</i>	The new y-coordinate of the mouse cursor position.

36.34.3.7 void jmouse_setposition_view (*t_object* * *patcherview*, double *cx*, double *cy*)

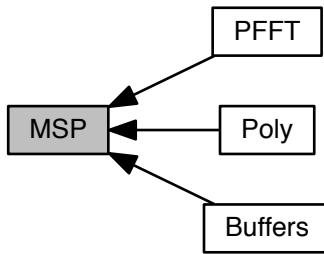
Set the position of the mouse cursor relative to the patcher canvas coordinates.

Parameters

<i>patcherview</i>	The patcherview upon which the mouse coordinates are based.
<i>cx</i>	The new x-coordinate of the mouse cursor position.
<i>cy</i>	The new y-coordinate of the mouse cursor position.

36.35 MSP

Collaboration diagram for MSP:



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.

Data Structures

- struct **t_pxdata**

Common struct for MSP objects.

- 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.

Macros

- #define **Z_NO_INPLACE**

flag indicating the object doesn't want signals in place

- #define **Z_PUT_LAST**

when list of ugens is resorted, put this object at end

- #define **Z_PUT_FIRST**

when list of ugens is resorted, put this object at beginning

- #define **PI**

The pi constant.

- #define **TWOPI**

- `#define PIOVERTWO`
Twice the pi constant.
- `#define dsp_setup`
This is commonly used rather than directly calling `z_dsp_setup()` in MSP objects.
- `#define dsp_free`
This is commonly used rather than directly calling `z_dsp_free()` in MSP objects.

Typedefs

- `typedef void * t_vptr`
A void pointer.
- `typedef void * vptr`
A void pointer.
- `typedef float t_float`
A float – always a 32 bit floating point number.
- `typedef double t_double`
A double – always a 64 bit floating point number.
- `typedef double t_sample`
A sample value – width determined by MSP version.

Enumerations

- `enum { SYS_MAXBLKSIZE, SYS_MAXSIGS }`
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_advv (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_dspinitbbox (t_class *c)`
This routine must be called in your object's initialization routine.

Variables

- `BEGIN_USING_C_LINKAGE` `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.

36.35.1 Detailed Description

36.35.2 Macro Definition Documentation

36.35.2.1 `#define PI`

The pi constant.

36.35.2.2 `#define PIOVERTWO`

Half of the pi constant.

36.35.2.3 `#define TWOPI`

Twice the pi constant.

36.35.3 Typedef Documentation

36.35.3.1 `typedef double t_double`

A double – always a 64 bit floating point number.

36.35.3.2 `typedef float t_float`

A float – always a 32 bit floating point number.

36.35.3.3 `typedef double t_sample`

A sample value – width determined by MSP version.

36.35.3.4 `typedef void* t_vptr`

A void pointer.

36.35.3.5 `typedef void* vptr`

A void pointer.

36.35.4 Enumeration Type Documentation

36.35.4.1 `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

36.35.5 Function Documentation

36.35.5.1 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

<i>c</i>	The class to make dsp-ready.
----------	------------------------------

See also

[class_dspinitjbox\(\)](#)

36.35.5.2 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

<i>c</i>	The class to make dsp-ready.
----------	------------------------------

See also

[class_dspinit\(\)](#)

36.35.5.3 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

<i>f</i>	The perform routine to use for processing audio.
----------	--

<i>n</i>	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](#)

36.35.5.4 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

<i>f</i>	The perform routine to use for processing audio.
<i>n</i>	The number of arguments that will follow in the vector parameter.
<i>vector</i>	The arguments that will be passed to the perform routine.

See also

[The DSP Method and Perform Routine](#)
[Using Connection Information](#)

36.35.5.5 int sys_getblksize (void)

Query MSP for the current global vector (block) size.

Returns

The current global vector size for the MSP environment.

36.35.5.6 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.

36.35.5.7 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.

36.35.5.8 int sys_getmaxblksize (void)

Query MSP for the maximum global vector (block) size.

Returns

The maximum global vector size for the MSP environment.

36.35.5.9 float sys_getsr (void)

Query MSP for the global sample rate.

Returns

The global sample rate of the MSP environment.

36.35.5.10 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](#)

36.35.5.11 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](#)

36.35.6 Variable Documentation

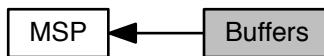
36.35.6.1 BEGIN_USING_C_LINKAGE typedef t_int*(* t_perfroutine) (t_int *args)

A function pointer for the audio perform routine used by MSP objects to process blocks of samples.

36.36 Buffers

Your object can access shared data stored in an MSP buffer~ object.

Collaboration diagram for Buffers:



Data Structures

- struct [t_buffer_info](#)
Common buffer~ data/metadata.

Typedefs

- typedef struct _buffer_ref [t_buffer_ref](#)
A buffer~ reference.
- typedef [t_object](#) [t_buffer_obj](#)
A buffer~ object.

Functions

- [BEGIN_USING_C_LINKAGE t_buffer_ref * buffer_ref_new \(t_object *self, t_symbol *name\)](#)
Create a reference to a buffer~ object by name.
- void [buffer_ref_set \(t_buffer_ref *x, t_symbol *name\)](#)
Change a buffer reference to refer to a different buffer~ object by name.
- [t_atom_long buffer_ref_exists \(t_buffer_ref *x\)](#)
Query to find out if a buffer~ with the referenced name actually exists.
- [t_buffer_obj * buffer_ref_getobject \(t_buffer_ref *x\)](#)
Query a buffer reference to get the actual buffer~ object being referenced, if it exists.
- [t_max_err buffer_ref_notify \(t_buffer_ref *x, t_symbol *s, t_symbol *msg, void *sender, void *data\)](#)
Your object needs to handle notifications issued by the buffer~ you reference.
- void [buffer_view \(t_buffer_obj *buffer_object\)](#)
Open a viewer window to display the contents of the buffer~.
- float * [buffer_locksamples \(t_buffer_obj *buffer_object\)](#)
Claim the buffer~ and get a pointer to the first sample in memory.
- void [buffer_unlocksamples \(t_buffer_obj *buffer_object\)](#)
Release your claim on the buffer~ contents so that other objects may read/write to the buffer~.
- [t_atom_long buffer_getchannelcount \(t_buffer_obj *buffer_object\)](#)
Query a buffer~ to find out how many channels are present in the buffer content.
- [t_atom_long buffer_getframecount \(t_buffer_obj *buffer_object\)](#)
Query a buffer~ to find out how many frames long the buffer content is in samples.
- [t_atom_float buffer_getsamplerate \(t_buffer_obj *buffer_object\)](#)
Query a buffer~ to find out its native sample rate in samples per second.

- `t_atom_float buffer_getmillisamplerate (t_buffer_obj *buffer_object)`
Query a buffer~ to find out its native sample rate in samples per millisecond.
- `t_max_err buffer_setpadding (t_buffer_obj *buffer_object, t_atom_long samplecount)`
Set the number of samples with which to zero-pad the buffer~'s contents.
- `t_max_err buffer_setdirty (t_buffer_obj *buffer_object)`
Set the buffer's dirty flag, indicating that changes have been made.
- `t_symbol * buffer_getfilename (t_buffer_obj *buffer_object)`
Retrieve the name of the last file to be read by a buffer~.

36.36.1 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 direct gain access to the `t_buffer` struct. This is potentially dangerous, and not guaranteed to be forward (or backward) compatible. Beginning with Max 6.1, developers accessing buffer~ objects are encouraged to use the `t_buffer_ref` API. The `t_buffer_ref` API provides many enhancements to improve thread-safety, simplify your perform routine, and manage the binding to the buffer~ object.

A class that accesses a buffer~ is the simpwave~ object included with Max SDK example projects.

While the Max 6 signal processing chain operates on 64-bit double-precision floats, the `t_buffer_obj` storage remains as 32-bit single-precision float format. This is essential to maintain backward compatibility with older third-party externals.

If you have written to the buffer~ and thus changed the values of its samples, you should now mark the buffer~ as dirty. This will ensure that objects such as waveform~ update their rendering of the contents of this buffer~. This can be accomplished with the following call:

```
object_method(b, gensym("dirty"));
```

36.36.2 Typedef Documentation

36.36.2.1 `typedef t_object t_buffer_obj`

A buffer~ object.

This represents the actual buffer~ object. You can use this to send messages, query attributes, etc. of the actual buffer object referenced by a `t_buffer_ref`.

36.36.2.2 `typedef struct _buffer_ref t_buffer_ref`

A buffer~ reference.

Use this struct to represent a reference to a buffer~ object in Max. Use the `buffer_ref_getbuffer()` call to return a pointer to the buffer. You can then make calls on the buffer itself.

36.36.3 Function Documentation

36.36.3.1 `t_atom_long buffer_getchannelcount (t_buffer_obj * buffer_object)`

Query a buffer~ to find out how many channels are present in the buffer content.

Parameters

<i>buffer_object</i>	the buffer object
----------------------	-------------------

Returns

the number of channels in the buffer

36.36.3.2 t_symbol* buffer_getfilename (t_buffer_obj * *buffer_object*)

Retrieve the name of the last file to be read by a buffer~.

(Not the last file written).

Parameters

<i>buffer_object</i>	the buffer object
----------------------	-------------------

Returns

The name of the file last read, or gensym("") if no files have been read.

Version

Introduced in Max 7.0.1

36.36.3.3 t_atom_long buffer_getframecount (t_buffer_obj * *buffer_object*)

Query a buffer~ to find out how many frames long the buffer content is in samples.

Parameters

<i>buffer_object</i>	the buffer object
----------------------	-------------------

Returns

the number of frames in the buffer

36.36.3.4 t_atom_float buffer_getmillisamplerate (t_buffer_obj * *buffer_object*)

Query a buffer~ to find out its native sample rate in samples per millisecond.

Parameters

<i>buffer_object</i>	the buffer object
----------------------	-------------------

Returns

the sample rate in samples per millisecond

36.36.3.5 t_atom_float buffer_getsamplerate (t_buffer_obj * *buffer_object*)

Query a buffer~ to find out its native sample rate in samples per second.

Parameters

<i>buffer_object</i>	the buffer object
----------------------	-------------------

Returns

the sample rate in samples per second

36.36.3.6 float* buffer_locksamples (t_buffer_obj * *buffer_object*)

Claim the buffer~ and get a pointer to the first sample in memory.

When you are done reading/writing to the buffer you must call [buffer_unlocksamples\(\)](#). If the attempt to claim the buffer~ fails the returned pointer will be NULL.

Parameters

<i>buffer_object</i>	the buffer object
----------------------	-------------------

Returns

a pointer to the first sample in memory, or NULL if the buffer doesn't exist.

36.36.3.7 t_atom_long buffer_ref_exists (t_buffer_ref * *x*)

Query to find out if a buffer~ with the referenced name actually exists.

Parameters

<i>x</i>	the buffer reference
----------	----------------------

Returns

non-zero if the buffer~ exists, otherwise zero

36.36.3.8 t_buffer_obj* buffer_ref_getobject (t_buffer_ref * *x*)

Query a buffer reference to get the actual buffer~ object being referenced, if it exists.

Parameters

<i>x</i>	the buffer reference
----------	----------------------

Returns

the buffer object if exists, otherwise NULL

36.36.3.9 BEGIN_USING_C_LINKAGE t_buffer_ref* buffer_ref_new (t_object * *self*, t_symbol * *name*)

Create a reference to a buffer~ object by name.

You must release the buffer reference using [object_free\(\)](#) when you are finished using it.

Parameters

<i>self</i>	pointer to your object
<i>name</i>	the name of the buffer~

Returns

a pointer to your new buffer reference

36.36.3.10 `t_max_err buffer_ref_notify (t_buffer_ref * x, t_symbol * s, t_symbol * msg, void * sender, void * data)`

Your object needs to handle notifications issued by the buffer~ you reference.

You do this by defining a "notify" method. Your notify method should then call this notify method for the `t_buffer_ref`.

Parameters

<i>x</i>	the buffer reference
<i>s</i>	the registered name of the sending object
<i>msg</i>	then name of the notification/message sent
<i>sender</i>	the pointer to the sending object
<i>data</i>	optional argument sent with the notification/message

Returns

a max error code

36.36.3.11 `void buffer_ref_set (t_buffer_ref * x, t_symbol * name)`

Change a buffer reference to refer to a different buffer~ object by name.

Parameters

<i>x</i>	the buffer reference
<i>name</i>	the name of a different buffer~ to reference

36.36.3.12 `t_max_err buffer_setdirty (t_buffer_obj * buffer_object)`

Set the buffer's dirty flag, indicating that changes have been made.

Parameters

<i>buffer_object</i>	the buffer object
----------------------	-------------------

Returns

an error code

36.36.3.13 `t_max_err buffer_setpadding (t_buffer_obj * buffer_object, t_atom_long samplecount)`

Set the number of samples with which to zero-pad the buffer~'s contents.

The typical application for this need is to pad a buffer with enough room to allow for the reach of a FIR kernel in convolution.

Parameters

<i>buffer_object</i>	the buffer object
<i>samplecount</i>	the number of sample to pad the buffer with on each side of the contents

Returns

an error code

36.36.3.14 void buffer_unlocksamples (t_buffer_obj * *buffer_object*)

Release your claim on the buffer~ contents so that other objects may read/write to the buffer~.

Parameters

<i>buffer_object</i>	the buffer object
----------------------	-------------------

36.36.3.15 void buffer_view (t_buffer_obj * *buffer_object*)

Open a viewer window to display the contents of the buffer~.

Parameters

<i>buffer_object</i>	the buffer object
----------------------	-------------------

36.37 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.

36.37.1 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~
}
  
```

36.38 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);
    }
}
```

36.39 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.

Macros

- `#define MAGIC`
Magic number used to determine if memory pointed to by a `t_object` is valid.*
- `#define NOGOOD(x)`
Returns true if a pointer is not a valid object.
- `#define MSG_MAXARG`
Maximum number of arguments that can be passed as a typed-list rather than using `A_GIMME`.
- `#define object_method_direct(rt, sig, x, s, ...)`
do a strongly typed direct call to a method of an object

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_fromboxtext (t_object *patcher, const char *text)`
Create an object from the passed in text.
- `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)`

- `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.
- `t_max_err object_register_getnames (t_symbol *name_space, long *namecount, t_symbol ***names)`

Returns all registered names in a namespace.
- `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.
- `void * object_subscribe (t_symbol *name_space, t_symbol *s, t_symbol *classname, void *x)`

Subscribes a client to wait for an object to register.
- `t_max_err object_unsubscribe (t_symbol *name_space, t_symbol *s, t_symbol *classname, void *x)`

Unsubscribe a client from a registered object, detaching if the object is registered.
- `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.
- `void * object_super_method (t_object *x, t_symbol *s,...)`

Sends an untyped message to an object using superclass methods.
- `void * object_this_method (t_object *x, t_symbol *s,...)`

Sends an untyped message to an object, respects a thread specific class stack from `object_super_method()` calls.
- `t_max_err object_attr_touch (t_object *x, t_symbol *attrname)`

Mark an attribute as being touched by some code not from the attribute setter.
- `t_max_err object_attr_touch_parse (t_object *x, char *attrnames)`

Mark one or more attributes as being touched by some code not from the attribute setter.
- `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, t_atom_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.

36.39.1 Detailed Description

See also

<http://www.cycling74.com/twiki/bin/view/ProductDocumentation/JitterSdk<ObjectModel>
<http://www.cycling74.com/twiki/bin/view/ProductDocumentation/JitterSdk<RegNotify>

36.39.2 Macro Definition Documentation

36.39.2.1 #define MSG_MAXARG

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](#).

36.39.2.2 #define object_method_direct(*rt*, *sig*, *x*, *s*, ...)

do a strongly typed direct call to a method of an object

Parameters

<i>rt</i>	The type of the return value (double, void*, void...)
<i>sig</i>	the actual signature of the function in brackets ! something like (t_object *, double, long)
<i>x</i>	The object where the method we want to call will be looked for, it will also always be the first argument to the function call
<i>s</i>	The message selector
...	Any arguments to the call, the first one will always be the object (x)

Returns

will return anything that the called function returns, typed by (rt)

Remarks

Example: To call the function identified by `getcolorat` on the object `pwindow` which is declared like:
`t_jrgba pwindow_getcolorat(t_object *window, double x, double y)`

```

1 double x = 44.73;
2 double y = 79.21;
3 t_object *pwindow;
4 t_jrgba result = object_method_direct(t_jrgba, (t_object *, double, double), pwindow, gensym("getcolorat"),
    x, y);

```

36.39.3 Function Documentation

36.39.3.1 void classname_openhelp (char * *classname*)

Open the help patcher for a given object class name.

Parameters

<i>classname</i>	The class name for which to open the help patcher.
------------------	--

36.39.3.2 void classname_openquery (char * *classname*)

Open a search in the file browser for files with the name of the given class.

Parameters

<i>classname</i>	The class name for which to query.
------------------	------------------------------------

36.39.3.3 void classname_openrefpage (char * *classname*)

Open the reference page for a given object class name.

Parameters

<i>classname</i>	The class name for which to open the reference page.
------------------	--

36.39.3.4 t_object* newobject_fromboxtext (t_object * patcher, const char * text)

Create an object from the passed in text.

The passed in text is in the same format as would be typed into an object box. It can be used for UI objects or text objects so this is the simplest way to create objects from C.

Parameters

<i>patcher</i>	An instance of a patcher object.
<i>text</i>	The text as if typed into an object box.

Returns

A pointer to the newly created object instance, or NULL if creation of the object fails.

See also

[newobject_sprintf\(\)](#)

36.39.3.5 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

<i>patcher</i>	An instance of a patcher object.
<i>d</i>	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,

```
1 "@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".

```
1 t_dictionary *d;
2 t_object *o;
3 char text[4];
4
5 strncpy_zero(text, "foo", 4);
6
7 d = dictionary_sprintf("@maxclass comment @varname _name \
8 @text \"%s\" @patching_rect %.2f %.2f %.2f %.2f \
9 @fontsize %f @textcolor %f %f %f 1.0 \
10 @fontname %s @bgcolor 0.001 0.001 0.001 0.001 0.", \
11 text, 20.0, 20.0, 200.0, 24.0, \
12 18, 0.9, 0.9, 0.9, "Arial");
13
14 o = newobject_fromdictionary(aPatcher, d);
```

See also

[newobject_sprintf\(\)](#)
[newobject_fromdictionary\(\)](#)
[atom_setparse\(\)](#)

36.39.3.6 **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

<i>patcher</i>	An instance of a patcher object.
<i>fmt</i>	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,

```
1 "@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".

```
1 t_object *my_comment;
2 char text[4];
3
4 strncpy_zero(text, "foo", 4);
5
6 my_comment = newobject_sprintf(aPatcher, "@maxclass comment @varname _name \
7     @text \"%s\" @patching_rect %.2f %.2f %.2f %.2f \
8     @fontsize %f @textcolor %f %f %f 1.0 \
9     @fontname %s @bgcolor 0.001 0.001 0.001 0.001 \
10    text, 20.0, 20.0, 200.0, 24.0,
11    18, 0.9, 0.9, 0.9, "Arial");
```

See also

[dictionary_sprintf\(\)](#)
[newobject_fromdictionary\(\)](#)
[atom_setparse\(\)](#)

36.39.3.7 **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

<i>c</i>	The class pointer, returned by class_new()
----------	--

Returns

This function returns a new instance of an object class if successful, or NULL if unsuccessful.

36.39.3.8 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

<code>name_space</code>	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.
<code>s</code>	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.
<code>x</code>	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\(\)](#)

Referenced by [jit_object_attach\(\)](#).

36.39.3.9 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

<code>x</code>	The attaching client object. Generally, this is the pointer to your Max object.
<code>registeredobject</code>	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\(\)](#)

36.39.3.10 **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

<i>x</i>	The attaching client object. Generally, this is the pointer to your Max object.
<i>object_to_attach</i>	A pointer to the object to which you wish to register and then to which to attach.
<i>reg_name_space</i>	The namespace in which to register the <i>object_to_attach</i> .

Returns

A Max error code.

See also

[object_register\(\)](#)
[object_attach_byptr\(\)](#)

36.39.3.11 **t_max_err object_attr_touch (t_object * x, t_symbol * attrname)**

Mark an attribute as being touched by some code not from the attribute setter.

This will notify clients that the attribute has changed.

Parameters

<i>x</i>	The object whose attribute has been changed
<i>attrname</i>	The attribute name

Returns

A Max error code

36.39.3.12 **t_max_err object_attr_touch_parse (t_object * x, char * attrnames)**

Mark one or more attributes as being touched by some code not from the attribute setter.

This will notify clients that the attributes have changed. Utility to call [object_attr_touch\(\)](#) for several attributes

Parameters

<i>x</i>	The object whose attribute has been changed
<i>attrnames</i>	The attribute names as a space separated string

Returns

A Max error code

36.39.3.13 `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.

Referenced by `jit_object_class()`, and `max_jit_obex_new()`.

36.39.3.14 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.

Referenced by `jit_object_classname()`.

36.39.3.15 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:

```
1 long isprint = object_classname_compare(x, gensym("print"));
```

36.39.3.16 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 <code>object_register()</code> to register the object. If you don't know the registered object's namespace, the <code>object_findregisteredbyptr()</code> function can be used to determine it.
------------	---

<i>s</i>	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.
<i>x</i>	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.

Referenced by `jit_object_detach()`.

36.39.3.17 t_max_err object_detach_byptr (void * *x*, void * *registeredobject*)

Detach a client from a registered object.

Parameters

<i>x</i>	The attaching client object. Generally, this is the pointer to your Max object.
<i>registeredobject</i>	The object from which to detach.

Returns

A Max error code.

See also

[object_detach\(\)](#)
[object_attach_byptr\(\)](#)

36.39.3.18 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

<i>ac</i>	The number of atoms.
<i>av</i>	A pointer to the first atom in the array.

Returns

The dictionary retrieved from the atoms.

See also

[attr_dictionary_process\(\)](#)

36.39.3.19 void* object_findregistered (t_symbol * *name_space*, t_symbol * *s*)

Determines a registered object's pointer, given its namespace and name.

Parameters

<i>name_space</i>	The namespace of the registered object
<i>s</i>	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.

Referenced by `jit_object_findregistered()`.

36.39.3.20 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

<i>name_space</i>	Pointer to a <code>t_symbol *</code> , to receive the namespace of the registered object
<i>s</i>	Pointer to a <code>t_symbol *</code> , to receive the name of the registered object within the namespace
<i>x</i>	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.

Referenced by `jit_object_findregisteredbyptr()`.

36.39.3.21 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

<i>x</i>	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.

Referenced by `jit_object_free()`.

36.39.3.22 method object_getmethod (void * x, t_symbol * s)

Retrieves an object's `method` for a particular message selector.

Parameters

<i>x</i>	The object whose method is being queried
<i>s</i>	The message selector

Returns

This function returns the `method` if successful, or `method_false()` if unsuccessful.

Referenced by `jit_object_getmethod()`.

36.39.3.23 `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

<code>x</code>	The object whose value is of interest
<code>ac</code>	Pointer to a long variable to receive the count of arguments in <code>av</code> . The long variable itself should be set to 0 previous to calling this function.
<code>av</code>	Pointer to a <code>t_atom *</code> , to receive object data. The <code>t_atom *</code> 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:

```
1 class_addmethod(c, (method)myobject_getvalueof, "getvalueof", A_CANT, 0);
```

The `getvalueof` method should be prototyped as:

```
1 t_max_err myobject_getvalueof(t_myobject *x, long *ac, t_atom **av);
```

And implemented, generally, as:

```
1 t_max_err myobj_getvalueof(t_myobj *x, long *ac, t_atom **av)
2 {
3     if (ac && av) {
4         if (*ac && *av) {
5             // memory has been passed in; use it.
6         } else {
7             // allocate enough memory for your data
8             *av = (t_atom *)getbytes(sizeof(t_atom));
9         }
10        *ac = 1; // our data is a single floating point value
11        atom_setfloat(*av, x->objvalue);
12    }
13    return MAX_ERR_NONE;
14 }
15
16 @remark      By convention, and to permit the interoperability of objects using the obex API,
17 developers should allocate memory in their <tt>getvalueof</tt> methods using the getbytes()
function.
```

Referenced by `max_jit_obex_gimmeback_dumpout()`.

36.39.3.24 `void* object_method(void *x, t_symbol *s, ...)`

Sends an untyped message to an object.

There are some caveats to its use, however, particularly for 64-bit architectures. `object_method_direct()` should be used in cases where floating-point or other non-integer types are being passed on the stack or in return values.

Parameters

<i>x</i>	The object that will receive the message
<i>s</i>	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`:

```
1 void *bang_result;
2 bang_result = object_method(bang_me, gensym("bang"));
```

Referenced by [jit_attr_symcompare\(\)](#).

36.39.3.25 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

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>v</i>	An argument to pass to the method.
<i>rv</i>	The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

36.39.3.26 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

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>ac</i>	The number of arguments to pass to the method.
<i>av</i>	The address of the first of the array of arguments to pass to the method.
<i>rv</i>	The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

36.39.3.27 `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

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>v</i>	An argument to pass to the method.
<i>rv</i>	The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

36.39.3.28 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

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>ac</i>	The number of arguments to pass to the method.
<i>av</i>	The address of the first of the array of arguments to pass to the method.
<i>rv</i>	The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

36.39.3.29 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

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>v</i>	An argument to pass to the method.
<i>rv</i>	The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

36.39.3.30 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

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>ac</i>	The number of arguments to pass to the method.
<i>av</i>	The address of the first of the array of arguments to pass to the method.
<i>rv</i>	The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

36.39.3.31 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

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>rv</i>	The address of an atom to hold a return value.
<i>fmt</i>	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\(\)](#)

36.39.3.32 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

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>v</i>	An argument to pass to the method.
<i>rv</i>	The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

36.39.3.33 **t_max_err object_method_long_array(t_object *x, t_symbol *s, long ac, t_atom_long *av, t_atom *rv)**

Convenience wrapper for [object_method_typed\(\)](#) that passes an array of long integers values as an argument.

Parameters

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>ac</i>	The number of arguments to pass to the method.
<i>av</i>	The address of the first of the array of arguments to pass to the method.
<i>rv</i>	The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

36.39.3.34 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

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>v</i>	An argument to pass to the method.
<i>rv</i>	The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

36.39.3.35 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

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>ac</i>	The number of arguments to pass to the method.
<i>av</i>	The address of the first of the array of arguments to pass to the method.
<i>rv</i>	The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

36.39.3.36 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

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>parsestr</i>	A C-string to parse into an array of atoms to pass to the method.
<i>rv</i>	The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)
[atom_setparse\(\)](#)

36.39.3.37 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

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>v</i>	An argument to pass to the method.
<i>rv</i>	The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

36.39.3.38 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

<i>x</i>	The object to which the message will be sent.
<i>s</i>	The name of the method to call on the object.
<i>ac</i>	The number of arguments to pass to the method.
<i>av</i>	The address of the first of the array of arguments to pass to the method.
<i>rv</i>	The address of an atom to hold a return value.

Returns

A Max error code.

See also

[object_method_typed\(\)](#)

36.39.3.39 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

<i>x</i>	The object that will receive the message
<i>s</i>	The message selector
<i>ac</i>	Count of message arguments in <i>av</i>
<i>av</i>	Array of <i>t_atoms</i> ; the message arguments
<i>rv</i>	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.

Referenced by [jit_object_method_typed\(\)](#), [max_jit_attr_args\(\)](#), [max_jit_obex_gimmeback\(\)](#), and [max_jit_obex_gimmeback_dumpout\(\)](#).

36.39.3.40 *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

<i>x</i>	The object that will receive the message
<i>mp</i>	Undocumented
<i>s</i>	The message selector
<i>ac</i>	Count of message arguments in <i>av</i>
<i>av</i>	Array of <i>t_atoms</i> ; the message arguments
<i>rv</i>	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.

36.39.3.41 *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

<i>name_space</i>	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.
<i>classname</i>	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.

36.39.3.42 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

<i>name_space</i>	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.
<i>classname</i>	The name of the class of the object to be created
<i>ac</i>	Count of arguments in <i>av</i>
<i>av</i>	Array of <i>t_atoms</i> ; 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.

36.39.3.43 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

<i>x</i>	Pointer to the registered object
<i>s</i>	The message to send
<i>data</i>	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:

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

The `notify` method should be prototyped as:

```
1 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.

Referenced by `jit_object_notify()`, `max_jit_attr_set()`, and `max_jit_obex_attr_set()`.

36.39.3.44 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

<i>x</i>	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()).
<i>s</i>	The message selector t_symbol *
<i>argc</i>	Number of elements in the argument list in <i>argv</i>
<i>argv</i>	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.

Referenced by [ext_main\(\)](#).

36.39.3.45 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

<i>x</i>	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()).
<i>key</i>	The symbolic name for the data to be retrieved
<i>val</i>	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:

```

1 void post_containers(t_obexobj **x)
2 {
3     t_patcher *p;
4     t_box *b;
5     t_max_err err;
6
7     err = object_obex_lookup(x, gensym("#P"), (t_object **) &p);
8     err = object_obex_lookup(x, gensym("#B"), (t_object **) &b);
9
10    post("my patcher is located at 0x%X", p);
11    post("my box is located at 0x%X", b);
12 }
```

36.39.3.46 t_max_err object_obex_store (void * *x*, t_symbol * *key*, t_object * *val*)

Stores data in the object's obex.

Parameters

<i>x</i>	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()).
<i>key</i>	A symbolic name for the data to be stored
<i>val</i>	A t_object *, to be stored in the obex, referenced under the <i>key</i> .

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:

```
1 object_obex_store(x, _sym_dumpout, outlet_new(x, NULL));
```

36.39.3.47 void object_openhelp (t_object * x)

Open the help patcher for a given instance of an object.

Parameters

<i>x</i>	The object instance for which to open the help patcher.
----------	---

36.39.3.48 void object_openquery (t_object * x)

Open a search in the file browser for files with the name of the given object.

Parameters

<i>x</i>	The object instance for which to query.
----------	---

36.39.3.49 void object_openrefpage (t_object * x)

Open the reference page for a given instance of an object.

Parameters

<i>x</i>	The object instance for which to open the reference page.
----------	---

36.39.3.50 void* object_register (t_symbol * name_space, t_symbol * s, void * x)

Registers an object in a namespace.

Parameters

<i>name_space</i>	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.
<i>s</i>	The name of the object in the namespace. This name will be used by other objects to attach and detach from the registered object.
<i>x</i>	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()`.

Referenced by `jit_object_register()`.

36.39.3.51 t_max_err object_register_getnames (t_symbol * name_space, long * namecount, t_symbol * names)**

Returns all registered names in a namespace.

Parameters

<code>name_space</code>	Pointer to a <code>t_symbol</code> , the namespace to lookup names in
<code>namecount</code>	Pointer to a long, to receive the count of the registered names within the namespace
<code>names</code>	Pointer to a <code>t_symbol **</code> , to receive the allocated names. This pointer should be freed after use

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in "ext_obex.h" if unsuccessful.

36.39.3.52 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

<code>x</code>	The object whose value is of interest
<code>ac</code>	The count of arguments in <code>av</code>
<code>av</code>	Array of <code>t_atoms</code> ; 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:

```
1 class_addmethod(c, (method)myobject_setvalueof, "setvalueof", A_CANT, 0);
```

The `setvalueof` method should be prototyped as:

```
1 t_max_err myobject_setvalueof(t_myobject *x, long *ac, t_atom **av);
```

And implemented, generally, as:

```
1 t_max_err myobject_setvalueof(t_myobject *x, long ac, t_atom *av)
2 {
3     if (ac && av) {
4         // simulate receipt of a float value
5         myobject_float(x, atom_getfloat(av));
6     }
7     return MAX_ERR_NONE;
8 }
```

36.39.3.53 void* object_subscribe (t_symbol * name_space, t_symbol * s, t_symbol * classname, void * x)

Subscribes a client to wait for an object to register.

Upon registration, the object will attach. 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. See below for more information, in the reference for `object_notify`.

Parameters

<code>name_space</code>	The namespace of the registered object. This should be the same value used in <code>object_register</code> to register the object. If you don't know the registered object's namespace, the <code>object_findregisteredbyptr</code> function can be used to determine it.
<code>s</code>	The name of the registered object in the namespace. If you don't know the name of the registered object, the <code>object_findregisteredbyptr</code> function can be used to determine it.
<code>classname</code>	The classname of the registered object in the namespace to use as a filter. If NULL, then it will attach to any class of object.
<code>x</code>	The client object to attach. Generally, this is the pointer to your Max object.

Returns

This function returns a pointer to the object if registered (to the object referred to by the combination of `name_space` and `s` arguments) if successful, or NULL if the object is not yet registered.

36.39.3.54 void* object_super_method (t_object * x, t_symbol * s, ...)

Sends an untyped message to an object using superclass methods.

Uses a thread specific stack to ensure traversal up the class hierarchy.

Parameters

<code>x</code>	The object that will receive the message
<code>s</code>	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.

36.39.3.55 void* object_this_method (t_object * x, t_symbol * s, ...)

Sends an untyped message to an object, respects a thread specific class stack from `object_super_method()` calls.

Parameters

<code>x</code>	The object that will receive the message
<code>s</code>	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.

36.39.3.56 **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.

Referenced by [jit_object_unregister\(\)](#).

36.39.3.57 t_max_err object_unsubscribe (t_symbol * name_space, t_symbol * s, t_symbol * classname, void * x)

Unsubscribe a client from a registered object, detaching if the object is registered.

Parameters

<i>name_space</i>	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.
<i>s</i>	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.
<i>classname</i>	The classname of the registered object in the namespace to use as a filter. Currently unused for unsubscribe.
<i>x</i>	The client object to detach. Generally, this is the pointer to your Max object.

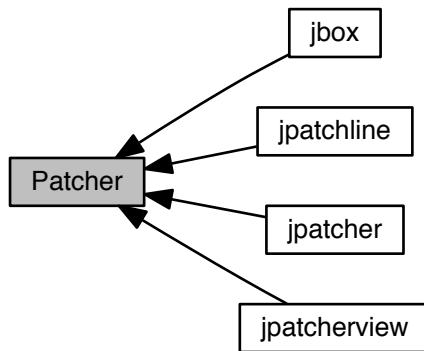
Returns

This function returns the error code [MAX_ERR_NONE](#) if successful, or one of the other error codes defined in "ext_obex.h" if unsuccessful.

36.40 Patcher

Max's patcher represents a graph of objects that communicate with messages.

Collaboration diagram for Patcher:



Modules

- [jpatcher](#)
The patcher.
- [jbox](#)
A box in the patcher.
- [jpatchline](#)
A patch cord.
- [jpatcherview](#)
A view of a patcher.

Data Structures

- [struct t_jbox](#)
The `t_jbox` struct provides the header for a Max user-interface object.

Typedefs

- [typedef t_object t_box](#)
A box.

Enumerations

- [enum { PI_DEEP, PI_REQUIREFIRSTIN, PI_WANTBOX }](#)
patcher iteration flags

Variables

- **BEGIN_USING_C_LINKAGE** `typedef t_object t_patcher`
A patcher.

36.40.1 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.

36.40.2 Typedef Documentation

36.40.2.1 `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_← typed()`, or by using [Attributes](#) accessors.

36.40.3 Enumeration Type Documentation

36.40.3.1 `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

36.40.4 Variable Documentation

36.40.4.1 **BEGIN_USING_C_LINKAGE** `typedef t_object t_patcher`

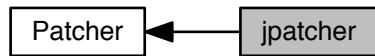
A patcher.

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.

36.41 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 unlocked background color.
- `t_max_err jpatcher_get_locked_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 unlocked background color.
- `t_max_err jpatcher_set_locked_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_box *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_object * jpatcher_get_hubholder (t_object *p)`
Given a patcher, return the patcher that will be responsible for holding the parameter hub.
- `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.

36.41.1 Detailed Description

The patcher.

36.41.2 Function Documentation

36.41.2.1 void [jpatcher_deleteobj](#) ([t_object](#) * *p*, [t_jbox](#) * *b*)

Delete an object that is in a patcher.

Parameters

<i>p</i>	The patcher.
<i>b</i>	The object box to delete.

36.41.2.2 [t_max_err](#) [jpatcher_get_bgcolor](#) ([t_object](#) * *p*, [t_jrgba](#) * *prgba*)

Retrieve a patcher's unlocked background color.

Parameters

<i>p</i>	The patcher to be queried.
<i>prgba</i>	The address of a valid t_jrgba struct that will be filled-in with the current patcher color values.

Returns

A Max error code.

36.41.2.3 char [jpatcher_get_bghidden](#) ([t_object](#) * *p*)

Determine whether a patcher's background layer is hidden.

Parameters

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

True if the background layer is hidden, otherwise false.

36.41.2.4 char [jpatcher_get_bglocked](#) ([t_object](#) * *p*)

Determine whether a patcher's background layer is locked.

Parameters

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

True if the background layer is locked, otherwise false.

36.41.2.5 t_object* jpatcher_get_box (t_object * *p*)

If a patcher is inside a box, return its box.

Parameters

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

A pointer to the box containing the patcher, otherwise NULL.

36.41.2.6 long jpatcher_get_count (t_object * *p*)

Determine the number of boxes in a patcher.

Parameters

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

The number of boxes in the patcher.

36.41.2.7 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.

36.41.2.8 long jpatcher_get_default_fontface (t_object * *p*)

Return the index of the default font face used for new objects in a patcher.

Parameters

<i>p</i>	A pointer to a patcher instance.
----------	----------------------------------

Returns

The index of the default font face used for new objects in a patcher.

36.41.2.9 t_symbol* jpatcher_get_default_fontname (t_object * *p*)

Return the name of the default font used for new objects in a patcher.

Parameters

<i>p</i>	A pointer to a patcher instance.
----------	----------------------------------

Returns

The name of the default font used for new objects in a patcher.

36.41.2.10 float jpatcher_get_default_fontsize (*t_object* * *p*)

Return the size of the default font used for new objects in a patcher.

Parameters

<i>p</i>	A pointer to a patcher instance.
----------	----------------------------------

Returns

The size of the default font used for new objects in a patcher.

36.41.2.11 *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

<i>p</i>	A pointer to a patcher instance.
<i>pr</i>	The address of valid <i>t_rect</i> whose values will be filled-in upon return.

Returns

A Max error code.

36.41.2.12 char jpatcher_get_dirty (*t_object* * *p*)

Determine whether a patcher's dirty bit has been set.

Parameters

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

True if the patcher is dirty, otherwise false.

36.41.2.13 *t_max_err* jpatcher_get_editing_bgcolor (*t_object* * *p*, *t_jrgba* * *prgba*)

Retrieve a patcher's editing background color.

Parameters

<i>p</i>	The patcher to be queried.
----------	----------------------------

<code>prgba</code>	The address of a valid <code>t_jrgba</code> struct that will be filled-in with the current patcher color values.
--------------------	--

Returns

A Max error code.

36.41.2.14 char jpatcher_get_fghidden (`t_object` * *p*)

Determine whether a patcher's foreground layer is hidden.

Parameters

<code>p</code>	The patcher to be queried.
----------------	----------------------------

Returns

True if the foreground layer is hidden, otherwise false.

36.41.2.15 `t_symbol`* jpatcher_get_filename (`t_object` * *p*)

Retrieve a patcher's file name.

Parameters

<code>p</code>	The patcher to be queried.
----------------	----------------------------

Returns

The patcher's file name.

36.41.2.16 `t_symbol`* jpatcher_get_filepath (`t_object` * *p*)

Retrieve a patcher's file path.

Parameters

<code>p</code>	The patcher to be queried.
----------------	----------------------------

Returns

The patcher's file path.

36.41.2.17 long jpatcher_get_fileversion (`t_object` * *p*)

Return the file version of the patcher.

Parameters

<code>p</code>	A pointer to the patcher whose version number is desired.
----------------	---

Returns

The file version number.

36.41.2.18 `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

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

The first jpatchline in a patcher.

36.41.2.19 *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

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

The first box in a patcher.

See also

[jbox_get_prevobject\(\)](#) [jbox_get_nextobject\(\)](#) [jpatcher_get_lastobject\(\)](#)

36.41.2.20 *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

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

The first view of a patcher.

36.41.2.21 *t_max_err* jpatcher_get_gridsize (*t_object *p, double *gridsizeX, double *gridsizeY*)

Retrieve a patcher's grid size.

Parameters

<i>p</i>	The patcher to be queried.
<i>gridsizeX</i>	The address of a double that will be set to the current horizontal grid spacing for the patcher.
<i>gridsizeY</i>	The address of a double that will be set to the current vertical grid spacing for the patcher.

Returns

A Max error code.

36.41.2.22 *t_object jpatcher_get_hubholder (*t_object *p*)**

Given a patcher, return the patcher that will be responsible for holding the parameter hub.

Parameters

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

The patcher's parameter hub holder.

36.41.2.23 *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

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

The last box in a patcher.

See also

[jbox_get_prevobject\(\)](#) [jbox_get_nextobject\(\)](#) [jpatcher_get_firstobject\(\)](#)

36.41.2.24 *t_max_err* jpatcher_get_locked_bgcolor (*t_object *p, t_jrgba *prgba*)

Retrieve a patcher's locked background color.

Parameters

<i>p</i>	The patcher to be queried.
<i>prgba</i>	The address of a valid t_jrgba struct that will be filled-in with the current patcher color values.

Returns

A Max error code.

36.41.2.25 *t_symbol jpatcher_get_name (*t_object *p*)**

Retrieve a patcher's name.

Parameters

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

The patcher's name.

36.41.2.26 *t_object jpatcher_get_parentpatcher (*t_object *p*)**

Given a patcher, return its parent patcher.

Parameters

<i>p</i>	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.

36.41.2.27 char jpatcher_get_presentation (*t_object* * *p*)

Determine whether a patcher is currently in presentation mode.

Parameters

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

True if the patcher is in presentation mode, otherwise false.

36.41.2.28 *t_max_err* jpatcher_get_rect (*t_object* * *p*, *t_rect* * *pr*)

Query a patcher to determine its location and size.

Parameters

<i>p</i>	A pointer to a patcher instance.
<i>pr</i>	The address of valid <i>t_rect</i> whose values will be filled-in upon return.

Returns

A Max error code.

36.41.2.29 *t_symbol jpatcher_get_title (*t_object* * *p*)**

Retrieve a patcher's title.

Parameters

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

The patcher's title.

36.41.2.30 *t_object jpatcher_get_toppatcher (*t_object* * *p*)**

Given a patcher, return the top-level patcher for the tree in which it exists.

Parameters

<i>p</i>	The patcher to be queried.
----------	----------------------------

Returns

The patcher's top-level parent patcher.

36.41.2.31 int jpatcher_is_patcher (t_object * *p*)

Determine if a [t_object](#)* is a patcher object.

Parameters

<i>p</i>	The object pointer to test.
----------	-----------------------------

Returns

Returns true if the object is a patcher, otherwise returns non-zero.

36.41.2.32 t_max_err jpatcher_set_bgcolor (t_object * *p*, t_jrgba * *prgba*)

Set a patcher's unlocked background color.

Parameters

<i>p</i>	The patcher to be queried.
<i>prgba</i>	The address of a t_jrgba struct containing the new color to use.

Returns

A Max error code.

36.41.2.33 t_max_err jpatcher_set_bghidden (t_object * *p*, char *c*)

Set whether a patcher's background layer is hidden.

Parameters

<i>p</i>	The patcher whose dirty bit will be set.
<i>c</i>	Pass true to hide the patcher's background layer, otherwise pass false.

Returns

A Max error code.

36.41.2.34 t_max_err jpatcher_set_bglocked (t_object * *p*, char *c*)

Set whether a patcher's background layer is locked.

Parameters

<i>p</i>	The patcher whose dirty bit will be set.
----------	--

<i>c</i>	Pass true to lock the patcher's background layer, otherwise pass false.
----------	---

Returns

A Max error code.

36.41.2.35 t_max_err jpatcher_set_defrect (t_object * *p*, t_rect * *pr*)

Set a patcher's default location and size.

Parameters

<i>p</i>	A pointer to a patcher instance.
<i>pr</i>	The address of a t_rect with the new position and size.

Returns

A Max error code.

36.41.2.36 t_max_err jpatcher_set_dirty (t_object * *p*, char *c*)

Set a patcher's dirty bit.

Parameters

<i>p</i>	The patcher whose dirty bit will be set.
<i>c</i>	The new value for the patcher's dirty bit (pass true or false).

Returns

A Max error code.

36.41.2.37 t_max_err jpatcher_set_editing_bgcolor (t_object * *p*, t_jrgba * *prgba*)

Set a patcher's editing background color.

Parameters

<i>p</i>	The patcher to be queried.
<i>prgba</i>	The address of a t_jrgba struct containing the new color to use.

Returns

A Max error code.

36.41.2.38 t_max_err jpatcher_set_fghidden (t_object * *p*, char *c*)

Set whether a patcher's foreground layer is hidden.

Parameters

<i>p</i>	The patcher whose dirty bit will be set.
<i>c</i>	Pass true to hide the patcher's foreground layer, otherwise pass false.

Returns

A Max error code.

36.41.2.39 t_max_err jpatcher_set_gridsize (t_object * p, double gridsizeX, double gridsizeY)

Set a patcher's grid size.

Parameters

<i>p</i>	The patcher to be queried.
<i>gridsizeX</i>	The new horizontal grid spacing for the patcher.
<i>gridsizeY</i>	The new vertical grid spacing for the patcher.

Returns

A Max error code.

36.41.2.40 t_max_err jpatcher_set_locked (t_object * p, char c)

Lock or unlock a patcher.

Parameters

<i>p</i>	The patcher whose locked state will be changed.
<i>c</i>	Pass true to lock a patcher, otherwise pass false.

Returns

A Max error code.

36.41.2.41 t_max_err jpatcher_set_locked_bgcolor (t_object * p, t_jrgba * prgba)

Set a patcher's locked background color.

Parameters

<i>p</i>	The patcher to be queried.
<i>prgba</i>	The address of a t_jrgba struct containing the new color to use.

Returns

A Max error code.

36.41.2.42 t_max_err jpatcher_set_presentation (t_object * p, char c)

Set a patcher to presentation mode.

Parameters

<i>p</i>	The patcher whose locked state will be changed.
<i>c</i>	Pass true to switch the patcher to presentation mode, otherwise pass false.

Returns

A Max error code.

36.41.2.43 t_max_err jpatcher_set_rect(t_object *p, t_rect *pr)

Set a patcher's location and size.

Parameters

<i>p</i>	A pointer to a patcher instance.
<i>pr</i>	The address of a t_rect with the new position and size.

Returns

A Max error code.

36.41.2.44 t_max_err jpatcher_set_title(t_object *p, t_symbol *ps)

Set a patcher's title.

Parameters

<i>p</i>	The patcher whose locked state will be changed.
<i>ps</i>	The new title for the patcher.

Returns

A Max error code.

36.41.2.45 t_symbol* jpatcher_uniqueboxname(t_object *p, t_symbol *classname)

Generate a unique name for a box in patcher.

Parameters

<i>p</i>	A pointer to a patcher instance.
<i>classname</i>	The name of an object's class.

Returns

The newly-generated unique name.

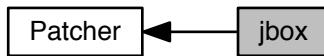
Remarks

This is the function used by pattr to assign names to objects in a patcher.

36.42 jbox

A box in the patcher.

Collaboration diagram for jbox:



Data Structures

- struct [t_jboxdrawparams](#)
The t_jboxdrawparams structure.

Macros

- #define JBOX_DRAWFIRSTIN
draw first inlet
- #define JBOX_NODRAWBOX
don't draw the frame
- #define JBOX_DRAWINLAST
draw inlets after update method
- #define JBOX_TRANSPARENT
don't make transparent unless you need it (for efficiency)
- #define JBOX_NOGROW
don't even draw grow thingie
- #define JBOX_GROWY
can grow in y direction by dragging
- #define JBOX_GROWBOTH
can grow independently in both x and y
- #define JBOX_IGNORELOCKCLICK
box should ignore a click if patcher is locked
- #define JBOX_HILITE
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
immediately set box into the background
- #define JBOX_NOFLOATINSPECTOR
no floating inspector window
- #define JBOX_TEXTFIELD
save/load text from textfield, unless JBOX_BINBUF flag is set
- #define JBOX_FIXWIDTH
give the box a textfield based fix-width (bfixwidth) method
- #define JBOX_FONTATTR
if you want font related attribute you must add this to [jbox_initclass\(\)](#)

- `#define JBOX_TEXTJUSTIFICATIONATTR`
give your object a textjustification attr to control textfield
- `#define JBOX_BINBUF`
save/load text from b_binbuf
- `#define JBOX_MOUSEDRAGDELTA`
hides mouse cursor in drag and sends mousedragdelta instead of mousedrag (for infinite scrolling like number)
- `#define JBOX_COLOR`
support the "color" method for color customization
- `#define JBOX_DRAWIOLOCKED`
draw inlets and outlets when locked (default is not to draw them)
- `#define JBOX_DRAWBACKGROUND`
set to have box bg filled in for you based on getdrawparams method or brgba attribute
- `#define JBOX_NOINSPECTFIRSTIN`
flag for objects such as bpatcher that have a different b_firstin,
- `#define JBOX_FOCUS`
more advanced focus support (passed to jbox_initclass() to add "nextfocus" and "prevfocus" attributes to the U← I object). Not implemented as of 2009-05-11
- `#define JBOX_BOXVIEW`
enable jboxview methods
- `#define JBOX_LEGACYCOLOR`
add undocumented color N message to objects from Max 4 that used it
- `#define JBOX_COPYLEGACYDEFAULT`
if there is a legacy default, copy it instead of the regular default
- `#define JBOX_NOLEGACYDEFAULT`
if there is a legacy default, don't copy any default

Enumerations

- enum { `JBOX_FONTFACE_REGULAR, JBOX_FONTFACE_BOLD, JBOX_FONTFACE_ITALIC, JBOX_F← ONTFACE_BOLDITALIC` }
actual numerical values of the b_fontface attribute; use jbox_fontface() to weight
- enum `HitTestResult` {
`HitNothing, HitBox, HitInlet, HitOutlet,`
`HitGrowBox, HitLine, HitLineLocked` }
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)`

- `t_max_err jbox_get_presentation_rect (t_object *box, t_rect *pr)`

Change the patching rect of a box.
- `t_max_err jbox_set_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 boxes 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)`

- **`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.

36.42.1 Detailed Description

A box in the patcher.

36.42.2 Macro Definition Documentation

36.42.2.1 `#define JBOX_NOINSPECTFIRSTIN`

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

36.42.3 Enumeration Type Documentation

36.42.3.1 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

36.42.3.2 `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
- `HitLineLocked`** a line in a locked patcher (for probing)

36.42.4 Function Documentation

36.42.4.1 `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

<i>b</i>	The address of your object's t_jbox member (which should be the first member of the object's struct).
----------	---

36.42.4.2 char* jbox_get_annotation (t_object * *bb*)

Retrieve a box's annotation string, if the user has given it an annotation.

Parameters

<i>bb</i>	The box to query.
-----------	-------------------

Returns

The user-created annotation string for a box, or NULL if no string exists.

36.42.4.3 char jbox_get_background (t_object * *b*)

Determine whether a box is located in the patcher's background layer.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

Zero if the object is in the foreground, otherwise non-zero.

36.42.4.4 char jbox_get_canhilite (t_object * *b*)

Retrieve a box flag value from a box.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

The value of the canhilite bit in the box's flags.

36.42.4.5 t_max_err jbox_get_color (t_object * *b*, t_jrgba * *prgba*)

Retrieve a box's 'color' attribute.

Parameters

<i>b</i>	The box to query.
<i>prgba</i>	The address of a valid t_rect whose values will be filled-in upon return.

Returns

A Max error code.

36.42.4.6 char jbox_get_drawfirstin (t_object * *b*)

Determine whether a box draws its first inlet.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

Zero if the inlet is not drawn, otherwise non-zero.

36.42.4.7 char jbox_get_drawinlast (t_object * *b*)

Retrieve a box flag value from a box.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

The value of the drawinlast bit in the box's flags.

36.42.4.8 t_symbol* jbox_get_fontname (t_object * *b*)

Retrieve a box's 'fontname' attribute.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

The font name.

36.42.4.9 double jbox_get_fontsize (t_object * *b*)

Retrieve a box's 'fontsize' attribute.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

The font size in points.

36.42.4.10 char jbox_get_growboth (t_object * *b*)

Retrieve a box flag value from a box.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

The value of the growboth bit in the box's flags.

36.42.4.11 char jbox_get_growy (t_object * *b*)

Retrieve a box flag value from a box.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

The value of the growy bit in the box's flags.

36.42.4.12 char jbox_get_hidden (t_object * *b*)

Retrieve a box's 'hidden' attribute.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

True if the box is hidden, otherwise false.

36.42.4.13 t_symbol* jbox_get_hint (t_object * *b*)

Retrieve a box's hint text as a symbol.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

The box's hint text.

36.42.4.14 char* jbox_get_hintstring (t_object * *bb*)

Retrieve a box's hint text as a C-string.

Parameters

<i>bb</i>	The box to query.
-----------	-------------------

Returns

The box's hint text.

36.42.4.15 t_symbol* jbox_get_id (t_object * *b*)

Retrieve a boxes unique id.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

The unique id of the object. This is a symbol that is referenced, for example, by patchlines.

36.42.4.16 `char jbox_get_ignoreclick (t_object * b)`

Determine whether a box ignores clicks.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

Zero if the object responds to clicks, otherwise non-zero.

36.42.4.17 t_symbol* jbox_get_maxclass (t_object * *b*)

Retrieve the name of the class of the box's object.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

The name of the class of the box's object.

36.42.4.18 t_object* jbox_get_nextobject (t_object * *b*)

The next box in the patcher's (linked) list of boxes.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

The next box in the list.

36.42.4.19 char jbox_get_nogrow (t_object * *b*)

Retrieve a box flag value from a box.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

The value of the nogrow bit in the box's flags.

36.42.4.20 t_object* jbox_get_object (t_object * *b*)

Retrieve a pointer to the box's object.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

A pointer to the box's object.

36.42.4.21 `char jbox_get_outline(t_object *b)`

Determine whether a box draws an outline.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

Zero if the outline is not drawn, otherwise non-zero.

36.42.4.22 t_object* jbox_get_patcher(t_object * *b*)

Retrieve a box's patcher.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

If the box has a patcher, the patcher's pointer is returned. Otherwise NULL is returned.

36.42.4.23 t_max_err jbox_get_patchning_position(t_object * *box*, t_pt * *pos*)

Fetch the position of a box for the patching view.

Parameters

<i>box</i>	The box whose position will be retrieved.
<i>pos</i>	The address of a valid t_pt whose x and y values will be filled in.

Returns

A Max error code.

36.42.4.24 t_max_err jbox_get_patchning_rect(t_object * *box*, t_rect * *pr*)

Retrieve the patching rect of a box.

Parameters

<i>box</i>	The box whose rect values will be retrieved.
<i>pr</i>	The address of a valid t_rect whose values will be filled in.

Returns

A Max error code.

36.42.4.25 t_max_err jbox_get_patchning_size(t_object * *box*, t_size * *size*)

Fetch the size of a box for the patching view.

Parameters

<i>box</i>	The box whose size will be retrieved.
<i>size</i>	The address of a valid t_size whose width and height values will be filled in.

Returns

A Max error code.

36.42.4.26 char jbox_get_presentation (t_object * *b*)

Determine if a box is included in the presentation view.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

Non-zero if in presentation mode, otherwise zero.

36.42.4.27 t_max_err jbox_get_presentation_position (t_object * *box*, t_pt * *pos*)

Fetch the position of a box for the presentation view.

Parameters

<i>box</i>	The box whose position will be retrieved.
<i>pos</i>	The address of a valid t_pt whose x and y values will be filled in.

Returns

A Max error code.

36.42.4.28 t_max_err jbox_get_presentation_rect (t_object * *box*, t_rect * *pr*)

Retrieve the presentation rect of a box.

Parameters

<i>box</i>	The box whose rect values will be retrieved.
<i>pr</i>	The address of a valid t_rect whose values will be filled in.

Returns

A Max error code.

36.42.4.29 t_max_err jbox_get_presentation_size (t_object * *box*, t_size * *size*)

Fetch the size of a box for the presentation view.

Parameters

<i>box</i>	The box whose size will be retrieved.
<i>size</i>	The address of a valid t_size whose width and height values will be filled in.

Returns

A Max error code.

36.42.4.30 `t_object* jbox_get_prevobject(t_object * b)`

The previous box in the patcher's (linked) list of boxes.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

The next box in the list.

36.42.4.31 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

<i>box</i>	The box whose rect will be fetched.
<i>which</i>	The name of the rect attribute to be fetched, for example <code>_sym_presentation_rect</code> or <code>_sym_patchrect</code> .
<i>pr</i>	The address of a valid <code>t_rect</code> whose members will be filled in by this function.

Returns

A Max error code.

36.42.4.32 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

<i>box</i>	The box whose rect will be fetched.
<i>patcherview</i>	A patcherview in which the box exists.
<i>rect</i>	The address of a valid <code>t_rect</code> whose members will be filled in by this function.

Returns

A Max error code.

36.42.4.33 t_object* jbox_get_textfield (t_object * *b*)

Retrieve a pointer to a box's textfield.

Parameters

<i>b</i>	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.

36.42.4.34 t_symbol* jbox_get_varname (t_object * *b*)

Retrieve a box's scripting name.

Parameters

<i>b</i>	The box to query.
----------	-------------------

Returns

The box's scripting name.

36.42.4.35 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

<i>b</i>	The address of your UI object's t_jbox member (which should be the first member of the object's struct).
<i>flags</i>	Flags to set the box's behavior, such as JBOX_NODRAWBOX .
<i>argc</i>	The count of atoms in the argv parameter.
<i>argv</i>	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.

36.42.4.36 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

<i>b</i>	The address of your object's t_jbox member.
<i>s</i>	The name of the send object.
<i>msg</i>	The notification name.
<i>sender</i>	The sending object's address.
<i>data</i>	A pointer to some data passed to the box's notify method.

Returns

A Max error code.

36.42.4.37 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

<i>b</i>	The address of your object's t_jbox member.
----------	---

36.42.4.38 void jbox_redraw (*t_jbox * b*)

Request that your object/box be re-drawn by Max.

Parameters

<i>b</i>	The address of your object's t_jbox member.
----------	---

36.42.4.39 void jbox_set_annotation ([t_object](#) * *bb*, char * *s*)

Set a box's annotation string.

Parameters

<i>bb</i>	The box to query.
<i>s</i>	The annotation string for the box.

Returns

A Max error code.

36.42.4.40 [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

<i>b</i>	The box to query.
<i>c</i>	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.

36.42.4.41 [t_max_err](#) jbox_set_color ([t_object](#) * *b*, [t_jrgba](#) * *prgba*)

Set a box's 'color' attribute.

Parameters

<i>b</i>	The box to query.
<i>prgba</i>	The address of a t_rect containing the desired color for the box/object.

Returns

A Max error code.

36.42.4.42 [t_max_err](#) jbox_set_fontname ([t_object](#) * *b*, [t_symbol](#) * *ps*)

Set a box's 'fontname' attribute.

Parameters

<i>b</i>	The box to query.
<i>ps</i>	The font name. Note that the font name may be case-sensitive.

Returns

A Max error code.

36.42.4.43 `t_max_err jbox_set_fontsize(t_object *b, double d)`

Set a box's 'fontsize' attribute.

Parameters

<i>b</i>	The box to query.
<i>d</i>	The fontsize in points.

Returns

A Max error code.

36.42.4.44 t_max_err jbox_set_hidden (t_object * *b*, char *c*)

Set a box's 'hidden' attribute.

Parameters

<i>b</i>	The box to query.
<i>c</i>	Set to true to hide the box, otherwise false.

Returns

A Max error code.

36.42.4.45 t_max_err jbox_set_hint (t_object * *b*, t_symbol * *s*)

Set a box's hint text using a symbol.

Parameters

<i>b</i>	The box to query.
<i>s</i>	The new text to use for the box's hint.

Returns

A Max error code.

36.42.4.46 void jbox_set_hintstring (t_object * *bb*, char * *s*)

Set a box's hint text using a C-string.

Parameters

<i>bb</i>	The box to query.
<i>s</i>	The new text to use for the box's hint.

Returns

A Max error code.

36.42.4.47 t_max_err jbox_set_ignoreclick (t_object * *b*, char *c*)

Set whether a box ignores clicks.

Parameters

<i>b</i>	The box to query.
<i>c</i>	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.

36.42.4.48 *t_max_err jbox_set_outline(t_object *b, char c)*

Set whether a box draws an outline.

Parameters

<i>b</i>	The box to query.
<i>c</i>	Pass zero to hide the outline, or non-zero to indicate that the box should draw the outline.

Returns

A Max error code.

36.42.4.49 *t_max_err jbox_set_patching_position(t_object *box, t_pt *pos)*

Set the position of a box for the patching view.

Parameters

<i>box</i>	The box whose positon will be changed.
<i>pos</i>	The address of a t_pt with the new x and y values.

Returns

A Max error code.

36.42.4.50 *t_max_err jbox_set_patching_rect(t_object *box, t_rect *pr)*

Change the patching rect of a box.

Parameters

<i>box</i>	The box whose rect will be changed.
<i>pr</i>	The address of a t_rect with the new rect values.

Returns

A Max error code.

36.42.4.51 *t_max_err jbox_set_patching_size(t_object *box, t_size *size)*

Set the size of a box for the patching view.

Parameters

<i>box</i>	The box whose size will be changed.
<i>size</i>	The address of a t_size with the new width and height values.

Returns

A Max error code.

36.42.4.52 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

<i>box</i>	The box whose position will be changed.
<i>pos</i>	The address of a t_pt with the new x and y values.

Returns

A Max error code.

36.42.4.53 t_max_err jbox_set_presentation (t_object * *b*, char *c*)

Determine if a box is included in the presentation view.

Parameters

<i>b</i>	The box to query.
<i>c</i>	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.

36.42.4.54 t_max_err jbox_set_presentation_position (t_object * *box*, t_pt * *pos*)

Set the position of a box for the presentation view.

Parameters

<i>box</i>	The box whose rect will be changed.
<i>pos</i>	The address of a t_pt with the new x and y values.

Returns

A Max error code.

36.42.4.55 t_max_err jbox_set_presentation_rect (t_object * *box*, t_rect * *pr*)

Change the presentation rect of a box.

Parameters

<i>box</i>	The box whose rect will be changed.
<i>pr</i>	The address of a t_rect with the new rect values.

Returns

A Max error code.

36.42.4.56 t_max_err jbox_set_presentation_size (t_object *box, t_size *size)

Set the size of a box for the presentation view.

Parameters

<i>box</i>	The box whose size will be changed.
<i>size</i>	The address of a t_size with the new width and height values.

Returns

A Max error code.

36.42.4.57 t_max_err jbox_set_rect (t_object *box, t_rect *pr)

Set both the presentation rect and the patching rect.

Parameters

<i>box</i>	The box whose rect will be changed.
<i>pr</i>	The address of a t_rect with the new rect values.

Returns

A Max error code.

36.42.4.58 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

<i>box</i>	The box whose rect will be changed.
<i>which</i>	The name of the rect attribute to be changed, for example <code>_sym_presentation_rect</code> or <code>_sym_patchrect</code> .
<i>pr</i>	The address of a valid t_rect that will replace the current values used by the box.

Returns

A Max error code.

36.42.4.59 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

<i>box</i>	The box whose rect will be changed.
<i>patcherview</i>	A patcherview in which the box exists.
<i>rect</i>	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.

36.42.4.60 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

<i>box</i>	The box whose size will be changed.
<i>size</i>	The address of a t_size with the new size values.

Returns

A Max error code.

36.42.4.61 t_max_err jbox_set_varname (t_object *b, t_symbol *ps)

Set a box's scripting name.

Parameters

<i>b</i>	The box to query.
<i>ps</i>	The new scripting name for the box.

Returns

A Max error code.

36.43 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.

36.43.1 Detailed Description

A patch cord.

36.43.2 Function Documentation

36.43.2.1 `t_object* jpatchline_get_box1 (t_object * l)`

Return the object box from which a patchline originates.

Parameters

/	A pointer to the patchline's instance.
---	--

Returns

The object box from which the patchline originates.

36.43.2.2 t_object* jpatchline_get_box2(t_object * I)

Return the destination object box for a patchline.

Parameters

/	A pointer to the patchline's instance.
---	--

Returns

The destination object box for a patchline.

36.43.2.3 t_max_err jpatchline_get_color(t_object * I, t_jrgba * prgba)

Get the color of a patch line.

Parameters

/	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.

36.43.2.4 t_max_err jpatchline_get_endpoint(t_object * I, double * x, double * y)

Retrieve a patchline's ending point.

Parameters

/	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.

36.43.2.5 char jpatchline_get_hidden(t_object * I)

Determine if a patch line is hidden.

Parameters

/	A patchline instance.
---	-----------------------

Returns

Zero if the patchline is visible, non-zero if it is hidden.

36.43.2.6 long jpatchline_get_inletnum (t_object * *l*)

Return the inlet number of the destination object box to which a patchline is connected.

Parameters

/	A pointer to the patchline's instance.
---	--

Returns

The inlet number.

36.43.2.7 t_object* jpatchline_get_nextline (t_object * *b*)

Given a patchline, traverse to the next patchline in the (linked) list.

Parameters

<i>b</i>	A patchline instance.
----------	-----------------------

Returns

The next patchline. If the current patchline is at the end (tail) of the list, then NULL is returned.

36.43.2.8 long jpatchline_get_nummidpoints (t_object * *l*)

Determine the number of midpoints (segments) in a patchline.

Parameters

/	A pointer to the patchline's instance.
---	--

Returns

The number of midpoints in the patchline.

36.43.2.9 long jpatchline_get_outletnum (t_object * *l*)

Return the outlet number of the originating object box from which a patchline begins.

Parameters

/	A pointer to the patchline's instance.
---	--

Returns

The outlet number.

36.43.2.10 `t_max_err jpatchline_get_startpoint(t_object *l, double *x, double *y)`

Retrieve a patchline's starting point.

Parameters

<i>l</i>	A pointer to the patchline's instance.
<i>x</i>	The address of a variable to hold the x-coordinate of the starting point's position upon return.
<i>y</i>	The address of a variable to hold the y-coordinate of the starting point's position upon return.

Returns

A Max error code.

36.43.2.11 t_max_err jpatchline_set_color(t_object *l, t_jrgba *prgba)

Set the color of a patch line.

Parameters

<i>l</i>	A patchline instance.
<i>prgba</i>	The address of a valid t_jrgba struct containing the color to use.

Returns

An error code.

36.43.2.12 t_max_err jpatchline_set_hidden(t_object *l, char c)

Set a patchline's visibility.

Parameters

<i>l</i>	A patchline instance.
<i>c</i>	Pass 0 to make a patchline visible, or non-zero to hide it.

Returns

An error code.

36.44 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.
- `void patcherview_canvas_to_screen (t_object *pv, double cx, double cy, long *sx, long *sy)`
Convert the point cx, cy in canvas coordinates to screen coordinates.
- `void patcherview_screen_to_canvas (t_object *pv, long sx, long sy, double *cx, double *cy)`
Convert the point cx, cy in screen coordinates to canvas coordinates.
- `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).

36.44.1 Detailed Description

A view of a patcher.

36.44.2 Function Documentation

36.44.2.1 void patcherview_canvas_to_screen (*t_object* * *pv*, double *cx*, double *cy*, long * *sx*, long * *sy*)

Convert the point *cx*, *cy* in canvas coordinates to screen coordinates.

Parameters

<i>pv</i>	The patcherview instance the canvas coords are relative to.
<i>cx</i>	The x dimension of the canvas coordinate relative to the patcherview.
<i>cy</i>	The y dimension of the canvas coordinate relative to the patcherview.
<i>sx</i>	A pointer to a long to receive the screen coordinate x dimension.
<i>sy</i>	A pointer to a long to receive the screen coordinate y dimension.

36.44.2.2 *t_object** patcherview_findpatcherview (int *x*, int *y*)

Find a patcherview at the given screen coords.

Parameters

<i>x</i>	The horizontal coordinate at which to find a patcherview.
<i>y</i>	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.

36.44.2.3 *t_object** patcherview_get_jgraphics (*t_object* * *pv*)

Given a patcherview, return the [t_jgraphics](#) context for that view.

Parameters

<i>pv</i>	The patcherview instance.
-----------	---------------------------

Returns

The [t_jgraphics](#) context for the view.

36.44.2.4 char patcherview_get_locked (*t_object* * *p*)

Find out if a patcherview is locked.

Parameters

<i>p</i>	The patcherview instance whose attribute value will be fetched.
----------	---

Returns

Returns 0 if unlocked, otherwise returns non-zero.

36.44.2.5 `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

<i>pv</i>	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.

36.44.2.6 `t_object* patcherview_get_patcher (t_object *pv)`

Given a patcherview, return its patcher.

Parameters

<i>pv</i>	The patcherview instance for which to fetch the patcher.
-----------	--

Returns

The patcher.

36.44.2.7 `char patcherview_get_presentation (t_object *pv)`

Find out if a patcherview is a presentation view.

Parameters

<i>pv</i>	The patcherview instance whose attribute value will be fetched.
-----------	---

Returns

Returns 0 if the view is not a presentation view, otherwise returns non-zero.

36.44.2.8 `t_max_err patcherview_get_rect (t_object *pv, t_rect *pr)`

Get the value of the rect attribute for a patcherview.

Parameters

<i>pv</i>	The patcherview instance whose attribute value will be fetched.
<i>pr</i>	The address of a valid t_rect struct, whose contents will be filled upon return.

Returns

An error code.

36.44.2.9 `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

<i>pv</i>	The patcherview instance whose top view you want to get.
-----------	--

Returns

The top patcherview.

36.44.2.10 char patcherview_get_visible (*t_object* * *pv*)

Query a patcherview to determine whether it is visible.

Parameters

<i>pv</i>	The patcherview instance to query.
-----------	------------------------------------

Returns

Returns zero if the patcherview is invisible, otherwise returns non-zero.

36.44.2.11 double patcherview_get_zoomfactor (*t_object* * *pv*)

Fetch the zoom-factor of a patcherview.

Parameters

<i>pv</i>	The patcherview instance whose attribute value will be fetched.
-----------	---

Returns

The factor by which the view is zoomed.

36.44.2.12 void patcherview_screen_to_canvas (*t_object* * *pv*, long *sx*, long *sy*, double * *cx*, double * *cy*)

Convert the point *cx*, *cy* in canvas coordinates to screen coordinates.

Parameters

<i>pv</i>	The patcherview instance the canvas coords are relative to.
<i>sx</i>	The screen position x coordinate.
<i>sy</i>	The screen position y coordinate
<i>cx</i>	A pointer to a double to receive the canvas coordinate for the given screen x position.
<i>cy</i>	A pointer to a double to receive the canvas coordinate for the given screen y position.

36.44.2.13 *t_max_err* patcherview_set_locked (*t_object* * *p*, char *c*)

Lock or unlock a patcherview.

Parameters

<i>p</i>	The patcherview instance whose attribute value will be set.
<i>c</i>	Set this value to zero to unlock the patcherview, otherwise pass a non-zero value.

Returns

An error code.

36.44.2.14 t_max_err patcherview_set_presentation (t_object * p, char c)

Set whether or not a patcherview is a presentation view.

Parameters

<i>p</i>	The patcherview instance whose attribute value will be set.
<i>c</i>	Set this value to non-zero to make the patcherview a presentation view, otherwise pass zero.

Returns

An error code.

36.44.2.15 t_max_err patcherview_set_rect(t_object *pv, t_rect *pr)

Set the value of the rect attribute for a patcherview.

Parameters

<i>pv</i>	The patcherview instance whose attribute value will be set.
<i>pr</i>	The address of a valid t_rect struct.

Returns

An error code.

36.44.2.16 t_max_err patcherview_set_visible(t_object *pv, char c)

Set the 'visible' attribute of a patcherview.

Parameters

<i>pv</i>	The patcherview instance whose attribute value will be set.
<i>c</i>	Whether or not the patcherview should be made visible.

Returns

An error code.

36.44.2.17 t_max_err patcherview_set_zoomfactor(t_object *pv, double d)

Set the zoom-factor of a patcherview.

Parameters

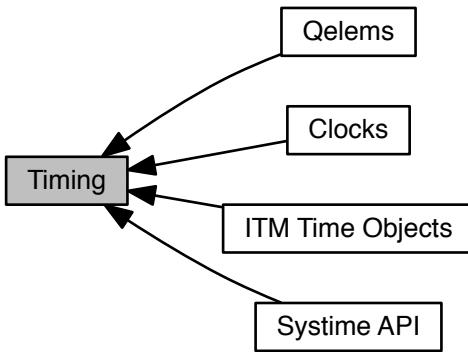
<i>pv</i>	The patcherview instance whose attribute value will be set.
<i>d</i>	The zoom-factor at which the patcherview should display the patcher.

Returns

An error code.

36.45 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.

36.45.1 Detailed Description

36.46 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 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 gettimeofday (void)`
Find out the current logical time of the scheduler in milliseconds.
- `double gettimeofday_forobject (t_object *x)`
Find the correct scheduler for the object and return the current time in milliseconds.
- `void *scheduler_new (void)`

- `void *scheduler_set (void *x)`
Create a new local scheduler.
`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.

36.46.1 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.

36.46.2 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);
}
```

36.46.3 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.

36.46.3.1 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);
    }
}

```

36.46.4 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.

36.46.5 Function Documentation

36.46.5.1 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()`

36.46.5.2 void `clock_fdelay(void * c, double time)`

Schedule the execution of a Clock using a floating-point argument.

`clock_delay()` sets a clock to go off at a certain number of milliseconds from the current logical time.

Parameters

<i>c</i>	Clock to schedule.
<i>time</i>	Delay, in milliseconds, before the Clock will execute.

See also

[clock_delay\(\)](#)

36.46.5.3 void `clock_gettime(double * time)`

Find out the current logical time of the scheduler in milliseconds as a floating-point number.

Parameters

<i>time</i>	Returns the current time.
-------------	---------------------------

See also

[gettime\(\)](#)
[setclock_gettime\(\)](#)
[setclock_gettime\(\)](#)

36.46.5.4 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

<i>obj</i>	Argument that will be passed to clock function <i>fn</i> when it is called. This will almost always be a pointer to your object.
<i>fn</i>	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.

36.46.5.5 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.
---	------------------

36.46.5.6 long gettime(void)

Find out the current logical time of the scheduler in milliseconds.

Returns

Returns the current time.

See also

[clock_gettime\(\)](#)

36.46.5.7 double gettime_forobject(t_object *x)

Find the correct scheduler for the object and return the current time in milliseconds.

Returns

Returns the current time.

See also

[clock_gettime\(\)](#)

36.46.5.8 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](#)

36.46.5.9 void* scheduler_get()

Get the currently set scheduler.

Returns

This routine returns a pointer to the current scheduler,

See also

[Creating Schedulers](#)

36.46.5.10 void scheduler_gettime(void *x, double *time)

Retrieve the current time of the selected scheduler.

Parameters

<i>x</i>	The scheduler to query.
<i>time</i>	The current time of the selected scheduler.

See also[Creating Schedulers](#)**36.46.5.11 void* scheduler_new (void)**

Create a new local scheduler.

Returns

A pointer to the newly created scheduler.

See also[Creating Schedulers](#)**36.46.5.12 void scheduler_run (void * x, double until)**

Run scheduler events to a selected time.

Parameters

<i>x</i>	The scheduler to advance.
<i>until</i>	The ending time for this run (in milliseconds).

See also[Creating Schedulers](#)**36.46.5.13 void* scheduler_set (void * x)**

Make a scheduler current, so that future related calls (such as [clock_delay\(\)](#)) will affect the appropriate scheduler.

Parameters

<i>x</i>	The scheduler to make current.
----------	--------------------------------

Returns

This routine returns a pointer to the previously current scheduler, saved and restored when local scheduling is complete.

See also[Creating Schedulers](#)**36.46.5.14 void scheduler_settime (void * x, double time)**

Set the current time of the scheduler.

Parameters

<i>x</i>	The scheduler to set.
<i>time</i>	The new current time for the selected scheduler (in milliseconds).

See also

[Creating Schedulers](#)

36.46.5.15 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

<i>x</i>	The scheduler to affect.
<i>amount</i>	Number of milliseconds to shift by.

See also

[Creating Schedulers](#)

36.46.5.16 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

<i>x</i>	A setclock object to be used for scheduling this clock.
<i>c</i>	Clock object containing the function to be executed.
<i>time</i>	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\(\)](#)

36.46.5.17 void setclock_fdelay(t_object * *s*, void * *c*, double *time*)

Schedule a Clock on a scheduler, using a floating-point time argument.

Parameters

<i>s</i>	A setclock object to be used for scheduling this clock.
<i>c</i>	Clock object containing the function to be executed.
<i>time</i>	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\(\)](#)

36.46.5.18 void **setclock_gettime** (**t_object** * *s*, double * *time*)

Find out the current time value of a **setclock** object in floating-point milliseconds.

Parameters

<i>s</i>	A setclock object.
<i>time</i>	The current time in milliseconds.

See also

[Scheduling with setclock Objects](#)
[setclock_gettime\(\)](#)

36.46.5.19 long setclock_gettime (t_object * x)

Find out the current time value of a setclock object.

Parameters

<i>x</i>	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\(\)](#)

36.46.5.20 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

<i>x</i>	The setclock object that was used to schedule this clock. If 0, the clock is unscheduled from the internal millisecond scheduler.
<i>c</i>	Clock object to be removed from the scheduler.

See also

[Scheduling with setclock Objects](#)

36.46.5.21 double systimer_gettime (void)

While most Max 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.

36.47 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.

36.47.1 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.

36.47.2 Function Documentation

36.47.2.1 void qelem_free (void * *x*)

Free a Qelem object created with [qelem_new\(\)](#).

Typically this will be in your object's free function.

Parameters

<i>x</i>	The Qelem to destroy.
----------	-----------------------

36.47.2.2 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

<i>x</i>	The Qelem whose function will be executed in the main thread.
----------	---

36.47.2.3 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

<i>obj</i>	Argument to be passed to function fun when the Qelem executes. Normally a pointer to your object.
<i>fn</i>	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.

36.47.2.4 void qelem_set (void * *q*)

Cause a Qelem to execute.

Parameters

<i>q</i>	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.

36.47.2.5 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

<i>q</i>	The Qelem whose execution you wish to cancel.
----------	---

36.48 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`, `SYSDATEFORMAT_FLAGS_MEDIUM`, `SYSDATEFORMAT_FLAGS_LONG` }

Flags for the `sysdateformat_formatdatetime()` function.

Functions

- `t_uint32 systime_ticks (void)`
Find out the operating system's time in ticks.
- `t_uint32 systime_ms (void)`
Find out the operating system's time in milliseconds.
- `t_int64 systime_datetime_milliseconds (void)`
Find out the current date/time as number of ms since 1/1/1970.
- `void systime_datetime (t_datetime *d)`
Find out the operating system's date and time.
- `t_ptr_uint systime_seconds (void)`
Find out the operating system's time in seconds.
- `void systime_secondstodate (t_ptr_uint secs, t_datetime *d)`
Convert a time in seconds into a `t_datetime` representation.
- `t_ptr_uint 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`.

36.48.1 Detailed Description

The Systime API provides the means of getting the system time, instead of the scheduler time as you would with functions like `gettime()`.

36.48.2 Enumeration Type Documentation

36.48.2.1 enum e_max_dateflags

Flags for the `sysdateformat_formatdatetime()` function.

Enumerator

`SYSDATEFORMAT_FLAGS_SHORT` short

`SYSDATEFORMAT_FLAGS_MEDIUM` medium

`SYSDATEFORMAT_FLAGS_LONG` long

36.48.3 Function Documentation

36.48.3.1 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

<code>d</code>	The address of a <code>t_datetime</code> to fill.
<code>dateflags</code>	One of the values defined in <code>e_max_dateflags</code> .
<code>timeflags</code>	Currently unused. Pass 0.
<code>s</code>	An already allocated string to hold the human friendly result.
<code>buflen</code>	The number of characters allocated to the string <code>s</code> .

36.48.3.2 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

<code>strf</code>	A string containing the datetime.
<code>d</code>	The address of a <code>t_datetime</code> to fill.

36.48.3.3 void systime_datetime (`t_datetime * d`)

Find out the operating system's date and time.

Parameters

<code>d</code>	Returns the system's date and time in a <code>t_datetime</code> data structure.
----------------	---

36.48.3.4 `t_int64 systime_datetime_milliseconds (void)`

Find out the current date/time as number of ms since 1/1/1970.

Returns

the number of milliseconds since 1/1/1970.

36.48.3.5 `t_ptr_uint systime_datetoseconds (t_datetime * d)`

Convert a `t_datetime` representation of time into seconds.

Parameters

<i>d</i>	The address of a t_datetime that contains a valid period of time.
----------	---

Returns

The number of seconds represented by *d*.

36.48.3.6 t_uint32 systime_ms (void)

Find out the operating system's time in milliseconds.

Note that this is approximately the number of milliseconds since the OS was started up.

Returns

the system time in milliseconds.

36.48.3.7 t_ptr_uint systime_seconds (void)

Find out the operating system's time in seconds.

Returns

the system time in seconds.

36.48.3.8 void systime_secondstodate (t_ptr_uint secs, t_datetime * d)

Convert a time in seconds into a [t_datetime](#) representation.

Parameters

<i>secs</i>	A number of seconds to be represented as a t_datetime .
<i>d</i>	The address of a t_datetime that will be filled with the converted value.

36.48.3.9 t_uint32 systime_ticks (void)

Find out the operating system's time in ticks.

Returns

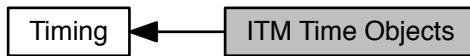
the system time in ticks.

Referenced by `jit_rand_setseed()`.

36.49 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, TIME_FLAGS_TICKSONLY, TIME_FLAGS_FIXEDONLY, TIME_FLAGS_LOOKAHEAD,
 TIME_FLAGS_USECLOCK, TIME_FLAGS_USEQELEM, TIME_FLAGS_FIXED, TIME_FLAGS_PERMANENT,
 TIME_FLAGS_TRANSPORT, TIME_FLAGS_EVENTLIST, TIME_FLAGS_CHECKSCHEDULE, TIME_FLAGS_LISTENTICKS,
 TIME_FLAGS_NOUNITS, TIME_FLAGS_BBUSOURCE, TIME_FLAGS_POSITIVE }

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 itm in ticks.
- void **itm_dump** (**t_itm** *x)
Print diagnostic information about an itm object to the Max window.
- void **itm_settimesignature** (**t_itm** *x, long num, long denom, long flags)
Set an itm object's current time signature.
- void **itm_gettimesignature** (**t_itm** *x, long *num, long *denom)
Query an itm object for its current time signature.
- void **itm_pause** (**t_itm** *x)
Pause the passage of time for an itm object.
- void **itm_resume** (**t_itm** *x)
Start the passage of time for an itm object, from it's current location.
- long **itm_getstate** (**t_itm** *x)

- **void `itm_setresolution` (double res)**

Set the number of ticks-per-quarter-note globally for the itm system.
- **double `itm_getresolution` (void)**

Get the number of ticks-per-quarter-note globally from the itm system.
- **`t_symbol * itm_getname (t_itm *x)`**

Given an itm 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 itm 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 itm 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 itm 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 itm object.
- **void `itm_barbeatunitstoticks (t_itm *x, long bars, long beats, double *ticks, char position)`**

Convert a time value in bbu to the equivalent value in ticks, given the context of a specified itm object.
- **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 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, const char *attrname, const char *atrrlabel, 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.

36.49.1 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).

36.49.2 Enumeration Type Documentation

36.49.2.1 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 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

36.49.3 Function Documentation

36.49.3.1 void class_time_addattr (**t_class** * c, const char * attrname, const char * attrlabel, long flags)

Create an attribute permitting a time object to be changed in a user-friendly way.

Parameters

<i>c</i>	Class being initialized.
<i>attrname</i>	Name of the attribute associated with the time object.
<i>attrlabel</i>	Descriptive label for the attribute (appears in the inspector)
<i>flags</i>	Options, see "Flags that determine time object behavior" above

36.49.3.2 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

<i>x</i>	An itm object.
<i>bars</i>	The measure number of the location/position.
<i>beats</i>	The beat number of the location/position.
<i>units</i>	The number of ticks past the beat of the location/position.
<i>ticks</i>	The address of a variable to hold the number of ticks upon return.
<i>position</i>	Set this parameter to TIME_FLAGS_LOCATION or to zero (for position mode).

36.49.3.3 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

<i>x</i>	The itm object.
----------	-----------------

36.49.3.4 void itm_dump (*t_itm* * *x*)

Print diagnostic information about an itm object to the Max window.

Parameters

<i>x</i>	The itm object.
----------	-----------------

36.49.3.5 void* itm_getglobal (void)

Return the global (default / unnamed) itm object.

Returns

The global [t_itm](#) object.

36.49.3.6 *t_symbol itm_getname (*t_itm* * *x*)**

Given an itm object, return its name.

Parameters

x	The item object.
---	------------------

Returns

The name of the item.

36.49.3.7 void* item_getnamed (t_symbol * s, void * scheduler, t_symbol * defaultclocksourcename, long create)

Return a named item object.

Parameters

s	The name of the item to return.
<i>scheduler</i>	
<i>defaultclock-sourcename</i>	
<i>create</i>	If non-zero, then create this named item should it not already exist.

Returns

The global [t_item](#) object.

36.49.3.8 double item_getresolution (void)

Get the number of ticks-per-quarter-note globally from the item system.

Returns

The number of ticks-per-quarter-note.

See also

[item_setresolution\(\)](#)

36.49.3.9 long item_getstate (t_item * x)

Find out if time is currently progressing for a given item object.

Parameters

x	The item object.
---	------------------

Returns

Returns non-zero if the time is running, or zero if it is paused.

See also

[item_pause\(\)](#)
[item_resume\(\)](#)

36.49.3.10 double item_getticks (t_item * x)

Report the current time of the item in ticks.

You can use functions such as [item_ticksbarbeatunits\(\)](#) or [item_tickstoms\(\)](#) to convert to a different representation of the time.

Parameters

x	The itm object.
---	-----------------

Returns

The current time in ticks.

36.49.3.11 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.

36.49.3.12 void itm_gettimesignature (t_itm * x, long * num, long * denom)

Query an itm object for its current time signature.

Parameters

x	The itm 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.

36.49.3.13 long itm_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).

36.49.3.14 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 itm object.

Parameters

<i>x</i>	An itm object.
<i>ms</i>	A time specified in ms.

Returns

The time specified in samples.

36.49.3.15 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 itm object.

Parameters

<i>x</i>	An itm object.
<i>ms</i>	A time specified in ms.

Returns

The time specified in ticks.

36.49.3.16 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

<i>x</i>	The itm object.
----------	-----------------

36.49.3.17 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

<i>x</i>	The itm object.
----------	-----------------

36.49.3.18 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

<i>x</i>	The itm object.
----------	-----------------

36.49.3.19 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

<i>x</i>	An itm object.
<i>samps</i>	A time specified in samples.

Returns

The time specified in ms.

36.49.3.20 void itm_setresolution (double *res*)

Set the number of ticks-per-quarter-note globally for the itm system.

The default is 480.

Parameters

<i>res</i>	The number of ticks-per-quarter-note.
------------	---------------------------------------

See also

[itm_getresolution\(\)](#)

36.49.3.21 void itm_settimesignature (t_itm **x*, long *num*, long *denom*, long *flags*)

Set an itm object's current time signature.

Parameters

<i>x</i>	The itm object.
<i>num</i>	The top number of the time signature.
<i>denom</i>	The bottom number of the time signature.
<i>flags</i>	Currently unused – pass zero.

36.49.3.22 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 itm object.

Parameters

<i>x</i>	An itm object.
<i>ticks</i>	The number of ticks to translate into a time represented as bars, beats, and ticks.
<i>bars</i>	The address of a variable to hold the measure number of the location/position upon return.
<i>beats</i>	The address of a variable to hold the beat number of the location/position upon return.
<i>units</i>	The address of a variable to hold the number of ticks past the beat of the location/position upon return.
<i>position</i>	Set this parameter to TIME_FLAGS_LOCATION or to zero (for position mode).

36.49.3.23 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 itm object.

Parameters

<i>x</i>	An itm object.
<i>ticks</i>	A time specified in ticks.

Returns

The time specified in ms.

36.49.3.24 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

<i>ox</i>	Time object.
<i>vitm</i>	The associated ITM object (use time_getitm() to determine it).
<i>oq</i>	A time object that holds a quantization interval, can be NULL.

Returns

Interval (in ticks) for scheduling this object.

36.49.3.25 void* time_getitm (t_timeobject * ox)

Return the ITM object associated with this time object.

Parameters

<i>ox</i>	Time object.
-----------	--------------

Returns

The associated [t_itm](#) object.

36.49.3.26 double time_getms (t_timeobject * x)

Convert the value of a time object to milliseconds.

Parameters

<i>x</i>	The time object.
----------	------------------

Returns

The time object's value, converted to milliseconds.

36.49.3.27 t_object* time_getnamed (t_object * owner, t_symbol * attrname)

Return a time object associated with an attribute of an owning object.

Parameters

<i>owner</i>	Object that owns this time object (task routine, if any, will pass owner as argument).
<i>attrname</i>	Name of the attribute associated with the time object.

Returns

The [t_timeobject](#) associated with the named attribute.

36.49.3.28 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

<i>tx</i>	The time object.
<i>phase</i>	Pointer to a double to receive the progress within the specified time value of the associated ITM object.
<i>slope</i>	Pointer to a double to receive the slope (phase difference) within the specified time value of the associated ITM object.
<i>ticks</i>	

36.49.3.29 double time_getticks (t_timeobject * *x*)

Convert the value of a time object to ticks.

Parameters

<i>x</i>	The time object.
----------	------------------

Returns

The time object's value, converted to ticks.

36.49.3.30 long time_isfixedunit (t_timeobject * *x*)

Return whether this time object currently holds a fixed (millisecond-based) value.

Parameters

<i>x</i>	Time object.
----------	--------------

Returns

True if time object's current value is fixed, false if it is tempo-relative.

36.49.3.31 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

<i>x</i>	The time object.
<i>attr</i>	Name of the millisecond based attribute in the owning object that will be updated
<i>flags</i>	If TIME_FLAGS_LISTENTICKS is passed here, updating will not happen if the time value is fixed (ms) based

36.49.3.32 void* time_new (t_object * owner, t_symbol * attrname, method tick, long flags)

Create a new time object.

Parameters

<i>owner</i>	Object that will own this time object (task routine, if any, will pass owner as argument).
<i>attrname</i>	Name of the attribute associated with the time object.
<i>tick</i>	Task routine that will be executed (can be NULL)
<i>flags</i>	Options, see "Flags that determine time object behavior" above

Returns

The newly created [t_timeobject](#).

36.49.3.33 void time_now (t_timeobject * x, t_timeobject * quantize)

Schedule a task for right now, with optional quantization.

Parameters

<i>x</i>	The time object that schedules temporary events. The time interval is ignored and 0 ticks is used instead.
<i>quantize</i>	A time object that holds a quantization interval, can be NULL for no quantization.

36.49.3.34 void time_schedule (t_timeobject * x, t_timeobject * quantize)

Schedule a task, with optional quantization.

Parameters

<i>x</i>	The time object that schedules temporary events (must have been created with TIME_FLAGS_USECLOCK but not TIME_FLAGS_PERMANENT)
<i>quantize</i>	A time object that holds a quantization interval, can be NULL for no quantization.

36.49.3.35 void time_schedule_limit (t_timeobject * x, t_timeobject * quantize)

Schedule a task, with optional minimum interval,.

Parameters

<i>x</i>	The time object that schedules temporary events (must have been created with TIME_FLAGS_USECLOCK but not TIME_FLAGS_PERMANENT)
<i>quantize</i>	The minimum interval into the future when the event can occur, can be NULL if there is no minimum interval.

36.49.3.36 void time_setclock (**t_timeobject** * *tx*, **t_symbol** * *sc*)

Associate a named setclock object with a time object (unsupported).

Parameters

<i>tx</i>	Time object.
<i>sc</i>	Name of an associated setclock object.

36.49.3.37 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

<i>tx</i>	The time object.
<i>s</i>	Message selector.
<i>argc</i>	Count of arguments.
<i>argv</i>	Message arguments.

36.49.3.38 void time_stop (*t_timeobject* * *x*)

Stop a currently scheduled time object.

Parameters

<i>x</i>	The time object.
----------	------------------

36.49.3.39 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

<i>x</i>	The time object.
----------	------------------

36.49.4 Variable Documentation**36.49.4.1 BEGIN_USING_C_LINKAGE *typedef t_object t_itm***

A low-level object for tempo-based scheduling.

See also

[t_timeobject](#)
[ITM](#)

36.49.4.2 BEGIN_USING_C_LINKAGE *typedef t_object t_timeobject*

A high-level time object for tempo-based scheduling.

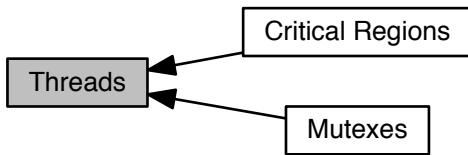
See also

[t_itm](#)
[ITM](#)

36.50 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**

Macros

- `#define ATOMIC_INCREMENT(atomicptr)`

*increment an atomic int value
return value of ATOMIC_INCREMENT and ATOMIC_DECREMENT is the *new* value after performing the operation*

- `#define ATOMIC_INCREMENT_BARRIER(atomicptr)`

*increment an atomic int value with a memory barrier
return value of ATOMIC_INCREMENT and ATOMIC_DECREMENT is the *new* value after performing the operation*

- `#define ATOMIC_DECREMENT(atomicptr)`

*decrement an atomic int value
return value of ATOMIC_INCREMENT and ATOMIC_DECREMENT is the *new* value after performing the operation*

- `#define ATOMIC_DECREMENT_BARRIER(atomicptr)`

*decrement an atomic int value with a memory barrier
return value of ATOMIC_INCREMENT and ATOMIC_DECREMENT is the *new* value after performing the operation*

TypeDefs

- `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, SYSTHREAD_MUTEX_ERRORCHECK, SYSTHREAD_MUTEX_RECURSIVE }`
`systhread_mutex_new() flags`

Functions

- void `schedule` (void *ob, method fun, long when, t_symbol *sym, short argc, t_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()`.
- long `systhread_detach` (t_systhread thread)

Detach a thread.
- 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 a scheduler thread.
- short `systhread_isaudiothread` (void)

Check to see if the function currently being executed is in an audio thread.

Variables

- `BEGIN_USING_C_LINKAGE` typedef void * t_systhread

An opaque thread instance pointer.

36.50.1 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/JitterSdk+SchedQueue>
<http://www.cycling74.com/story/2005/5/2/133649/9742>

36.50.2 Enumeration Type Documentation

36.50.2.1 enum e_max_systhread_mutex_flags

[systhread_mutex_new\(\)](#) flags

Enumerator

SYSTHREAD_MUTEX_NORMAL Normal.
SYSTHREAD_MUTEX_ERRORCHECK Error-checking.
SYSTHREAD_MUTEX_RECURSIVE Recursive.

36.50.3 Function Documentation

36.50.3.1 [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

<i>ob</i>	First argument passed to the function fun when it executes.
<i>fn</i>	Function to be called, see below for how it should be declared.
<i>sym</i>	Second argument passed to the function fun when it executes.
<i>argc</i>	Count of arguments in <i>argv</i> . <i>argc</i> is also the third argument passed to the function fun when it executes.
<i>argv</i>	Array containing a variable number of t_atom function arguments. If this argument is non-zero, <i>defer</i> allocates memory to make a copy of the arguments (according to the size passed in <i>argc</i>) 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:

```
1 void myobject_do (myObject *client, t_symbol *s, short argc, t_atom *argv);
```

See also

[defer_low\(\)](#)

Referenced by [jit_error_code\(\)](#), [jit_error_sym\(\)](#), and [jit_post_sym\(\)](#).

36.50.3.2 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

<i>ob</i>	First argument passed to the function fun when it executes.
<i>fn</i>	Function to be called, see below for how it should be declared.
<i>sym</i>	Second argument passed to the function fun when it executes.
<i>argc</i>	Count of arguments in argv. argc is also the third argument passed to the function fun when it executes.
<i>argv</i>	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:

```
1 void myobject_do (myObject *client, t_symbol *s, short argc, t_atom *argv);
```

See also

[defer\(\)](#)

36.50.3.3 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.

36.50.3.4 void schedule (void * ob, method fun, long when, t_symbol * sym, short argc, t_atom * argv)

Cause a function to be executed at the timer level at some time in the future.

Parameters

<i>ob</i>	First argument passed to the function fun when it executes.
<i>fun</i>	Function to be called, see below for how it should be declared.
<i>when</i>	The logical time that the function fun will be executed.
<i>sym</i>	Second argument passed to the function fun when it executes.
<i>argc</i>	Count of arguments in argv. argc is also the third argument passed to the function fun when it executes.

<code>argv</code>	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:

```
1 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\(\)](#)

36.50.3.5 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

<code>ob</code>	First argument passed to the function fun when it executes.
<code>fun</code>	Function to be called, see below for how it should be declared.
<code>delay</code>	The delay from the current time before the function will be executed.
<code>sym</code>	Second argument passed to the function fun when it executes.
<code>argc</code>	Count of arguments in argv. argc is also the third argument passed to the function fun when it executes.
<code>argv</code>	Array containing a variable number of t_atom function arguments. If this argument is non-zero, schedule_delay() 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.

```
1 void myobject_click (t_myobject *x, Point pt, short modifiers)
2 {
3     t_atom a[1];
4     a[0].a_type = A_LONG;
5     a[0].a_w.w_long = Random();
6     schedule_delay(x, myobject_sched, 0 ,0, 1, a);
7 }
8
9 void myobject_sched (t_myobject *x, t_symbol *s, short ac, t_atom *av)
10 {
11     outlet_int(x->m_out, av->a_w.w_long);
12 }
```

See also

[schedule\(\)](#)

```
36.50.3.6 long systhread_create ( method entryproc, void * arg, long stacksize, long priority, long flags, t_systhread *  
        thread )
```

Create a new thread.

Parameters

<i>entryproc</i>	A method to call in the new thread when the thread is created.
<i>arg</i>	An argument to pass to the method specified for entryproc. Typically this might be a pointer to your object's struct.
<i>stacksize</i>	Not used. Pass 0 for this argument.
<i>priority</i>	Pass 0 for default priority. The priority can range from -32 to 32 where -32 is low, 0 is default and 32 is high.
<i>flags</i>	Not used. Pass 0 for this argument.
<i>thread</i>	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](#).

36.50.3.7 long systhread_detach (t_systhread *thread*)

Detach a thread.

After detaching a thread you cannot call [systhread_join\(\)](#) on it.

Parameters

<i>thread</i>	The thread to join.
---------------	---------------------

Returns

A Max error code as defined in [e_max_errorcodes](#).

Remarks

You should either call [systhread_join\(\)](#) on a thread or [systhread_detach\(\)](#) to allow the system to reclaim resources.

36.50.3.8 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

<i>status</i>	You will typically pass 0 for status. This value will be accessible by systhread_join() , if needed.
---------------	--

36.50.3.9 int systhread_getpriority (t_systhread *thread*)

Get the thread priority for the given thread.

Parameters

<i>thread</i>	The thread for which to find the priority.
---------------	--

Returns

The current priority value for the given thread.

36.50.3.10 short systhread_isaudiothread(void)

Check to see if the function currently being executed is in an audio thread.

Returns

Returns true if the function is being executed in an audio thread, otherwise false.

36.50.3.11 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.

36.50.3.12 short systhread_istimerthread(void)

Check to see if the function currently being executed is in a scheduler thread.

Returns

Returns true if the function is being executed in a scheduler thread, otherwise false.

36.50.3.13 long systhread_join(t_systhread *thread*, unsigned int * *retval*)

Wait for thread to quit and get return value from [systhread_exit\(\)](#).

Parameters

<i>thread</i>	The thread to join.
<i>retval</i>	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.

36.50.3.14 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.

36.50.3.15 void systhread_setpriority(t_systhread *thread*, int *priority*)

Set the thread priority for the given thread.

Parameters

<i>thread</i>	The thread for which to set the priority.
<i>priority</i>	A value in the range -32 to 32 where -32 is lowest, 0 is default, and 32 is highest.

36.50.3.16 void systhread_sleep (long *milliseconds*)

Suspend the execution of the calling thread.

Parameters

<i>milliseconds</i>	The number of milliseconds to suspend the execution of the calling thread. The actual amount of time may be longer depending on various factors.
---------------------	--

36.50.3.17 long systhread_terminate (t_systhread *thread*)

Forcefully kill a thread – not recommended.

Parameters

<i>thread</i>	The thread to kill.
---------------	---------------------

Returns

A Max error code as defined in [e_max_errorcodes](#).

36.51 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:



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.

Variables

- `BEGIN_USING_C_LINKAGE` `typedef MPCriticalRegionID t_critical`
a critical region

36.51.1 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".

36.51.2 Function Documentation

36.51.2.1 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\(\)](#)

Referenced by [jit_global_critical_enter\(\)](#).

36.51.2.2 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.
---	---

Referenced by [jit_global_critical_exit\(\)](#).

36.51.2.3 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.
---	---

36.51.2.4 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.
---	---

36.51.2.5 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\(\)](#)

36.52 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()`.

36.52.1 Detailed Description

See also

[Critical Regions](#)

36.52.2 Function Documentation

36.52.2.1 long `systhread_mutex_free (t_systhread_mutex pmutex)`

Free a mutex created with `systhread_mutex_new()`.

Parameters

<code>pmutex</code>	The mutex instance pointer.
---------------------	-----------------------------

Returns

A Max error code as defined in [e_max_errorcodes](#).

36.52.2.2 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

<i>pmutex</i>	The mutex instance pointer.
---------------	-----------------------------

Returns

A Max error code as defined in [e_max_errorcodes](#).

See also

[systhread_mutex_trylock\(\)](#)

36.52.2.3 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

<i>pmutex</i>	The address of a variable to store the mutex pointer.
<i>flags</i>	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.

36.52.2.4 long systhread_mutex_newlock (t_systhread_mutex * *pmutex*, long *flags*)

Convenience utility that combines [systhread_mutex_new\(\)](#) and [systhread_mutex_lock\(\)](#).

Parameters

<i>pmutex</i>	The address of a variable to store the mutex pointer.
<i>flags</i>	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](#).

36.52.2.5 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

<i>pmutex</i>	The mutex instance pointer.
---------------	-----------------------------

Returns

Returns non-zero if there was a problem entering.

See also

[systhread_mutex_lock\(\)](#)

36.52.2.6 long systhread_mutex_unlock (t_systhread_mutex *pmutex*)

Exit a block of code locked with [systhread_mutex_lock\(\)](#).

Parameters

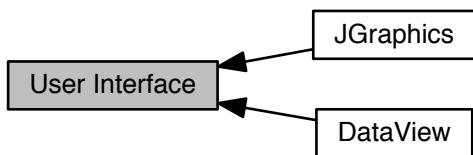
<i>pmutex</i>	The mutex instance pointer.
---------------	-----------------------------

Returns

A Max error code as defined in [e_max_errorcodes](#).

36.53 User Interface

Collaboration diagram for User Interface:



Modules

- [JGraphics](#)

JGraphics is the API for creating user interface objects introduced with Max 5.

- [DataView](#)

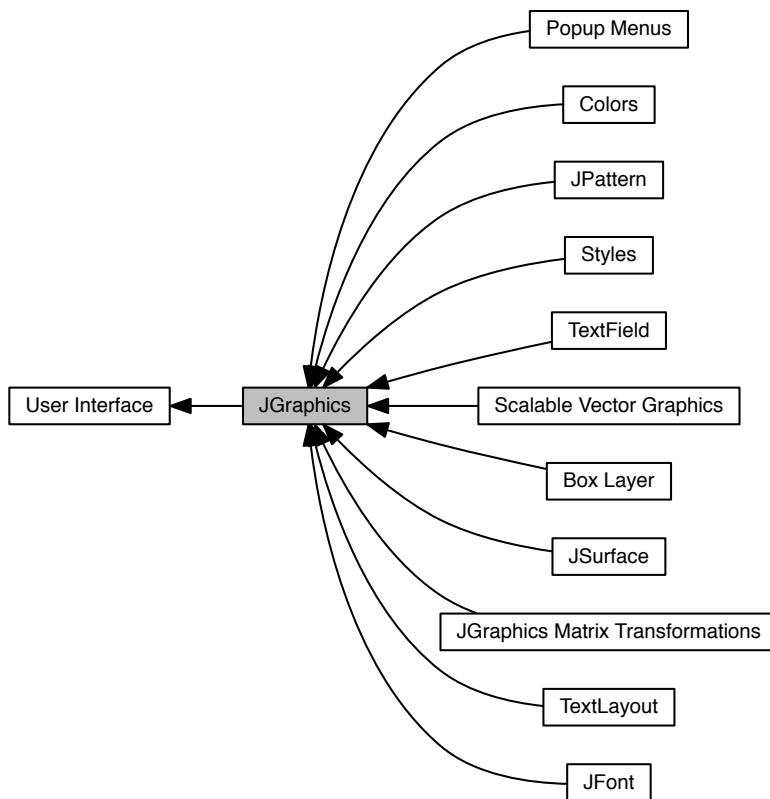
The jdataview object provides a mechanism to display data in a tabular format.

36.53.1 Detailed Description

36.54 JGraphics

JGraphics is the API for creating user interface objects introduced with Max 5.

Collaboration diagram for JGraphics:



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.

- [JPattern](#)

A pattern is like a brush that is used to fill a path with.

- [Colors](#)

- [Styles](#)

Styles provide a means by which to inherit attribute values from a patcher that are consistently used across many objects.

- [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.

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.

Macros

- #define [JGRAPHICS_RECT_BOTTOM\(rect\)](#)

Determine the coordinate of the bottom of a rect.

- #define [JGRAPHICS_RECT_RIGHT\(rect\)](#)

Determine the coordinate of the right side of a rect.

- #define [JGRAPHICS_PI](#)

Utility macro to return the value of Pi.

- #define [JGRAPHICS_2PI](#)

Utility macro to return the value of twice Pi.

- #define [JGRAPHICS_PIOVER2](#)

Utility macro to return the value of half of Pi.

- #define [JGRAPHICS_3PIOVER2](#)

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 _jdesktopui [t_jdesktopui](#)

An instance of a transparent UI window on the desktop.

- typedef struct _jpopupmenu [t_jpopupmenu](#)

An instance of a pop-up menu.

- typedef struct _jsvg [t_jsvg](#)

An instance of an SVG object.

- typedef struct _jsvg_remap [t_jsvg_remap](#)

An object used for remapping colors in a t_svg.

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`, `JGRAPHICS_TEXT JUSTIFICATION RIGHT`, `JGRAPHICS_TEXT JUSTIFICATION HCENTERED`, `JGRAPHICS_TEXT JUSTIFICATION TOP`, `JGRAPHICS_TEXT JUSTIFICATION BOTTOM`, `JGRAPHICS_TEXT JUSTIFICATION VCENTERED`, `JGRAPHICS_TEXT JUSTIFICATION HJUSTIFIED`, `JGRAPHICS_TEXT JUSTIFICATION CENTERED` }

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.
- `t_jpath * jgraphics_path_createstroked` (`t_jpath *p`, double thickness, `t_jgraphics_line_join` join, `t_jgraphics_line_cap` cap)

Create a new path consisting of the original path stroked with a given thickness.
- 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.
- long `jgraphics_path_contains` (`t_jpath *path`, double x, double y)

Test if the path contains the point x,y.
- long `jgraphics_path_intersectsline` (`t_jpath *path`, double x1, double y1, double x2, double y2)

Test if the path intersects the line defined by x1,y1 and x2,y2.
- double `jgraphics_path_getlength` (`t_jpath *path`)

Return the length of a path.
- void `jgraphics_path_getpointalongpath` (`t_jpath *path`, double distancefromstart, double *x, double *y)

Return a point that lies a given distance from the start of the path.
- double `jgraphics_path_getnearestpoint` (`t_jpath *path`, double x, double y, double *path_x, double *path_y)

Finds the point on the path that is nearest to the point x,y passed in.
- long `jgraphics_path_getpathelems` (`t_jpath *path`, `t_jgraphics_path_elem **elems`)

Get the path elements and return number of path elements.
- 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_bubble (t_jgraphics *g, double bodyx, double bodyy, double bodywidth, double bodyheight, double cornersize, double arrowtipx, double arrowtipy, t_jgraphics_bubble_side whichside, double arrowedgeprop, double arrowwidth)`
Add a closed bubble path in the context.
- void `jgraphics_triangle (t_jgraphics *g, double x1, double y1, double x2, double y2, double x3, double y3)`
Add a closed triangular 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_text_path (t_jgraphics *g, const char *utf8)`
Add a path of text to the current path.
- 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_measuretext_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_canantialiasexttotransparentbg ()`

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, t_fourcc **filetypes, char *alloc)`

Get a list of 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.

- void `jgraphics_clip (t_jgraphics *g, double x, double y, double width, double height)`

Clip to a subset of the graphics context; once done, cannot be undone, only further reduced.

36.54.1 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).

36.54.2 Macro Definition Documentation

36.54.2.1 #define JGRAPHICS_2PI

Utility macro to return the value of twice Pi.

36.54.2.2 #define JGRAPHICS_3PIOVER2

Utility macro to return the 270° Case.

36.54.2.3 #define JGRAPHICS_PI

Utility macro to return the value of Pi.

36.54.2.4 #define JGRAPHICS_PIOVER2

Utility macro to return the value of half of Pi.

36.54.3 Enumeration Type Documentation

36.54.3.1 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.

36.54.3.2 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.

36.54.3.3 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.

36.54.4 Function Documentation

36.54.4.1 void jgraphics_append_path(t_jgraphics * g, t_jpath * path)

Add a path to a graphics context.

Parameters

<i>g</i>	The graphics context.
<i>path</i>	The path to add.

36.54.4.2 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

<i>g</i>	The graphics context.
<i>xc</i>	The horizontal coordinate of the arc's center.
<i>yc</i>	The vertical coordinate of the arc's center.
<i>radius</i>	The radius of the arc.
<i>angle1</i>	The starting angle of the arc in radians. Zero radians is center right (positive x axis).
<i>angle2</i>	The terminal angle of the arc in radians. Zero radians is center right (positive x axis).

```
36.54.4.3 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

<i>g</i>	The graphics context.
<i>xc</i>	The horizontal coordinate of the arc's center.
<i>yc</i>	The vertical coordinate of the arc's center.
<i>radius</i>	The radius of the arc.
<i>angle1</i>	The starting angle of the arc in radians. Zero radians is center right (positive x axis).
<i>angle2</i>	The terminal angle of the arc in radians. Zero radians is center right (positive x axis).

```
36.54.4.4 void jgraphics_bubble ( t_jgraphics * g, double bodyx, double bodyy, double bodywidth, double bodyheight, double cornersize, double arrowtipx, double arrowtipy, t_jgraphics_bubble_side whichside, double arrowedgeprop, double arrowwidth )
```

Add a closed bubble path in the context.

Parameters

<i>g</i>	The graphics context.
<i>bodyx</i>	Horizontal body origin.
<i>bodyy</i>	The vertical origin.
<i>bodywidth</i>	The width of the rect.
<i>bodyheight</i>	The height of the rect.
<i>cornersize</i>	Body rounded corners
<i>arrowtipx</i>	X position of arrow tip
<i>arrowtipy</i>	Y position of arrow tip
<i>whichside</i>	side to connect arrow, 0 = top, 1 = left, 2 = bottom, 3 = right,
<i>arrowedgeprop</i>	Arrow proportion along edge (0-1)
<i>arrowwidth</i>	Arrow base width

```
36.54.4.5 void jgraphics_clip ( t_jgraphics * g, double x, double y, double width, double height )
```

Clip to a subset of the graphics context; once done, cannot be undone, only further reduced.

Parameters

<i>g</i>	The t_jgraphics context to be clipped.
<i>x</i>	x origin of clip region.
<i>y</i>	y origin of clip region.

<i>width</i>	width of clip region.
<i>height</i>	height of clip region.

36.54.4.6 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

<i>g</i>	The graphics context.
----------	-----------------------

36.54.4.7 **t_jpath*** jgraphics_copy_path (**t_jgraphics** * *g*)

Get a copy of the current path from a context.

Parameters

<i>g</i>	the graphics context containing the current path
----------	--

Returns

A copy of the current path.

36.54.4.8 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

<i>g</i>	The graphics context.
<i>x1</i>	The first control point.
<i>y1</i>	The first control point.
<i>x2</i>	The second control point.
<i>y2</i>	The second control point.
<i>x3</i>	The destination point.
<i>y3</i>	The destination point.

36.54.4.9 void jgraphics_destroy (**t_jgraphics** * *g*)

Release or free a graphics context.

Parameters

<i>g</i>	The context to release.
----------	-------------------------

36.54.4.10 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.

36.54.4.11 void **jgraphics_ellipse** (**t_jgraphics** * *g*, double *x*, double *y*, double *width*, double *height*)

Add a closed elliptical path in the context.

Parameters

<i>g</i>	The graphics context.
<i>x</i>	The horizontal origin.
<i>y</i>	The vertical origin.
<i>width</i>	The width of the rect.
<i>height</i>	The height of the rect.

36.54.4.12 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

<i>g</i>	Pointer to a jgraphics context.
<i>extents</i>	The address of a <i>t_jgraphics_font_extents</i> structure to be filled with the results.

36.54.4.13 void jgraphics_get_current_point (*t_jgraphics* * *g*, double * *x*, double * *y*)

Get the current location of the cursor in a graphics context.

Parameters

<i>g</i>	The graphics context.
<i>x</i>	The address of a variable that will be set to the horizontal cursor location upon return.
<i>y</i>	The address of a variable that will be set to the vertical cursor location upon return.

36.54.4.14 void jgraphics_getfiletypes (void * *dummy*, long * *count*, *t_fourcc* ** *filetypes*, char * *alloc*)

Get a list of filetypes appropriate for use with jgraphics surfaces.

Parameters

<i>dummy</i>	Unused.
<i>count</i>	The address of a variable to be set with the number of types in filetypes upon return.
<i>filetypes</i>	The address of a variable that will represent the array of file types upon return.
<i>alloc</i>	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\(\)](#).

```

1   char      filename[MAX_PATH_CHARS];
2   t_fourcc  *type = NULL;
3   long      ntype;
4   long      outtype;
5   t_max_err err;
6   char      alloc;
7   short     path;
8   t_jsurface *surface;
9
10  if (want_to_show_dialog) {
11      jgraphics_getfiletypes(x, &ntype, &type, &alloc);
12      err = open_dialog(filename, &path, (void *)&outtype, (void *)type, ntype);
13      if (err)
14          goto out;
15  }
16  else {
17      strncpy_zero(filename, s->s_name, MAX_PATH_CHARS);
18      err = locatefile_extended(filename, &path, &outtype, type, ntype);
19      if (err)
20          goto out;
21  }

```

```

22     surface = jgraphics_image_surface_create_referenced(filename, path);
23 out:
24     if (alloc)
25         sysmem_freeptr((char *)type);

```

36.54.4.15 void **jgraphics_line_to** (**t_jgraphics** * *g*, double *x*, double *y*)

Add a line segment to the current path.

Parameters

<i>g</i>	The graphics context.
<i>x</i>	The destination point.
<i>y</i>	The destination point.

36.54.4.16 void **jgraphics_move_to** (**t_jgraphics** * *g*, double *x*, double *y*)

Move the cursor to a new point and begin a new subpath.

Parameters

<i>g</i>	The graphics context.
<i>x</i>	The new location.
<i>y</i>	The new location.

36.54.4.17 void **jgraphics_new_path** (**t_jgraphics** * *g*)

Begin a new path.

This action clears any current path in the context.

Parameters

<i>g</i>	The graphics context.
----------	-----------------------

36.54.4.18 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

<i>g</i>	The graphics context.
<i>x</i>	The horizontal origin.
<i>y</i>	The vertical origin.
<i>width</i>	The width of the oval.
<i>height</i>	The height of the oval.

36.54.4.19 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

<i>g</i>	The graphics context.
<i>xc</i>	The horizontal coordinate of the arc's center.
<i>yc</i>	The vertical coordinate of the arc's center.
<i>radiusx</i>	The horizontal radius of the arc.
<i>radiusy</i>	The vertical radius of the arc.
<i>angle1</i>	The starting angle of the arc in radians. Zero radians is center right (positive x axis).
<i>angle2</i>	The terminal angle of the arc in radians. Zero radians is center right (positive x axis).

36.54.4.20 long **jgraphics_path_contains** (*t_jpath * path, double x, double y*)

Test if the path contains the point x,y.

Parameters

<i>path</i>	the path
<i>x</i>	the x-coordinate of the point to test
<i>y</i>	the y-coordinate of the point to test

36.54.4.21 *t_jpath* jgraphics_path_createstroked (t_jpath * p, double thickness, t_jgraphics_line_join join, t_jgraphics_line_cap cap)*

Create a new path consisting of the original path stroked with a given thickness.

Parameters

<i>p</i>	the path to be stroked
<i>thickness</i>	thickness of the stroke
<i>join</i>	the style to join segments together at corners
<i>cap</i>	the style of end cap to use

Returns

the new path, which must be freed with [jgraphics_path_destroy\(\)](#) when done

36.54.4.22 void **jgraphics_path_destroy** (*t_jpath * path*)

Release/free a path.

Parameters

<i>path</i>	The path to release.
-------------	----------------------

36.54.4.23 double **jgraphics_path_getlength** (*t_jpath * path*)

Return the length of a path.

Parameters

<i>path</i>	the path
-------------	----------

Returns

the length of the path

36.54.4.24 double **jgraphics_path_getnearestpoint** (*t_jpath * path*, *double x, double y, double * path_x, double * path_y*)

Finds the point on the path that is nearest to the point x,y passed in.

Parameters

<i>path</i>	the path to search
<i>x</i>	x position of the target point
<i>y</i>	y position of the target point
<i>path_x</i>	pointer to double to receive the x position of closest point on path
<i>path_y</i>	pointer to double to receive the y position of the closest point on path

Returns

returns the distance along the path from the path start position to the found point on the path

36.54.4.25 long jgraphics_path_getpathelems (t_jpath * path, t_jgraphics_path_elem ** elems)

Get the path elements and return number of path elements.

Parameters

<i>path</i>	the path
<i>elems</i>	pointer to array of path elements

Returns

the number of path elements

36.54.4.26 void jgraphics_path_getpointalongpath (t_jpath * path, double distancefromstart, double * x, double * y)

Return a point that lies a given distance from the start of the path.

Parameters

<i>path</i>	the path
<i>distancefrom-start</i>	distance from the start point
<i>x</i>	pointer to double to receive the x position of the point
<i>y</i>	pointer to double to receive the y position of the point

36.54.4.27 long jgraphics_path_intersectsline (t_jpath * path, double x1, double y1, double x2, double y2)

Test if the path intersects the line defined by x1,y1 and x2,y2.

Parameters

<i>path</i>	the path
<i>x1</i>	the x-coordinate of the first point on the line
<i>y1</i>	the y-coordinate of the first point on the line
<i>x2</i>	the x-coordinate of the second point on the line
<i>y2</i>	the y-coordinate of the second point on the line

36.54.4.28 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

<i>g</i>	The graphics context.
<i>cornerRadius</i>	The amount by which to round corners.

36.54.4.29 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

<i>positioned_rect</i>	The address of a valid <i>t_rect</i> whose members will be filled in upon return.
<i>positioned_rect</i>	A pointer to a rect near which this rect should be positioned.
<i>keep_inside_rect</i>	A pointer to a rect defining the limits within which the new rect must reside.

36.54.4.30 void **jgraphics_rectangle** (*t_jgraphics * g*, *double x*, *double y*, *double width*, *double height*)

Add a closed rectangle path in the context.

Parameters

<i>g</i>	The graphics context.
<i>x</i>	The horizontal origin.
<i>y</i>	The vertical origin.
<i>width</i>	The width of the rect.
<i>height</i>	The height of the rect.

36.54.4.31 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

<i>g</i>	The graphics context.
<i>x</i>	The horizontal origin.
<i>y</i>	The vertical origin.
<i>width</i>	The width of the rect.
<i>height</i>	The height of the rect.
<i>ovalwidth</i>	The width of the oval used for the round corners.
<i>ovalheight</i>	The height of the oval used for the round corners.

36.54.4.32 long **jgraphics_rectcontainsrect** (*t_rect * outer*, *t_rect * inner*)

Simple utility to test for rectangle containment.

Parameters

<i>outer</i>	The address of the first rect for the test.
--------------	---

<i>inner</i>	The address of the second rect for the test.
--------------	--

Returns

Returns true if the inner rect is completely inside the outer rect, otherwise false.

36.54.4.33 long jgraphics_rectintersectsrect (t_rect * r1, t_rect * r2)

Simple utility to test for rectangle intersection.

Parameters

<i>r1</i>	The address of the first rect for the test.
<i>r2</i>	The address of the second rect for the test.

Returns

Returns true if the rects intersect, otherwise false.

36.54.4.34 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

<i>g</i>	The context you wish to reference.
----------	------------------------------------

Returns

A new reference to the context.

36.54.4.35 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

<i>g</i>	The graphics context.
<i>x1</i>	The first control point.
<i>y1</i>	The first control point.
<i>x2</i>	The second control point.
<i>y2</i>	The second control point.
<i>x3</i>	The destination point.
<i>y3</i>	The destination point.

36.54.4.36 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

<i>g</i>	The graphics context.
<i>x</i>	The destination point.
<i>y</i>	The destination point.

36.54.4.37 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

<i>g</i>	The graphics context.
<i>x</i>	The new location.
<i>y</i>	The new location.

36.54.4.38 int **jgraphics_round** (double *d*)

Utility for rounding a double to an int.

Parameters

<i>d</i>	floating-point input.
----------	-----------------------

Returns

rounded int output.

36.54.4.39 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

<i>g</i>	The graphics context.
<i>family</i>	The name of the font family (e.g. "Arial").
<i>slant</i>	Define the slant to use for the font.
<i>weight</i>	Define the weight to use for the font.

36.54.4.40 void **jgraphics_select_jfont** (**t_jgraphics** * *g*, **t_jfont** * *jfont*)

Specify a font for a graphics context by passing a [t_jfont](#) object.

Parameters

<i>g</i>	The graphics context.
<i>jfont</i>	A jfont object whose attributes will be copied to the context.

36.54.4.41 void **jgraphics_set_font_size** (**t_jgraphics** * *g*, double *size*)

Specify the font size for a context.

Parameters

<i>g</i>	The graphics context.
<i>size</i>	The font size.

36.54.4.42 void jgraphics_set_underline (*t_jgraphics* * *g*, *char* *underline*)

Turn underlining on/off for text in a context.

Parameters

<i>g</i>	The graphics context.
<i>underline</i>	Pass true or false to set the appropriate behavior.

36.54.4.43 void jgraphics_show_text (*t_jgraphics* * *g*, *const char* * *utf8*)

Display text at the current position in a context.

Parameters

<i>g</i>	The graphics context.
<i>utf8</i>	The text to display.

36.54.4.44 long jgraphics_system_canantialias_text_to_transparentbg ()

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.

36.54.4.45 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

<i>g</i>	Pointer to a jgraphics context.
<i>utf8</i>	A string containing the text whose dimensions we wish to find.
<i>width</i>	The address of a variable to be filled with the width of the rendered text.
<i>height</i>	The address of a variable to be filled with the height of the rendered text.

36.54.4.46 void jgraphics_text_measuretext_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

<i>g</i>	Pointer to a jgraphics context.
<i>utf8</i>	A string containing the text whose dimensions we wish to find.
<i>wrapwidth</i>	The number of pixels in width at which the text should be wrapped if it is too long.
<i>includewhitespace</i>	Set zero to not include white space in the calculation, otherwise set this parameter to 1.
<i>pace</i>	
<i>width</i>	The address of a variable to be filled with the width of the rendered text.
<i>height</i>	The address of a variable to be filled with the height of the rendered text.
<i>numlines</i>	The address of a variable to be filled with the number of lines required to render the text.

36.54.4.47 void jgraphics_text_path (*t_jgraphics* * *g*, const char * *utf8*)

Add a path of text to the current path.

Parameters

<i>g</i>	The graphics context.
<i>utf8</i>	The text to generate path for.

36.54.4.48 void jgraphics_triangle (*t_jgraphics* * *g*, double *x1*, double *y1*, double *x2*, double *y2*, double *x3*, double *y3*)

Add a closed triangular path in the context.

Parameters

<i>g</i>	The graphics context.
<i>x1</i>	Coordinate for the first point.
<i>y1</i>	Coordinate for the first point.
<i>x2</i>	Coordinate for the second point.
<i>y2</i>	Coordinate for the second point.
<i>x3</i>	Coordinate for the third point.
<i>y3</i>	Coordinate for the third point.

36.54.4.49 void jgraphics_user_to_device (*t_jgraphics* * *g*, double * *x*, double * *y*)

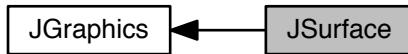
User coordinates are those passed to drawing functions in a given *t_jgraphics* context.

Device coordinates refer to patcher canvas coordinates, before any zooming.

36.55 JSurface

A surface is an abstract base class for something you render to.

Collaboration diagram for JSurface:



Typedefs

- `t_jsurface`
An instance of a jgraphics surface.

Functions

- `t_jsurface * jgraphics_image_surface_create(t_graphics_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_graphics_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.
- `t_max_err jgraphics_image_surface_writejpeg(t_jsurface *surface, const char *filename, short path)`
Export a JPEG 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.

36.55.1 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.

36.55.2 Function Documentation

36.55.2.1 `t_jgraphics* jgraphics_create (t_jsurface * target)`

Create a context to draw on a particular surface.

When you are done, call `jgraphics_destroy()`.

Parameters

<code>target</code>	The surface to which to draw.
---------------------	-------------------------------

Returns

The new graphics context.

36.55.2.2 `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

<code>moduleRef</code>	A pointer to your external's module, which is passed to your external's main() function when the class is loaded.
<code>resname</code>	Base name of the resource data (without an extension)
<code>extcount</code>	Count of possible extensions (ignored on Windows)
<code>exts</code>	Array of symbol atoms containing possible filename extensions (ignored on Windows)

<i>data</i>	Returned resource data assigned to a pointer you supply
<i>datasize</i>	Size of the data returned

Remarks

You are responsible for freeing any data returned in the data pointer

Returns

A Max error code.

36.55.2.3 void jgraphics_image_surface_clear (t_jsurface * s, int x, int y, int width, int height)

Set all pixels in rect to 0.

Parameters

<i>s</i>	The surface to clear.
<i>x</i>	The horizontal origin of the rect to clear.
<i>y</i>	The vertical origin of the rect to clear.
<i>width</i>	The width of the rect to clear.
<i>height</i>	The height of the rect to clear.

36.55.2.4 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

<i>format</i>	Defines the color format for the new surface.
<i>width</i>	Defines the width of the new surface.
<i>height</i>	Defines the height of the new surface.

Returns

A pointer to the new surface.

36.55.2.5 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

<i>data</i>	The data. For example, an RGBA image loaded in memory.
<i>format</i>	The format of the data.
<i>width</i>	The width of the new surface.

<i>height</i>	The height of the new surface.
<i>stride</i>	The number of bytes between the start of rows in the dat buffer.
<i>freefun</i>	If not NULL, freefun will be called when the surface is destroyed
<i>freearg</i>	This will be passed to freefun if/when freefun is called.

Returns

A pointer to the new surface.

36.55.2.6 `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

<i>filename</i>	The name of the file.
<i>path</i>	The path id of the file.

Returns

A pointer to the new surface.

36.55.2.7 `t_jsurface* jgraphics_image_surface_create_from_filedata (const void * data, unsigned long datalen)`

Create a new surface from file data.

Parameters

<i>data</i>	A pointer to the raw PNG or JPG bits.
<i>datalen</i>	The number of bytes in data.

Returns

The new surface.

See also

[jgraphics_write_image_surface_to_filedata\(\)](#)

36.55.2.8 `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

<i>moduleRef</i>	A pointer to your external's module, which is passed to your external's main() function when the class is loaded.
<i>resname</i>	The name of the resource in the external.

Remarks

The following example shows an example of how this might be used in an external.

```

1 static s_my_surface = NULL;
2
3 int main(void *moduleRef)
4 {
5     // (Do typical class initialization here)
6
7     // now create the surface from a resource that we added to the Xcode/VisualStudio project
8     s_my_surface = jgraphics_image_surface_create_from_resource(moduleRef, "myCoolImage");
9
10    return 0;
11 }
```

36.55.2.9 `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

<code>filename</code>	The name of the file.
<code>path</code>	The path id of the file.

Returns

A pointer to the new surface.

36.55.2.10 `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

<code>g</code>	The graphics context in which to draw the surface.
<code>s</code>	The surface to draw.
<code>srcRect</code>	The rect within the surface that should be drawn.
<code>destRect</code>	The rect in the context to which to draw the srcRect.

See also

[jgraphics_image_surface_draw_fast\(\)](#)

36.55.2.11 `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

<i>g</i>	The graphics context in which to draw the surface.
<i>s</i>	The surface to draw.

See also

[jgraphics_image_surface_draw](#)

36.55.2.12 int jgraphics_image_surface_get_height(t_jsurface * s)

Retrieve the height of a surface.

Parameters

<i>s</i>	The surface to query.
----------	-----------------------

Returns

The height of the surface.

36.55.2.13 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

<i>s</i>	The surface.
<i>x</i>	The horizontal coordinate of the pixel.
<i>y</i>	The vertical coordinate of the pixel.
<i>color</i>	The address of a valid t_jrgba struct whose values will be filled in with the color of the pixel upon return.

36.55.2.14 int jgraphics_image_surface_get_width(t_jsurface * s)

Retrieve the width of a surface.

Parameters

<i>s</i>	The surface to query.
----------	-----------------------

Returns

The width of the surface.

36.55.2.15 void jgraphics_image_surface_scroll(t_jsurface * s, int x, int y, int width, int height, int dx, int dy, t_jpath ** path)**Parameters**

<i>s</i>	The surface to scroll.
<i>x</i>	The origin of the rect to scroll.

<i>y</i>	The origin of the rect to scroll.
<i>width</i>	The width of the rect to scroll.
<i>height</i>	The height of the rect to scroll.
<i>dx</i>	The amount to scroll the surface horizontally.
<i>dy</i>	The amount to scroll the surface vertically.
<i>path</i>	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.

36.55.2.16 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

<i>s</i>	The surface.
<i>x</i>	The horizontal coordinate of the pixel.
<i>y</i>	The vertical coordinate of the pixel.
<i>color</i>	The color of the pixel.

36.55.2.17 **t_max_err jgraphics_image_surface_writejpeg** (**t_jsurface * surface**, const char * *filename*, short *path*)

Export a JPEG file of the contents of a surface.

Parameters

<i>surface</i>	The surface to export.
<i>filename</i>	Specify the name of the file to create.
<i>path</i>	Specify the path id for where to create the file.

Returns

A Max error code.

36.55.2.18 **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

<i>surface</i>	The surface to export.
<i>filename</i>	Specify the name of the file to create.
<i>path</i>	Specify the path id for where to create the file.
<i>dpi</i>	Define the resolution of the image (e.g. 72).

Returns

A Max error code.

36.55.2.19 void **jgraphics_surface_destroy** (**t_jsurface * s**)

Release or free a surface.

Parameters

s	The surface to release.
----------	-------------------------

36.55.2.20 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.

36.55.2.21 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.

```

1 long size = 0;
2 void *data = NULL;
3
4 jgraphics_write_image_surface_to_filedata(x->j_surface, JGRAPHICS_FILEFORMAT_PNG, &data, &size);
5 if (size) {
6     x->j_format = gensym("png");
7     binarydata_appendtodictionary(data, size, gensym("data"), x->j_format, d);
8     x->j_imagedata = data;
9     x->j_imagedatasize = size;
10 }
```

See also

[jgraphics_image_surface_create_from_filedata\(\)](#)

36.56 Scalable Vector Graphics

Collaboration diagram for Scalable Vector Graphics:



Functions

- `t_jsvg * js_svg_create_from_file (const char *filename, short path)`
Read an SVG file, return a `t_jsvg` object.
- `t_jsvg * js_svg_create_from_resource (const void *moduleRef, const char *resname)`
Read an SVG file from a resource.
- `t_jsvg * js_svg_create_from_xmlstring (const char *svgXML)`
Create an SVG object from a string containing the SVG's XML.
- `void js_svg_get_size (t_jsvg *svg, double *width, double *height)`
Retrieve the size of an SVG object.
- `void js_svg_destroy (t_jsvg *svg)`
Free a `t_jsvg` object.
- `void js_svg_render (t_jsvg *svg, t_jgraphics *g)`
Render an SVG into a graphics context.

36.56.1 Detailed Description

36.56.2 Function Documentation

36.56.2.1 `t_jsvg* js_svg_create_from_file (const char * filename, short path)`

Read an SVG file, return a `t_jsvg` object.

Parameters

<code>filename</code>	The name of the file to read.
<code>path</code>	The path id of the file to read.

Returns

A new SVG object.

36.56.2.2 `t_jsvg* js_svg_create_from_resource (const void * moduleRef, const char * resname)`

Read an SVG file from a resource.

Parameters

<i>moduleRef</i>	The external's moduleRef.
<i>resname</i>	The name of the SVG resource.

Returns

A new SVG object.

See also

[jgraphics_image_surface_create_from_resource\(\)](#)

36.56.2.3 t_jsvg* jsvg_create_from_xmlstring (const char * svgXML)

Create an SVG object from a string containing the SVG's XML.

Parameters

<i>svgXML</i>	The SVG source.
---------------	-----------------

Returns

A new SVG object.

36.56.2.4 void jsvg_destroy (t_jsvg * svg)

Free a [t_jsvg](#) object.

Parameters

<i>svg</i>	The object to free.
------------	---------------------

36.56.2.5 void jsvg_get_size (t_jsvg * svg, double * width, double * height)

Retrieve the size of an SVG object.

Parameters

<i>svg</i>	An SVG object.
<i>width</i>	The address of a variable that will be set to the width upon return.
<i>height</i>	The address of a variable that will be set to the width upon return.

36.56.2.6 void jsvg_render (t_jsvg * svg, t_jgraphics * g)

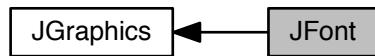
Render an SVG into a graphics context.

Parameters

<i>svg</i>	The SVG object to render.
<i>g</i>	The graphics context in which to render.

36.57 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.
- `long jfont_isequalto (t_jfont *font, t_jfont *other)`
Compare two fonts to see if they are equivalent.
- `void jfont_set_family (t_jfont *font, t_symbol *family)`
Set the name of the font family (e.g.
- `t_symbol * jfont_get_family (t_jfont *font)`
Get the name of the font family (e.g.
- `void jfont_set_slant (t_jfont *font, t_jgraphics_font_slant slant)`
Set the slant of the font.
- `t_jgraphics_font_slant jfont_get_slant (t_jfont *font)`
Get the slant of the font.
- `void jfont_set_weight (t_jfont *font, t_jgraphics_font_weight weight)`
Set the weight of the font.
- `t_jgraphics_font_weight jfont_get_weight (t_jfont *font)`
Get the weight of the font.

- void `jfont_set_font_size` (`t_jfont` *font, double size)
Set the size of a font object.
- double `jfont_get_font_size` (`t_jfont` *font)
Get the size of a font object.
- void `jfont_set_underline` (`t_jfont` *font, char ul)
Set the underlining of a font object.
- char `jfont_get_underline` (`t_jfont` *font)
Get the underline state 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_measuretext_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.
- void `jfont_get_em_dimensions` (`t_jfont` *font, double *width, double *height)
Given a font, find out the width and height of the 'M' character.
- `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.
- const char * `systemfontname` ()
Retrieve the name of Max's system font.
- const char * `systemfontname_bold` ()
Retrieve the name of Max's bold system font.
- const char * `systemfontname_light` ()
Retrieve the name of Max's light system font.
- `t_symbol` * `systemfontsym` ()
Retrieve the name of Max's system font as a symbol.

36.57.1 Detailed Description

36.57.2 Enumeration Type Documentation

36.57.2.1 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.

36.57.2.2 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.

36.57.3 Function Documentation

36.57.3.1 long jbox_get_font_slant(*t_object* * *b*)

Get the slant box's font.

Parameters

<i>b</i>	An object's box.
----------	------------------

Returns

A value from the [t_jgraphics_font_slant](#) enum.

36.57.3.2 long jbox_get_font_weight(*t_object* * *b*)

Get the slant box's font.

Parameters

<i>b</i>	An object's box.
----------	------------------

Returns

A value from the [t_jgraphics_font_weight](#) enum.

36.57.3.3 *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

<i>family</i>	The name of the font family (e.g. Arial).
<i>slant</i>	The type of slant for the font.
<i>weight</i>	The type of weight for the font.
<i>size</i>	The size of the font.

Returns

The new font object.

36.57.3.4 void jfont_destroy(*t_jfont* * *font*)

Release or free a font object.

Parameters

<i>font</i>	The font object to release.
-------------	-----------------------------

36.57.3.5 void jfont_extents(*t_jfont* * *font*, *t_jgraphics_font_extents* * *extents*)

Get extents of this font.

Parameters

<i>font</i>	The font object.
<i>extents</i>	The font extents upon return/

36.57.3.6 void jfont_get_em_dimensions (*t_jfont * font*, *double * width*, *double * height*)

Given a font, find out the width and height of the 'M' character.

This is equivalent to `jfont_text_measure(font, "M", width, height)` but is faster.

Parameters

<i>font</i>	The font object.
<i>width</i>	The address of a variable to hold the width upon return.
<i>height</i>	The address of a variable to hold the height upon return.

36.57.3.7 *t_symbol jfont_get_family (*t_jfont * font*)**

Get the name of the font family (e.g.

Arial).

Parameters

<i>font</i>	The font object.
-------------	------------------

Returns

A [t_symbol](#) representing the name of the font family.

36.57.3.8 double jfont_get_font_size (*t_jfont * font*)

Get the size of a font object.

Parameters

<i>font</i>	The font object.
-------------	------------------

Returns

The size of the font.

36.57.3.9 *t_jgraphics_font_slant* jfont_get_slant (*t_jfont * font*)

Get the slant of the font.

Parameters

<i>font</i>	The font object.
-------------	------------------

Returns

The current slant setting for the font.

36.57.3.10 char jfont_get_underline (*t_jfont * font*)

Get the underline state of a font object.

Parameters

<i>font</i>	The font object.
-------------	------------------

Returns

Nonzero value if the font will be underlined.

36.57.3.11 t_jgraphics_font_weight jfont_get_weight (t_jfont * *font*)

Get the weight of the font.

Parameters

<i>font</i>	The font object.
-------------	------------------

Returns

The current weight setting for the font.

36.57.3.12 t_max_err jfont_getfontlist (long * *count*, t_symbol * *list*)**

Get a list of font names.

Parameters

<i>count</i>	The address of a variable to hold the count of font names in list upon return.
<i>list</i>	The address of a t_symbol ** initialized to NULL. Upon return this will be set to an array of <i>count</i> t_symbol pointers. This array should be freed using sysmem_freeptr() when you are done with it.

Returns

A Max error code.

36.57.3.13 long jfont_isequalto (t_jfont * *font*, t_jfont * *other*)

Compare two fonts to see if they are equivalent.

Parameters

<i>font</i>	The first font object that is being compared.
<i>other</i>	The second font object that is being compared.

Returns

Nonzero value if the two fonts are equivalent.

36.57.3.14 t_jfont* jfont_reference (t_jfont * *font*)

Create new reference to an existing font object.

Parameters

<i>font</i>	The font object for which to obtain a reference.
-------------	--

Returns

The new font object reference.

36.57.3.15 void jfont_set_family (*t_jfont* * *font*, *t_symbol* * *family*)

Set the name of the font family (e.g.

Arial).

Parameters

<i>font</i>	The font object.
<i>family</i>	A t_symbol containing the name of the desired font family.

36.57.3.16 void jfont_set_font_size (*t_jfont* * *font*, double *size*)

Set the size of a font object.

Parameters

<i>font</i>	The font object.
<i>size</i>	The new size for the font object.

36.57.3.17 void jfont_set_slant (*t_jfont* * *font*, *t_jgraphics_font_slant* *slant*)

Set the slant of the font.

Parameters

<i>font</i>	The font object
<i>slant</i>	The desired slant.

36.57.3.18 void jfont_set_underline (*t_jfont* * *font*, char *ul*)

Set the underlining of a font object.

Parameters

<i>font</i>	The font object.
<i>ul</i>	Pass true to underline, or false for no underlining.

36.57.3.19 void jfont_set_weight (*t_jfont* * *font*, *t_jgraphics_font_weight* *weight*)

Set the weight of the font.

Parameters

<i>font</i>	The font object
<i>weight</i>	The desired weight (e.g. bold).

36.57.3.20 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

<i>font</i>	The font object.
<i>utf8</i>	The text whose rendering will be measured.
<i>width</i>	The address of a variable to hold the width upon return.
<i>height</i>	The address of a variable to hold the height upon return.

36.57.3.21 void jfont_text_measuretext_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

<i>font</i>	The font object.
<i>utf8</i>	The text whose rendering will be measured.
<i>wrapwidth</i>	The maximum width, above which text should wrap onto a new line.
<i>includewhitespace</i>	If non-zero, include whitespace in the measurement.
<i>width</i>	The address of a variable to hold the width upon return.
<i>height</i>	The address of a variable to hold the height upon return.
<i>numlines</i>	The address of a variable to hold the number of lines of text after wrapping upon return.

36.57.3.22 const char* systemfontname ()

Retrieve the name of Max's system font.

Returns

The name of Max's system font.

36.57.3.23 const char* systemfontname_bold ()

Retrieve the name of Max's bold system font.

Returns

The name of Max's bold system font.

36.57.3.24 const char* systemfontname_light ()

Retrieve the name of Max's light system font.

Returns

The name of Max's light system font.

36.57.3.25 `t_symbol* systemfontsym()`

Retrieve the name of Max's system font as a symbol.

Returns

The name of Max's system font.

36.58 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.

36.58.1 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.

36.58.2 Function Documentation

36.58.2.1 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

<i>x</i>	
<i>xx</i>	
<i>yx</i>	
<i>xy</i>	
<i>yy</i>	
<i>x0</i>	
<i>y0</i>	

Remarks

given *x,y* the matrix specifies the following transformation:

```
1 xnew = xx * x + xy * y + x0;
2 ynew = yx * x + yy * y + y0;
```

36.58.2.2 void jgraphics_matrix_init_identity (*t_jmatrix* * *x*)

Modify a matrix to be an identity transform.

Parameters

<i>x</i>	The <i>t_jmatrix</i> .
----------	------------------------

36.58.2.3 void jgraphics_matrix_init_rotate (*t_jmatrix* * *x*, double *radians*)

Initialize a *t_jmatrix* to rotate (offset) a point.

Parameters

<i>x</i>	The <i>t_jmatrix</i> .
<i>radians</i>	The angle or rotation in radians.

36.58.2.4 void jgraphics_matrix_init_scale (*t_jmatrix* * *x*, double *sx*, double *sy*)

Initialize a *t_jmatrix* to scale (offset) a point.

Parameters

<i>x</i>	The <i>t_jmatrix</i> .
<i>sx</i>	The horizontal scale factor.
<i>sy</i>	The vertical scale factor.

36.58.2.5 void jgraphics_matrix_init_translate (*t_jmatrix* * *x*, double *tx*, double *ty*)

Initialize a *t_jmatrix* to translate (offset) a point.

Parameters

<i>x</i>	The t_jmatrix .
<i>tx</i>	The amount of x-axis translation.
<i>ty</i>	The amount of y-axis translation.

36.58.2.6 void jgraphics_matrix_invert(t_jmatrix * *x*)

Invert an existing matrix.

Parameters

<i>x</i>	The t_jmatrix .
----------	---------------------------------

36.58.2.7 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

<i>result</i>	The resulting product t_jmatrix .
<i>a</i>	The first operand.
<i>b</i>	The second operand.

36.58.2.8 void jgraphics_matrix_rotate(t_jmatrix * *x*, double *radians*)

Apply a rotation to an existing matrix.

Parameters

<i>x</i>	The t_jmatrix .
<i>radians</i>	The angle or rotation in radians.

36.58.2.9 void jgraphics_matrix_scale(t_jmatrix * *x*, double *sx*, double *sy*)

Apply a scaling to an existing matrix.

Parameters

<i>x</i>	The t_jmatrix .
<i>sx</i>	The horizontal scale factor.
<i>sy</i>	The vertical scale factor.

36.58.2.10 void jgraphics_matrix_transform_point(const t_jmatrix * *matrix*, double * *x*, double * *y*)

Transform a point using a [t_jmatrix](#) transformation.

Parameters

<i>matrix</i>	The t_jmatrix .
<i>x</i>	The address of the variable holding the x coordinate.

<i>y</i>	The address of the variable holding the y coordinate.
----------	---

36.58.2.11 void jgraphics_matrix_translate (*t_jmatrix* * *x*, double *tx*, double *ty*)

Apply a translation to an existing matrix.

Parameters

<i>x</i>	The t_jmatrix .
<i>tx</i>	The amount of x-axis translation.
<i>ty</i>	The amount of y-axis translation.

36.59 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.

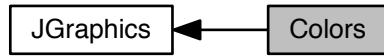
36.59.1 Detailed Description

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()`.

36.60 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_rgba](#) *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_rgba](#) *c)
Set the components of a color by providing an array of atoms.
- void [jrgba_set](#) ([t_rgba](#) *prgba, double r, double g, double b, double a)
Set the components of a color.
- void [jrgba_copy](#) ([t_rgba](#) *dest, [t_rgba](#) *src)
Copy a color.
- long [jrgba_compare](#) ([t_rgba](#) *rgba1, [t_rgba](#) *rgba2)
Compare two colors for equality.
- [t_max_err_jrgba_attr_get](#) ([t_rgba](#) *jrgba, long *argc, [t_atom](#) **argv)
Get the value of a [t_rgba](#) struct, returned as an array of atoms with the values for each component.
- [t_max_err_jrgba_attr_set](#) ([t_rgba](#) *jrgba, long argc, [t_atom](#) *argv)
Set the value of a [t_rgba](#) struct, given an array of atoms with the values to use.

36.60.1 Detailed Description

36.60.2 Function Documentation

36.60.2.1 [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

<i>argc</i>	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.
<i>argv</i>	The address to the first of an array of atoms that define the color.
<i>c</i>	The address of a t_jrgba struct for which the color will be defined.

Returns

A Max error code.

36.60.2.2 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

<i>jrgba</i>	The color struct whose color will be retrieved.
<i>argc</i>	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.
<i>argv</i>	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.

36.60.2.3 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

<i>jrgba</i>	The color struct whose color will be set.
<i>argc</i>	The number of atoms in the array. This must be 4.
<i>argv</i>	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.

36.60.2.4 long jrgba_compare (t_jrgba * rgba1, t_jrgba * rgba2)

Compare two colors for equality.

Parameters

<i>rgba1</i>	The address of a t_jrgba struct to compare.
<i>rgba2</i>	The address of another t_jrgba struct to compare.

Returns

returns 1 if *rgba1* == *rgba2*.

36.60.2.5 void jrgba_copy ([t_jrgba](#) * *dest*, [t_jrgba](#) * *src*)

Copy a color.

Parameters

<i>dest</i>	The address of a t_jrgba struct to which the color will be copied.
<i>src</i>	The address of a t_jrgba struct from which the color will be copied.

36.60.2.6 void jrgba_set ([t_jrgba](#) * *prgba*, double *r*, double *g*, double *b*, double *a*)

Set the components of a color.

Parameters

<i>prgba</i>	The address of a t_jrgba struct for which the color will be defined.
<i>r</i>	The value of the red component in a range of [0.0, 1.0].
<i>g</i>	The value of the green component in a range of [0.0, 1.0].
<i>b</i>	The value of the blue component in a range of [0.0, 1.0].
<i>a</i>	The value of the alpha component in a range of [0.0, 1.0].

36.60.2.7 void jrgba_to_atoms ([t_jrgba](#) * *c*, [t_atom](#) * *argv*)

Get the components of a color in an array of pre-allocated atoms.

Parameters

<i>argv</i>	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.
<i>c</i>	The address of a t_jrgba struct from which the color components will be fetched.

36.61 Styles

Styles provide a means by which to inherit attribute values from a patcher that are consistently used across many objects.

Collaboration diagram for Styles:



Macros

- `#define FILL_ATTR_SAVE`
Flag indicating we want this fill attribute saved (creates attrs).
- `#define CLASS_ATTR_STYLE_RGBA_NOSAVE(c, attrname, flags, structname, structmember, label)`
Define an RGBA style attribute with standard settings.
- `#define CLASS_ATTR_STYLE_RGBA(c, attrname, flags, structname, structmember, label)`
Define an RGBA style attribute with standard settings.
- `#define CLASS_ATTR_STYLE_RGBA_PREVIEW(c, attrname, flags, structname, structmember, label, previewtype)`
Define an RGBA style attribute with standard settings.
- `#define CLASS_ATTR_STYLE_ALIAS_NOSAVE(c, attrname, aliasname)`
Define an unsaved alias.
- `#define CLASS_ATTR_STYLE_ALIAS_COMPATIBILITY(c, attrname, aliasname)`
Define a Max 5/6 saved compatibility alias.
- `#define CLASS_ATTR_STYLE_ALIAS_RGBA_LEGACY(c, attrname, aliasname)`
Define a Max 4 legacy RGB attribute alias.

Functions

- `BEGIN_USING_C_LINKAGE void class_attr_setstyle (t_class *c, const char *s)`
Add an attribute to the current style.
- `void class_attr_style_alias (t_class *c, const char *name, const char *aliasname, long legacy)`
Add an alias to the current style.
- `void class_attr_setfill (t_class *c, const char *name, long flags)`
Specify that an attr requires a fill.
- `void jgraphics_attr_fillrect (t_object *b, t_jgraphics *g, t_symbol *attrname, t_rect *area)`
Fill using the current value of a named style color that exists either in the object or the defined style.
- `t_jpattern * jgraphics_attr_setfill (t_object *b, t_jgraphics *g, t_symbol *attrname, t_rect *area)`
Fill using the current value of a named style color that exists either in the object or the defined style.
- `void object_attr_getfillcolor_atposition (t_object *b, const char *attrname, double pos, t_jrgba *c)`
Determine the color at a given position in a fill.
- `long object_attr_getfill (t_object *obj, t_symbol *attrname)`
Determine if an attribute is a fill.
- `void class_attr_stylemap (t_class *c, char *attrname, char *mapname)`
Associate the name of an attribute of your class with the name of an attribute of a style.

36.61.1 Detailed Description

Styles provide a means by which to inherit attribute values from a patcher that are consistently used across many objects.

36.61.2 Macro Definition Documentation

36.61.2.1 `#define CLASS_ATTR_STYLE_ALIAS_COMPATIBILITY(c, attrname, aliasname)`

Define a Max 5/6 saved compatibility alias.

Parameters

<i>c</i>	The class whose attribute will be added to the style.
<i>attrname</i>	The name of the attribute of your class.
<i>aliasname</i>	The name of the alias.

36.61.2.2 `#define CLASS_ATTR_STYLE_ALIAS_NOSAVE(c, attrname, aliasname)`

Define an unsaved alias.

Parameters

<i>c</i>	The class whose attribute will be added to the style.
<i>attrname</i>	The name of the attribute of your class.
<i>aliasname</i>	The name of the alias.

See also

'jslider' example project in the SDK.

36.61.2.3 `#define CLASS_ATTR_STYLE_ALIAS_RGBA_LEGACY(c, attrname, aliasname)`

Define a Max 4 legacy RGB attribute alias.

Parameters

<i>c</i>	The class whose attribute will be added to the style.
<i>attrname</i>	The name of the attribute of your class.
<i>aliasname</i>	The name of the alias.

See also

'jslider' example project in the SDK.

36.61.2.4 `#define CLASS_ATTR_STYLE_RGBA(c, attrname, flags, structname, structmember, label)`

Define an RGBA style attribute with standard settings.

Parameters

<i>c</i>	The class whose attribute will be added to the style.
----------	---

<i>attrname</i>	The name of the attribute of your class.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>label</i>	A human-friendly label for the Max inspector.

See also

[CLASS_ATTR_STYLE_RGBA_NOSAVE](#) is a variant that does not save the state with the [Patcher](#).

[CLASS_ATTR_STYLE_RGBA_PREVIEW](#) is a variant that provides a style preview.

[class_attr_setstyle\(\)](#) is the lower level function used to provide the style part of the attribute definition.

36.61.2.5 #define CLASS_ATTR_STYLE_RGBA_NOSAVE(*c*, *attrname*, *flags*, *structname*, *structmember*, *label*)

Define an RGBA style attribute with standard settings.

Parameters

<i>c</i>	The class whose attribute will be added to the style.
<i>attrname</i>	The name of the attribute of your class.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>label</i>	A human-friendly label for the Max inspector.

See also

[CLASS_ATTR_STYLE_RGBA](#) is a variant that saves the state with the [Patcher](#).

[CLASS_ATTR_STYLE_RGBA_PREVIEW](#) is a variant that provides a style preview.

[class_attr_setstyle\(\)](#) is the lower level function used to provide the style part of the attribute definition.

36.61.2.6 #define CLASS_ATTR_STYLE_RGBA_PREVIEW(*c*, *attrname*, *flags*, *structname*, *structmember*, *label*, *previewtype*)

Define an RGBA style attribute with standard settings.

Parameters

<i>c</i>	The class whose attribute will be added to the style.
<i>attrname</i>	The name of the attribute of your class.
<i>flags</i>	Any flags you wish to declare for this attribute, as defined in e_max_attrflags .
<i>structname</i>	The C identifier for the struct (containing a valid t_object header) representing an instance of this class.
<i>structmember</i>	The C identifier of the member in the struct that holds the value of this attribute.
<i>label</i>	A human-friendly label for the Max inspector.
<i>previewtype</i>	Type of preview to use in the style bar, e.g. "triangle_fill"

See also

[CLASS_ATTR_STYLE_RGBA_NOSAVE](#) is a variant that does not save the state with the [Patcher](#).

[CLASS_ATTR_STYLE_RGBA](#) is a variant that saves the state with the [Patcher](#) but does not provide the preview.

[class_attr_setstyle\(\)](#) is the lower level function used to provide the style part of the attribute definition.

36.61.2.7 #define FILL_ATTR_SAVE

Flag indicating we want this fill attribute saved (creates attrs).

See also

[class_attr_setfill\(\)](#)

The 'uitextfield' example project in the SDK.

36.61.3 Function Documentation

36.61.3.1 void class_attr_setfill (*t_class * c*, *const char * name*, *long flags*)

Specify that an attr requires a fill.

Parameters

<i>c</i>	The class whose attribute is a fill.
<i>name</i>	The name of the attribute.
<i>flags</i>	0 for none, or FILL_ATTR_SAVE .

See also

The 'uitextfield' example project in the SDK.

36.61.3.2 BEGIN_USING_C_LINKAGE void class_attr_setstyle (*t_class * c*, *const char * s*)

Add an attribute to the current style.

Parameters

<i>c</i>	The class whose attribute will be added to the style.
<i>s</i>	The name of the attribute to be added to the style.

See also

The 'uitextfield' example project in the SDK.

36.61.3.3 void class_attr_style_alias (*t_class * c*, *const char * name*, *const char * aliasname*, *long legacy*)

Add an alias to the current style.

This is used for backward compatibility where an attribute using an old name will want a style applied to it from a different name. Typically you will use one of the macros such as [CLASS_ATTR_STYLE_ALIAS_NOSAVE](#) rather than using this function directly.

Parameters

<i>c</i>	The class for whom the alias will be created.
<i>name</i>	The name of the attribute of the style.
<i>aliasname</i>	The name of the alias.
<i>legacy</i>	Always pass 0 for this argument.

See also

[CLASS_ATTR_STYLE_ALIAS_NOSAVE](#)

The 'slider' project in the SDK.

36.61.3.4 void **class_attr_stylemap**(**t_class** * *c*, char * *attrname*, char * *mapname*)

Associate the name of an attribute of your class with the name of an attribute of a style.

Parameters

<i>c</i>	The class whose attribute will be added to the style.
<i>attrname</i>	The name of the attribute of your class.
<i>mapname</i>	The name of the attribute from the style.

See also

'jslider' example project in the SDK.

36.61.3.5 void jgraphics_attr_fillrect (*t_object* * *b*, *t_jgraphics* * *g*, *t_symbol* * *attrname*, *t_rect* * *area*)

Fill using the current value of a named style color that exists either in the object or the defined style.

Abridged example from the 'attrui' object:

```

1 long is_fill = object_attr_getfill(destination, x->j_attr);
2
3 if (is_fill) {
4     jgraphics_attr_fillrect((t_object *)destination, g, x->j_attr, rect);
5     jgraphics_rectangle(g, rect->x, rect->y, rect->width, rect->height);    // ready to be stroked
6 }
7 else {
8     object_attr_getjrgba(destination, x->j_attr, &color);
9     jgraphics_set_source_jrgba(g, &color);
10    jgraphics_rectangle(g, rect->x, rect->y, rect->width, rect->height);
11    jgraphics_fill_preserve(g);
12 }
13
14 jgraphics_set_source_jrgba(g, &bordercolor);
15 jgraphics_stroke(g);

```

Parameters

<i>b</i>	The instance of your object.
<i>g</i>	The jgraphics context.
<i>attrname</i>	The name of the attribute whose fill style you want.
<i>area</i>	The rect area to be filled.

36.61.3.6 *t_jpattern** jgraphics_attr_setfill (*t_object* * *b*, *t_jgraphics* * *g*, *t_symbol* * *attrname*, *t_rect* * *area*)

Fill using the current value of a named style color that exists either in the object or the defined style.

Example from the 'panel' object:

```

1 t_rect r;
2
3 r.x = r.y = thick * 0.5;
4 r.width = rect->width - thick;
5 r.height = rect->height - thick;
6
7 pat = jgraphics_attr_setfill((t_object *)x, g, ps_bgfillcolor, &r);
8 jgraphics_rectangle_rounded(g, r.x, r.y, r.width, r.height, round, round);
9 jgraphics_fill_preserve(g);
10 jgraphics_pattern_destroy(pat);
11
12 object_attr_getjrgba(x, ps_bordercolor, &color);
13 jgraphics_set_source_jrgba(g, &color);
14 jgraphics_set_line_width(g, thick);
15 jgraphics_stroke(g);

```

Parameters

<i>b</i>	The instance of your object.
<i>g</i>	The jgraphics context.
<i>attrname</i>	The name of the attribute whose fill style you want.
<i>area</i>	The rect area to be filled.

Returns

The pattern.

36.61.3.7 long object_attr_getfill (t_object * *obj*, t_symbol * *attrname*)

Determine if an attribute is a fill.

Parameters

<i>obj</i>	The instance of your object.
<i>attrname</i>	The name of the attribute to query.

Returns

true if the object is a fill, otherwise false.

36.61.3.8 void object_attr_getfillcolor_atposition (t_object * *b*, const char * *attrname*, double *pos*, t_jrgba * *c*)

Determine the color at a given position in a fill.

Parameters

<i>b</i>	The instance of your object.
<i>attrname</i>	The name of the attribute to query.
<i>pos</i>	The position in a range of [0.0, 1.0].
<i>c</i>	A valid t_jrgba whose members will be filled-in upon return.

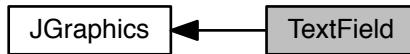
Returns

true if the object is a fill, otherwise false.

36.62 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)`
Set the 'wordwrap' attribute of a textfield.
- `t_max_err textfield_set_useellipsis (t_object *tf, char c)`
Return the value of the 'useellipsis' attribute of a textfield.
- `t_max_err textfield_get_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.

36.62.1 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](#)

36.62.2 Function Documentation

36.62.2.1 `char textfield_get_autoscroll (t_object * tf)`

Return the value of the 'autoscroll' attribute of a textfield.

Parameters

<code>tf</code>	The textfield instance pointer.
-----------------	---------------------------------

Returns

A value of the attribute.

36.62.2.2 `t_max_err textfield_get_bgcolor (t_object * tf, t_jrgba * prgba)`

Retrieve the background color of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
<i>prgba</i>	The address of a valid t_jrgba whose values will be filled-in upon return.

Returns

A Max error code.

36.62.2.3 char textfield_get_editonclick (t_object * tf)

Return the value of the 'editonclick' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
-----------	---------------------------------

Returns

A value of the attribute.

36.62.2.4 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

<i>tf</i>	The textfield instance pointer.
-----------	---------------------------------

Returns

The current text used as the empty text.

36.62.2.5 char textfield_get_noactivate (t_object * tf)

Return the value of the 'noactivate' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
-----------	---------------------------------

Returns

A value of the attribute.

36.62.2.6 t_object* textfield_get_owner (t_object * tf)

Return the object that owns a particular textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
-----------	---------------------------------

Returns

A pointer to the owning object.

36.62.2.7 char textfield_get_READONLY (*t_object* * *tf*)

Return the value of the 'readonly' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
-----------	---------------------------------

Returns

A value of the attribute.

36.62.2.8 char textfield_get_SELECTALLONEDIT (*t_object* * *tf*)

Return the value of the 'selectallonedit' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
-----------	---------------------------------

Returns

A value of the attribute.

36.62.2.9 *t_max_err* textfield_get_textcolor (*t_object* * *tf*, *t_jrgba* * *prgba*)

Retrieve the color of the text in a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
<i>prgba</i>	The address of a valid t_jrgba whose values will be filled-in upon return.

Returns

A Max error code.

36.62.2.10 *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

<i>tf</i>	The textfield instance pointer.
-----------	---------------------------------

<i>pleft</i>	The address of a variable to hold the value of the left margin upon return.
<i>ptop</i>	The address of a variable to hold the value of the top margin upon return.
<i>pright</i>	The address of a variable to hold the value of the right margin upon return.
<i>pbottom</i>	The address of a variable to hold the value of the bottom margin upon return.

Returns

A Max error code.

36.62.2.11 char textfield_get_underline (t_object * tf)

Return the value of the 'underline' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
-----------	---------------------------------

Returns

A value of the attribute.

36.62.2.12 char textfield_get_useellipsis (t_object * tf)

Return the value of the 'useellipsis' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
-----------	---------------------------------

Returns

A value of the attribute.

36.62.2.13 char textfield_get_wantsreturn (t_object * tf)

Return the value of the 'wantsreturn' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
-----------	---------------------------------

Returns

A value of the attribute.

36.62.2.14 char textfield_get_wantstab (t_object * tf)

Return the value of the 'wantstab' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
-----------	---------------------------------

Returns

A value of the attribute.

36.62.2.15 char textfield_get_wordwrap (*t_object* * *tf*)

Return the value of the 'wordwrap' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
-----------	---------------------------------

Returns

A value of the attribute.

36.62.2.16 *t_max_err* textfield_set_autoscroll (*t_object* * *tf*, char *c*)

Set the 'autoscroll' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
<i>c</i>	The new value for the attribute.

Returns

A Max error code.

36.62.2.17 *t_max_err* textfield_set_bgcolor (*t_object* * *tf*, *t_jrgba* * *prgba*)

Set the background color of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
<i>prgba</i>	The address of a t_jrgba containing the new color to use.

Returns

A Max error code.

36.62.2.18 *t_max_err* textfield_set_editonclick (*t_object* * *tf*, char *c*)

Set the 'editonclick' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
-----------	---------------------------------

<i>c</i>	The new value for the attribute.
----------	----------------------------------

Returns

A Max error code.

36.62.2.19 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

<i>tf</i>	The textfield instance pointer.
<i>txt</i>	A symbol containing the new text to display when the textfield has no content.

Returns

A Max error code.

36.62.2.20 t_max_err textfield_set_noactivate (t_object * *tf*, char *c*)

Set the 'noactivate' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
<i>c</i>	The new value for the attribute.

Returns

A Max error code.

36.62.2.21 t_max_err textfield_set_READONLY (t_object * *tf*, char *c*)

Set the 'readonly' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
<i>c</i>	The new value for the attribute.

Returns

A Max error code.

36.62.2.22 t_max_err textfield_set_SELECTALLONEDIT (t_object * *tf*, char *c*)

Set the 'selectallonedit' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
<i>c</i>	The new value for the attribute.

Returns

A Max error code.

36.62.2.23 t_max_err textfield_set_textcolor (t_object * tf, t_jrgba * prgba)

Set the color of the text in a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
<i>prgba</i>	The address of a t_jrgba containing the new color to use.

Returns

A Max error code.

36.62.2.24 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

<i>tf</i>	The textfield instance pointer.
<i>left</i>	The new value for the left margin.
<i>top</i>	The new value for the top margin.
<i>right</i>	The new value for the right margin.
<i>bottom</i>	The new value for the bottom margin.

Returns

A Max error code.

36.62.2.25 t_max_err textfield_set_underline (t_object * tf, char c)

Set the 'underline' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
<i>c</i>	The new value for the attribute.

Returns

A Max error code.

36.62.2.26 t_max_err textfield_set_useellipsis (t_object * tf, char c)

Set the 'useellipsis' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
<i>c</i>	The new value for the attribute.

Returns

A Max error code.

36.62.2.27 t_max_err textfield_set_wantsreturn (t_object * tf, char c)

Set the 'wantsreturn' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
<i>c</i>	The new value for the attribute.

Returns

A Max error code.

36.62.2.28 t_max_err textfield_set_wantstab (t_object * tf, char c)

Set the 'wantstab' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
<i>c</i>	The new value for the attribute.

Returns

A Max error code.

36.62.2.29 t_max_err textfield_set_wordwrap (t_object * tf, char c)

Set the 'wordwrap' attribute of a textfield.

Parameters

<i>tf</i>	The textfield instance pointer.
<i>c</i>	The new value for the attribute.

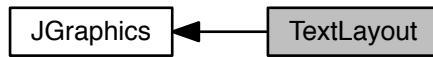
Returns

A Max error code.

36.63 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`, `JGRAPHICS_TEXTLAYOUT_USEELLIPSIS` }

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_settext (t_jtextlayout *textlayout, const char *utf8, t_jfont *jfont)`
Set the text 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_measuretext (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.
- `t_jpath * jtextlayout_createpath (t_jtextlayout *tl)`
Create a `t_jpath` object representing the text layout.

36.63.1 Detailed Description

A textlayout is lower-level text rendering object used by higher-level entities such as [TextField](#).

36.63.2 Enumeration Type Documentation

36.63.2.1 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)

36.63.3 Function Documentation

36.63.3.1 t_jtextlayout* jtextlayout_create()

Create a new textlayout object.

Returns

The new textlayout object.

36.63.3.2 t_jpath* jtextlayout_createpath(t_jtextlayout * tl)

Create a t_jpath object representing the text layout.

Parameters

<i>tl</i>	The textlayout object to retrieve a path for.
-----------	---

Returns

A t_jpath. When finished with the path free it with `jgraphics_path_destroy`.

36.63.3.3 void jtextlayout_destroy(t_jtextlayout * textlayout)

Release/free a textlayout object.

Parameters

<i>textlayout</i>	The textlayout object to release.
-------------------	-----------------------------------

36.63.3.4 void jtextlayout_draw(t_jtextlayout * tl, t_jgraphics * g)

Draw a textlayout in a given graphics context.

Parameters

<i>tl</i>	The textlayout object to query.
<i>g</i>	The graphics context in which to draw the text.

36.63.3.5 `t_max_err jtextlayout_getchar (t_jtextlayout * tl, long index, long * pch)`

Retrieve the unicode character at a given index.

Parameters

<i>tl</i>	The textlayout object to query.
<i>index</i>	The index from which to fetch the unicode character.
<i>pch</i>	The address of a variable to hold the unicode character value upon return.

Returns

A Max error code.

36.63.3.6 `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

<i>tl</i>	The textlayout object to query.
<i>index</i>	The index from which to fetch the unicode character.
<i>rect</i>	The address of a valid <code>t_rect</code> which will be filled in upon return.

Returns

A Max error code.

36.63.3.7 `long jtextlayout_getnumchars (t_jtextlayout * tl)`

Retrieve a count of the number of characters in a textlayout object.

Parameters

<i>tl</i>	The textlayout object to query.
-----------	---------------------------------

Returns

The number of characters.

36.63.3.8 `void jtextlayout_measuretext (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

<i>textlayout</i>	The textlayout object to query.
-------------------	---------------------------------

<i>startindex</i>	You can measure a subset of the characters. This defines the character from which to start.
<i>numchars</i>	Pass -1 for all characters from startindex to end
<i>includewhitespace</i>	Define whether to measure with or without whitespace truncated from edges.
<i>width</i>	Returns the width of text not including any margins.
<i>height</i>	Returns the height of text not including any margins.
<i>numlines</i>	Returns the number of lines of text.

36.63.3.9 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

<i>textlayout</i>	The textlayout object.
<i>utf8</i>	The text to render.
<i>jfont</i>	The font with which to render the text.
<i>x</i>	The text is placed within rect specified by x, y, width, height.
<i>y</i>	The text is placed within rect specified by x, y, width, height.
<i>width</i>	The text is placed within rect specified by x, y, width, height.
<i>height</i>	The text is placed within rect specified by x, y, width, height.
<i>justification</i>	How to justify the text within the rect.
<i>flags</i>	Additional flags to control behaviour.

36.63.3.10 void **jtextlayout_settext** (*t_jtextlayout* * *textlayout*, const char * *utf8*, *t_jfont* * *jfont*)

Set the text of a textlayout object.

Parameters

<i>textlayout</i>	The textlayout object.
<i>utf8</i>	The text to render.
<i>jfont</i>	The font with which to render the text.

36.63.3.11 void **jtextlayout_settextcolor** (*t_jtextlayout* * *textlayout*, *t_jrgba* * *textcolor*)

Set the color to render text in a textlayout object.

Parameters

<i>textlayout</i>	The textlayout object for which to set the color.
<i>textcolor</i>	The color for the text.

36.63.3.12 *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

<i>g</i>	The graphics context for the textlayout.
<i>bgcolor</i>	The background color for the textlayout.

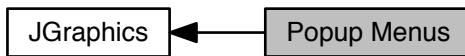
Returns

The new textlayout object.

36.64 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 jpopupmenuSetFont (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.
- `int jpopupmenu_popup_belowrect (t_jpopupmenu *menu, t_rect rect, int defitemid)`
Tell a menu to display below a given rectangle in a patcher.
- `int jpopupmenu_popup_aboverect (t_jpopupmenu *menu, t_rect rect, int defitemid)`
Tell a menu to display above a given rectangle in a patcher.

36.64.1 Detailed Description

Popup menu API so externals can create popup menus that can also be drawn into.

36.64.2 Function Documentation

36.64.2.1 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

<i>menu</i>	The pop-up menu to which the item will be added.
<i>itemid</i>	Each menu item should be assigned a unique integer id using this parameter.
<i>utf8Text</i>	The text to display in for the menu item.
<i>textColor</i>	The color to use for the menu item, or NULL to use the default color.
<i>checked</i>	A non-zero value indicates that the item should have a check-mark next to it.
<i>disabled</i>	A non-zero value indicates that the item should be disabled.
<i>icon</i>	A <i>t_jsurface</i> will be used as an icon for the menu item if provided here. Pass NULL for no icon.

36.64.2.2 void jpopupmenu_addseperator (*t_jpopupmenu* * *menu*)

Add a separator to a pop-menu.

Parameters

<i>menu</i>	The pop-up menu to which the separator will be added.
-------------	---

36.64.2.3 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

<i>menu</i>	The pop-up menu to which a menu will be added as a submenu.
<i>utf8Name</i>	The name of the menu item.
<i>submenu</i>	The pop-up menu which will be used as the submenu.
<i>disabled</i>	Pass a non-zero value to disable the menu item.

36.64.2.4 void jpopupmenu_clear (*t_jpopupmenu* * *menu*)

Clear the contents of a pop-up menu.

Parameters

<i>menu</i>	The pop-up menu whose contents will be cleared.
-------------	---

36.64.2.5 *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.

36.64.2.6 void **jpopupmenu_destroy** (**t_jpopupmenu** * *menu*)

Free a pop-up menu created with [jpopupmenu_create\(\)](#).

Parameters

<i>menu</i>	The pop-up menu to be freed.
-------------	------------------------------

36.64.2.7 int jpopupmenu_popup (*t_jpopupmenu* * *menu*, *t_pt* *screen*, int *defitemid*)

Tell a menu to display at a specified location.

Parameters

<i>menu</i>	The pop-up menu to display.
<i>screen</i>	The point at which to display in screen coordinates.
<i>defitemid</i>	The initially choosen item id.

Returns

The item id for the item in the menu choosen by the user.

36.64.2.8 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

<i>menu</i>	The pop-up menu to display.
<i>box</i>	The box above which to display the menu.
<i>view</i>	The patcherview for the box in which to display the menu.
<i>offset</i>	An offset from the box position at which to display the menu.
<i>defitemid</i>	The initially choosen item id.

Returns

The item id for the item in the menu choosen by the user.

36.64.2.9 int jpopupmenu_popup_aboverect (*t_jpopupmenu* * *menu*, *t_rect* *rect*, int *defitemid*)

Tell a menu to display above a given rectangle in a patcher.

Parameters

<i>menu</i>	The pop-up menu to display.
<i>rect</i>	The rectangle above which to display the menu.
<i>defitemid</i>	The initially choosen item id.

Returns

The item id for the item in the menu choosen by the user.

36.64.2.10 int jpopupmenu_popup_belowrect (*t_jpopupmenu* * *menu*, *t_rect* *rect*, int *defitemid*)

Tell a menu to display below a given rectangle in a patcher.

Parameters

<i>menu</i>	The pop-up menu to display.
<i>rect</i>	The rectangle below which to display the menu.
<i>defitemid</i>	The initially choosen item id.

Returns

The item id for the item in the menu choosen by the user.

36.64.2.11 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

<i>menu</i>	The pop-up menu to display.
<i>box</i>	The box above which to display the menu.
<i>view</i>	The patcherview for the box in which to display the menu.
<i>defitemid</i>	The initially choosen item id.

Returns

The item id for the item in the menu choosen by the user.

36.64.2.12 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.

Parameters

<i>menu</i>	The pop-up menu to which the colors will be applied.
<i>text</i>	The text color for menu items.
<i>bg</i>	The background color for menu items.
<i>highlightedtext</i>	The text color for the highlighted menu item.
<i>highlightedb</i>	The background color the highlighted menu item.

36.64.2.13 void jpopupmenu_setfont (t_jpopupmenu * menu, t_jfont * font)

Set the font used by a pop-up menu.

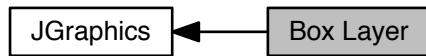
Parameters

<i>menu</i>	The pop-up menu whose font will be set.
<i>font</i>	A pointer to a font object, whose font info will be copied to the pop-up menu.

36.65 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.

36.65.1 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
    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.

36.65.2 Function Documentation

36.65.2.1 `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

<i>b</i>	The object/box for the layer opened by jbox_start_layer() .
<i>view</i>	The patcherview for the object opened by jbox_start_layer() .
<i>name</i>	The name of the layer.

Returns

A Max error code.

36.65.2.2 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

<i>b</i>	The object/box to invalidate.
<i>view</i>	The patcherview for the object which should be invalidated, or NULL for all patcherviews.
<i>name</i>	The name of the layer to invalidate.

Returns

A Max error code.

36.65.2.3 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

<i>b</i>	The object/box to be painted.
<i>view</i>	The patcherview for the object which should be painted, or NULL for all patcherviews.
<i>name</i>	The name of the layer to paint.
<i>x</i>	The x-coordinate for the position at which to paint the layer.
<i>y</i>	The y-coordinate for the position at which to paint the layer.

Returns

A Max error code.

36.65.2.4 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

<i>b</i>	The object/box to which the layer is attached.
<i>view</i>	The patcherview for the object to which the layer is attached.
<i>name</i>	A name for this layer.
<i>width</i>	The width of the layer.
<i>height</i>	The height of the layer.

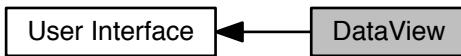
Returns

A [t_jgraphics](#) context for drawing into the layer.

36.66 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.

36.66.1 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.

36.66.2 Function Documentation

36.66.2.1 `t_object* jdatatable_getclient (t_object * dv)`

Get a pointer to a datatable's client.

The client is the object to which the datatable will send messages to get data, notify of changes to cells, etc.

Parameters

<i>dv</i>	The dataview instance.
-----------	------------------------

Returns

A pointer to the dataview's client object.

36.66.2.2 void* jdataview_new (void)

Create a dataview.

You should free it with [object_free\(\)](#).

Returns

A pointer to the new instance.

36.66.2.3 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

<i>dv</i>	The dataview instance.
<i>client</i>	The object to be assigned as the dataview's client.

36.67 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_unicodetouft8](#) (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.

36.67.1 Detailed Description

36.67.2 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;

36.67.2.1 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, &cvtbuffer, &
        cvtbuflen) == ERR_NONE) {
        // do something with the converted buffer
        sysmem_freeptr(cvtbuffer); // free newly allocated data buffer
    }
    object_free(conv); // free converter
}
```

36.67.3 Function Documentation

36.67.3.1 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

<i>src_encoding</i>	The name encoding of the input.
<i>in</i>	The input string.
<i>inbytes</i>	The number of bytes in the input string.
<i>dest_encoding</i>	The name of the encoding to use for the output.
<i>out</i>	The address of a char*, which will be allocated and filled with the string in the new encoding.
<i>outbytes</i>	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.

36.67.3.2 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

<i>s</i>	An array of wide (16-bit) unicode characters.
<i>len</i>	The length of <i>s</i> .
<i>outlen</i>	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.

36.67.3.3 long charset_utf8_count (char * utf8, long * bytecount)

Returns utf8 character count, and optionally bytecount.

Parameters

<i>utf8</i>	The UTF-8 encoded string whose characters are to be counted.
<i>bytecount</i>	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.

36.67.3.4 char* charset_utf8_offset (char * *utf8*, long *charoffset*, long * *byteoffset*)

Returns utf8 character offset (positive or negative), and optionally byte offset.

Parameters

<i>utf8</i>	A UTF-8 encoded string.
<i>charoffset</i>	The char offset into the string at which to find the byte offset.
<i>byteoffset</i>	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.

36.67.3.5 unsigned short* charset_utf8tounicode (char * *s*, long * *outlen*)

Convert a UTF8 C-String into a 16-bit-wide-character array.

Parameters

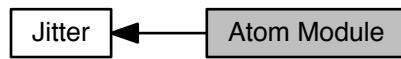
<i>s</i>	The string to be converted to unicode.
<i>outlen</i>	The address of a variable to hold the size of the number of chars but does not include the NULL terminator in the count.

Returns

A pointer to the buffer of unicode (wide) characters.

36.68 Atom Module

Collaboration diagram for Atom Module:



Functions

- `t_jit_err jit_atom_setlong (t_atom *a, t_atom_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.
- `t_atom_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 (t_atom_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.

36.68.1 Detailed Description

36.68.2 Function Documentation

36.68.2.1 `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

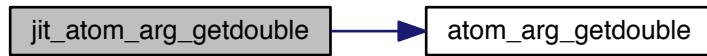
<i>c</i>	pointer to double (should contain desired default)
<i>idx</i>	atom argument index
<i>ac</i>	atom argument count
<i>av</i>	atom argument vector

Returns

`t_jit_err` error code. `JIT_ERR_NONE` if successful.

References `atom_arg_getdouble()`.

Here is the call graph for this function:



36.68.2.2 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

<i>c</i>	pointer to float (should contain desired default)
<i>idx</i>	atom argument index
<i>ac</i>	atom argument count
<i>av</i>	atom argument vector

Returns

`t_jit_err` error code. `JIT_ERR_NONE` if successful.

References `atom_arg_getfloat()`.

Here is the call graph for this function:



36.68.2.3 long jit_atom_arg_getlong (*t_atom_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

<i>c</i>	pointer to long (should contain desired default)
<i>idx</i>	atom argument index
<i>ac</i>	atom argument count
<i>av</i>	atom argument vector

Returns

t_jit_err error code. JIT_ERR_NONE if successful.

References atom_arg_getlong().

Referenced by max_jit_mop_matrix_args().

Here is the call graph for this function:



36.68.2.4 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

<i>c</i>	pointer to symbol pointer (should contain desired default)
<i>idx</i>	atom argument index
<i>ac</i>	atom argument count
<i>av</i>	atom argument vector

Returns

`t_jit_err` error code. `JIT_ERR_NONE` if successful.

References `atom_arg_getsym()`.

Referenced by `max_jit_mop_matrix_args()`.

Here is the call graph for this function:

**36.68.2.5 long jit_atom_getcharfix (`t_atom *a`)**

Retrieves atom value as an 8 bit fixed point number.

Parameters

<code>a</code>	atom pointer
----------------	--------------

Returns

8 bit fixed point value in the range 0-255. 0 if atom has no numeric value.

References `atom_getcharfix()`.

Referenced by `jit_matrix_fillplane()`, `jit_matrix_setall()`, and `jit_matrix_setcell()`.

Here is the call graph for this function:

**36.68.2.6 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.

References atom_getfloat().

Referenced by jit_matrix_fillplane(), jit_matrix_setall(), and jit_matrix_setcell().

Here is the call graph for this function:

**36.68.2.7 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.

Referenced by jit_matrix_exprfill(), jit_matrix_fillplane(), jit_matrix_getcell(), jit_matrix_setall(), jit_matrix_setcell(), jit_matrix_setcell1d(), jit_matrix_setcell2d(), jit_matrix_setcell3d(), jit_matrix_setplane1d(), jit_matrix_setplane2d(), and jit_matrix_setplane3d().

36.68.2.8 void* jit_atom_getobj (t_atom * a)

Retrieves atom value as object pointer.

Parameters

a	atom pointer
---	--------------

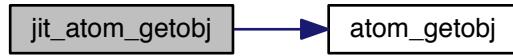
Returns

object pointer. NULL if atom has no object value.

References atom_getobj().

Referenced by jit_matrix_setcell().

Here is the call graph for this function:



36.68.2.9 `t_symbol * jit_atom_getsym (t_atom * a)`

Retrieves atom value as symbol pointer.

Parameters

<code>a</code>	atom pointer
----------------	--------------

Returns

symbol pointer. `_jit_sym_nothing` if atom has no symbolic value.

Referenced by `jit_matrix_exprfill()`, `jit_matrix_jit_gl_texture()`, `jit_matrix_op()`, `jit_matrix_setcell()`, and `max_jit_mop->jit_matrix()`.

36.68.2.10 `t_jit_err jit_atom_setfloat (t_atom * a, double b)`

Sets atom value to floating point number.

Parameters

<code>a</code>	atom pointer
<code>b</code>	floating point value

Returns

`t_jit_err` error code.

References `atom_setfloat()`.

Referenced by `jit_matrix_getcell()`.

Here is the call graph for this function:



36.68.2.11 t jit_err jit_atom_setlong (t_atom * a, t_atom_long b)

Sets atom value to long integer.

Parameters

<i>a</i>	atom pointer
<i>b</i>	integer value

Returns

`t_jit_err` error code.

Referenced by `jit_matrix_getcell()`, `jit_matrix_setcell1d()`, `jit_matrix_setcell2d()`, `jit_matrix_setcell3d()`, `jit_matrix_setplane1d()`, `jit_matrix_setplane2d()`, `jit_matrix_setplane3d()`, and `jit_ob3d_draw_chunk()`.

36.68.2.12 `t_jit_err jit_atom_setobj(t_atom * a, void * b)`

Sets atom value to object pointer.

Parameters

<i>a</i>	atom pointer
<i>b</i>	object pointer

Returns

`t_jit_err` error code.

References `atom_setobj()`.

Referenced by `jit_gl_begincapture()`, `jit_matrix_getcell()`, and `jit_matrix_jit_gl_texture()`.

Here is the call graph for this function:

**36.68.2.13 `t_jit_err jit_atom_setsym(t_atom * a, t_symbol * b)`**

Sets atom value to symbol.

Parameters

<i>a</i>	atom pointer
<i>b</i>	symbol value

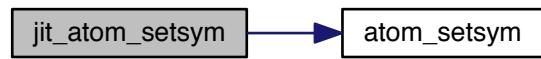
Returns

`t_jit_err` error code.

References `atom_setsym()`.

Referenced by `jit_matrix_getcell()`, `jit_matrix_setcell1d()`, `jit_matrix_setcell2d()`, `jit_matrix_setcell3d()`, `jit_matrix_setplane1d()`, `jit_matrix_setplane2d()`, `jit_matrix_setplane3d()`, `jit_mop_single_type()`, `jit_ob3d_new()`, `jit_object_exportattrs()`, and `max_jit_mop_outputmatrix()`.

Here is the call graph for this function:



36.69 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.
- `t_atom_long jit_attr_canget (t_jit_attr **x)`
Retrieves attribute gettable flag.
- `t_atom_long jit_attr_canset (t_jit_attr **x)`
Retrieves attribute settable flag.
- `t_atom_long jit_attr_usercanget (t_jit_attr **x)`
Retrieves attribute user gettable flag.
- `t_atom_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`.
- `t_atom_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, t_atom_long c)`
Sets attribute value as a long integer value.
- `t_atom_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, t_atom_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, t_atom_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, t_atom_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.

36.69.1 Detailed Description

36.69.2 Function Documentation

36.69.2.1 `t_atom_long jit_attr_caget(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.

References [t_jit_attr::flags](#), and [JIT_ATTR_GET_OPAQUE](#).

36.69.2.2 t_atom_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 object.

References [t_jit_attr::flags](#), and [JIT_ATTR_SET_OPAQUE](#).

36.69.2.3 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.

References [_jit_sym_float64](#), [t_jit_attr_filter_clip::max](#), [t_jit_attr_filter_clip::min](#), [t_jit_attr_filter_clip::scale](#), [t_jit_attr_filter_clip::type](#), [t_jit_attr_filter_clip::usemax](#), [t_jit_attr_filter_clip::usemin](#), and [t_jit_attr_filter_clip::usescale](#).

36.69.2.4 t_jit_object * jit_attr_filter_proc_new(method proc)

Constructs instance of [t_jit_attr_filter_proc](#).

Parameters

<i>proc</i>	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`.

References `t_jit_attr_filter_proc::proc`.

36.69.2.5 `t_jit_err jit_attr_filterget(t_jit_attr *x, void *y)`

Sets attribute getter filter.

Parameters

<i>x</i>	attribute object pointer
<i>y</i>	getter filter object

Returns

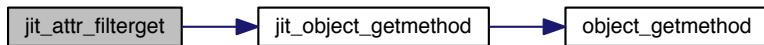
`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of any attribute object.

References `_jit_sym_filter`, `t_jit_attr::filterget`, and `jit_object_getmethod()`.

Here is the call graph for this function:



36.69.2.6 `t_jit_err jit_attr_filterset(t_jit_attr *x, void *y)`

Sets attribute setter filter.

Parameters

<i>x</i>	attribute object pointer
<i>y</i>	setter filter object

Returns

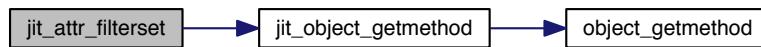
`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of any attribute object.

References `_jit_sym_filter`, `t_jit_attr::filterset`, and `jit_object_getmethod()`.

Here is the call graph for this function:



36.69.2.7 `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

<i>x</i>	attribute object pointer
<i>parent</i>	target object pointer
<i>ac</i>	pointer to argument count
<i>av</i>	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 object.

References `t_jit_attr::filterget`, and `t_jit_attr::get`.

36.69.2.8 `long jit_attr_getchar_array(void *x, t_symbol *s, long max, uchar *vals)`

Retrieves attribute value as an array of char values.

Parameters

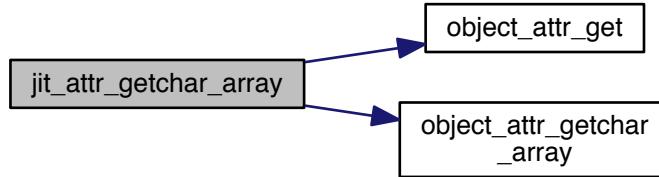
<i>x</i>	object pointer
<i>s</i>	attribute name
<i>max</i>	maximum number of values to copy
<i>vals</i>	pointer to retrieved values

Returns

number of values retrieved.

References `object_attr_get()`, and `object_attr_getchar_array()`.

Here is the call graph for this function:



36.69.2.9 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

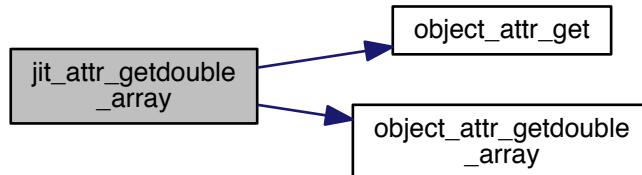
<i>x</i>	object pointer
<i>s</i>	attribute name
<i>max</i>	maximum number of values to copy
<i>vals</i>	pointer to retrieved values

Returns

number of values retrieved.

References object_attr_get(), and object_attr_getdouble_array().

Here is the call graph for this function:



36.69.2.10 t_atom_float jit_attr_getfloat(void *x, t_symbol *s)

Retrieves attribute value as a floating point value.

Parameters

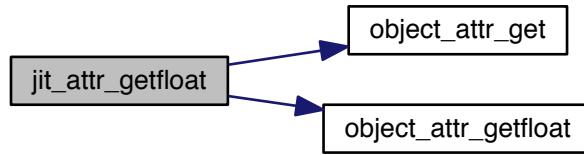
<i>x</i>	object pointer
<i>s</i>	attribute name

Returns

floating point value

References object_attr_get(), and object_attr_getfloat().

Here is the call graph for this function:



36.69.2.11 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

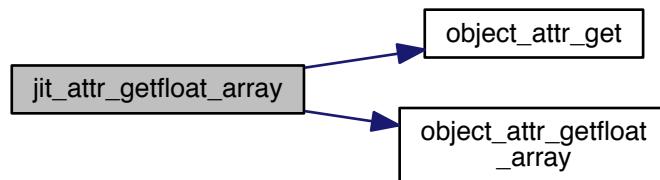
<i>x</i>	object pointer
<i>s</i>	attribute name
<i>max</i>	maximum number of values to copy
<i>vals</i>	pointer to retrieved values

Returns

number of values retrieved.

References object_attr_get(), and object_attr_getfloat_array().

Here is the call graph for this function:



36.69.2.12 `t_atom_long jit_attr_getlong(void *x, t_symbol *s)`

Retrieves attribute value as a long integer value.

Parameters

<i>x</i>	object pointer
<i>s</i>	attribute name

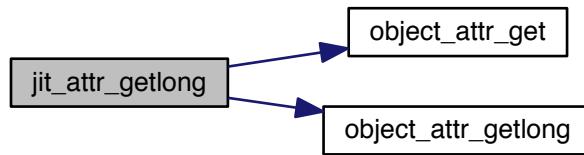
Returns

long integer value

References object_attr_get(), and object_attr_getlong().

Referenced by max jit_classex_mop_wrap(), max jit_mop_adapt_matrix_all(), max jit_mop_clear(), max jit_mop_free(), max jit_mop_getoutputmode(), max jit_mop_inputs(), max jit_mop_jit_matrix(), max jit_mop_matrix_args(), max jit_mop_notify(), max jit_mop_outputmatrix(), and max jit_mop_outputs().

Here is the call graph for this function:



36.69.2.13 long jit_attr_getlong_array (void * *x*, t_symbol * *s*, long *max*, t_atom_long * *vals*)

Retrieves attribute value as an array of long integer values.

Parameters

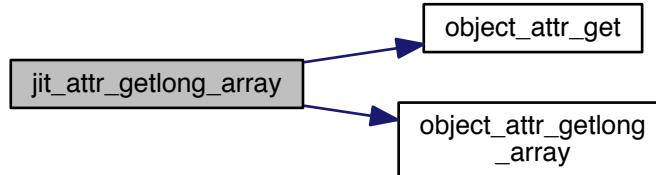
<i>x</i>	object pointer
<i>s</i>	attribute name
<i>max</i>	maximum number of values to copy
<i>vals</i>	pointer to retrieved values

Returns

number of values retrieved.

References `object_attr_get()`, and `object_attr_getlong_array()`.

Here is the call graph for this function:

**36.69.2.14 method jit_attr_getmethod (*t_jit_attr *x, t_symbol *methodname*)**

Retrieves attribute getter or setter method.

Parameters

<i>x</i>	attribute object pointer
<i>methodname</i>	"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.

References `_jit_sym_get`, `_jit_sym_set`, `t_jit_attr::filterget`, `t_jit_attr::filterset`, `t_jit_attr::get`, and `t_jit_attr::set`.

36.69.2.15 *t_symbol * jit_attr_getname (t_jit_attr *x)*

Retrieves attribute name.

Parameters

<i>x</i>	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.

References `_jit_sym_nothing`, and `t_jit_attr::name`.

36.69.2.16 `t_symbol* jit_attr_getsym(void *x, t_symbol *s)`

Retrieves attribute value as a symbol value.

Parameters

<i>x</i>	object pointer
<i>s</i>	attribute name

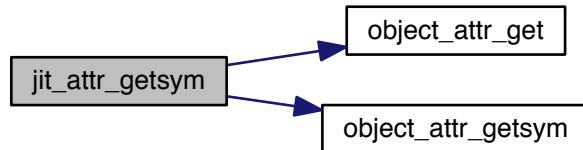
Returns

symbol value

References object_attr_get(), and object_attr_getsym().

Referenced by jit_mop_io_restrict_type(), jit_ob3d_free(), jit_ob3d_new(), max_jit_classex_mop_wrap(), max_jit_mop_assist(), max_jit_mop_free(), max_jit_mop_notify(), max_jit_mop_outputmatrix(), max_jit_ob3d_attach(), and max_jit_ob3d_detach().

Here is the call graph for this function:



36.69.2.17 `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

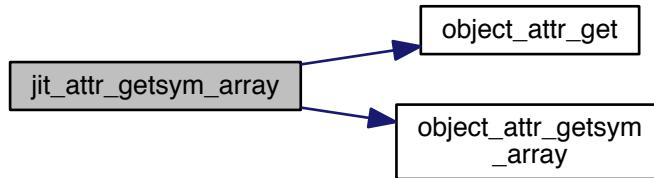
<i>x</i>	object pointer
<i>s</i>	attribute name
<i>max</i>	maximum number of values to copy
<i>vals</i>	pointer to retrieved values

Returns

number of values retrieved.

References `object_attr_get()`, and `object_attr_getsym_array()`.

Here is the call graph for this function:

**36.69.2.18 `t_symbol * jit_attr_gettype (t_jit_attr * x)`**

Retrieves attribute type.

Parameters

<code>x</code>	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.

References `_jit_sym_nothing`, and `t_jit_attr::type`.

36.69.2.19 `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

<code>name</code>	attribute name
<code>type</code>	data type
<code>size</code>	maximum size
<code>flags</code>	privacy flags
<code>mget</code>	getter method

<i>mset</i>	setter method
<i>offsetcount</i>	byte offset to count struct member (if zero, remain fixed size with max size)
<i>offset</i>	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`.

References `t_jit_attr_offset_array::filterget`, `t_jit_attr_offset_array::filterset`, `t_jit_attr_offset_array::flags`, `gensym()`, `t_jit_attr_offset_array::get`, `t_jit_attr_offset_array::name`, `t_jit_attr_offset_array::offset`, `t_jit_attr_offset_array::offsetcount`, `t_jit_attr_offset_array::reserved`, `t_jit_attr_offset_array::set`, `t_jit_attr_offset_array::size`, and `t_jit_attr_offset_array::type`.

Here is the call graph for this function:



36.69.2.20 `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

<i>name</i>	attribute name
<i>type</i>	data type
<i>flags</i>	privacy flags
<i>mget</i>	getter method
<i>mset</i>	setter method
<i>offset</i>	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`.

References `t_jit_attr_offset::filterget`, `t_jit_attr_offset::filterset`, `t_jit_attr_offset::flags`, `gensym()`, `t_jit_attr_offset::get`, `t_jit_attr_offset::name`, `t_jit_attr_offset::offset`, `t_jit_attr_offset::reserved`, `t_jit_attr_offset::set`, and `t_jit_attr_offset::type`.

Here is the call graph for this function:



36.69.2.21 `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

<i>x</i>	attribute object pointer
<i>parent</i>	target object pointer
<i>ac</i>	argument count
<i>av</i>	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 object.

References `t_jit_attr::filterset`, and `t_jit_attr::set`.

36.69.2.22 `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

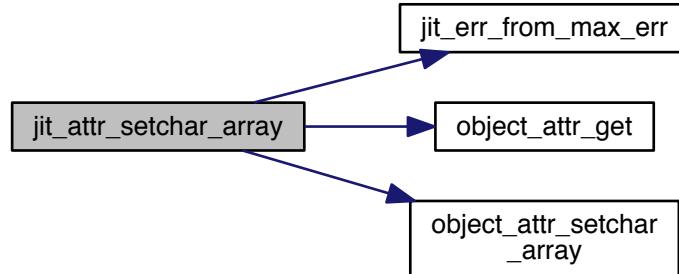
<i>x</i>	object pointer
<i>s</i>	attribute name
<i>count</i>	number of values
<i>vals</i>	pointer to values

Returns

`t_jit_err` error code.

References `jit_err_from_max_err()`, `object_attr_get()`, and `object_attr_setchar_array()`.

Here is the call graph for this function:

**36.69.2.23 `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

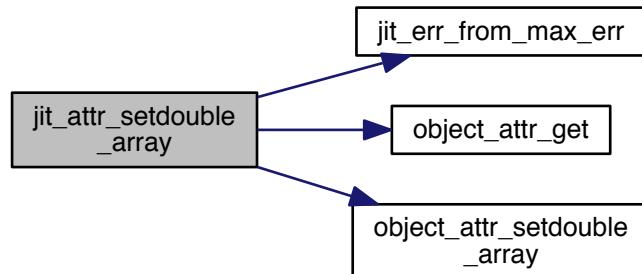
<code>x</code>	object pointer
<code>s</code>	attribute name
<code>count</code>	number of values
<code>vals</code>	pointer to values

Returns

`t_jit_err` error code.

References `jit_err_from_max_err()`, `object_attr_get()`, and `object_attr_setdouble_array()`.

Here is the call graph for this function:



36.69.2.24 `t_jit_err jit_attr_setfloat(void *x, t_symbol *s, t_atom_float c)`

Sets attribute value as a floating point value.

Parameters

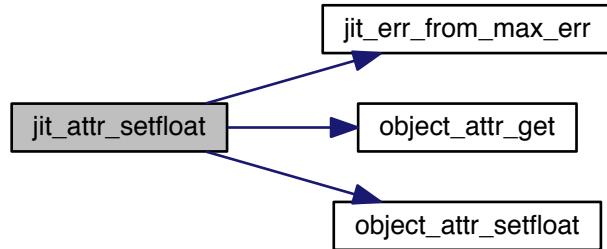
<i>x</i>	object pointer
<i>s</i>	attribute name
<i>c</i>	value

Returns

`t_jit_err` error code.

References `jit_err_from_max_err()`, `object_attr_get()`, and `object_attr_setfloat()`.

Here is the call graph for this function:



36.69.2.25 `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

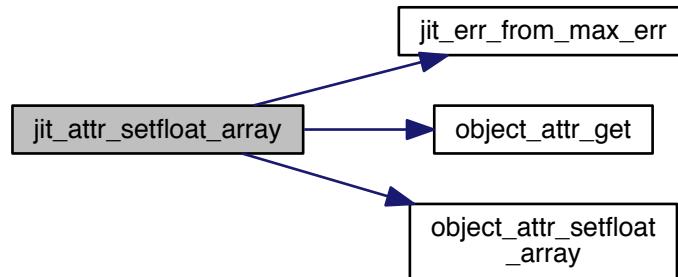
<i>x</i>	object pointer
<i>s</i>	attribute name
<i>count</i>	number of values
<i>vals</i>	pointer to values

Returns

`t_jit_err` error code.

References `jit_err_from_max_err()`, `object_attr_get()`, and `object_attr_setfloat_array()`.

Here is the call graph for this function:

**36.69.2.26 `t_jit_err jit_attr_setlong(void *x, t_symbol *s, t_atom_long c)`**

Sets attribute value as a long integer value.

Parameters

<code>x</code>	object pointer
<code>s</code>	attribute name
<code>c</code>	value

Returns

`t_jit_err` error code.

Referenced by `jit_mop_input_nolink()`, `jit_mop_output_nolink()`, `jit_mop_single_planecount()`, and `max_jit_mop←matrix_args()`.

36.69.2.27 `t_jit_err jit_attr_setlong_array(void *x, t_symbol *s, long count, t_atom_long *vals)`

Sets attribute value as an array of long integer values.

Parameters

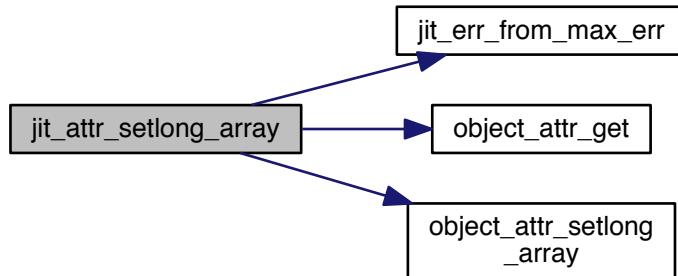
<code>x</code>	object pointer
<code>s</code>	attribute name
<code>count</code>	number of values
<code>vals</code>	pointer to values

Returns

`t_jit_err` error code.

References `jit_err_from_max_err()`, `object_attr_get()`, and `object_attr_setlong_array()`.

Here is the call graph for this function:



36.69.2.28 `t_jit_err jit_attr_setsym(void *x, t_symbol *s, t_symbol *c)`

Sets attribute value as a symbol value.

Parameters

<code>x</code>	object pointer
<code>s</code>	attribute name
<code>c</code>	value

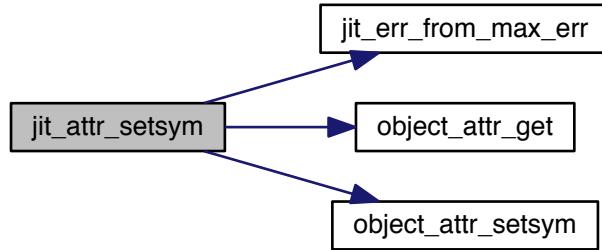
Returns

`t_jit_err` error code.

References `jit_err_from_max_err()`, `object_attr_get()`, and `object_attr_setsym()`.

Referenced by `jit_matrix_exprfill()`, `jit_matrix_op()`, `jit_mop_new()`, `jit_ob3d_free()`, `jit_ob3d_new()`, `max_jit_mop_free()`, `max_jit_mop_inputs()`, `max_jit_mop_jit_matrix()`, `max_jit_mop_notify()`, and `max_jit_mop_outputs()`.

Here is the call graph for this function:



36.69.2.29 `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

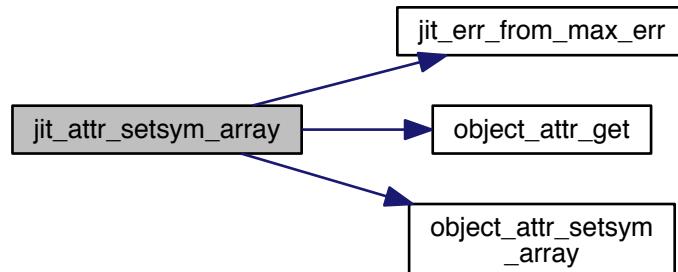
<code>x</code>	object pointer
<code>s</code>	attribute name
<code>count</code>	number of values
<code>vals</code>	pointer to values

Returns

`t_jit_err` error code.

References `jit_err_from_max_err()`, `object_attr_get()`, and `object_attr_setsym_array()`.

Here is the call graph for this function:



36.69.2.30 long jit_attr_symcompare (void * *x*, t_symbol * *name*)

Compares symbol name with name provided.

Parameters

<i>x</i>	attribute object pointer
<i>name</i>	attribute name

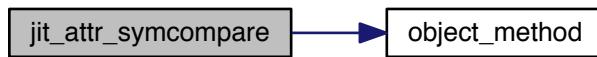
Returns

1 if equal, 0 if not equal

References `_jit_sym_getname`, and `object_method()`.

Referenced by `max_jit_obex_attr_get()`, and `max_jit_obex_attr_set()`.

Here is the call graph for this function:

**36.69.2.31 t_atom_long jit_attr_usercanget (t_jit_attr *x)**

Retrieves attribute user gettable flag.

Parameters

<i>x</i>	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.

References `t_jit_attr::flags`, and `JIT_ATTR_GET_OPAQUE_USER`.

36.69.2.32 t_atom_long jit_attr_usercanset (t_jit_attr *x)

Retrieves attribute user settable flag.

Parameters

<i>x</i>	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.

References t_jit_attr::flags, and JIT_ATTR_SET_OPAQUE_USER.

36.69.2.33 t_jit_object *jit_attribute_new (char *name, t_symbol *type, long flags, method mget, method mset)

Constructs instance of [t_jit_attribute](#).

Parameters

<i>name</i>	attribute name
<i>type</i>	data type
<i>flags</i>	privacy flags
<i>mget</i>	getter method
<i>mset</i>	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.

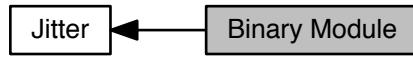
References t_jit_attribute::data, t_jit_attribute::filterget, t_jit_attribute::filterset, t_jit_attribute::flags, gensym(), t_jit_attribute::get, t_jit_attribute::name, t_jit_attribute::reserved, t_jit_attribute::set, t_jit_attribute::size, and t_jit_attribute::type.

Here is the call graph for this function:



36.70 Binary Module

Collaboration diagram for Binary Module:



Functions

- `t_jit_err jit_bin_read_header (t_filehandle fh, t_uint32 *version, t_int32 *filesize)`
Reads the header of a JXF binary file.
- `t_jit_err jit_bin_read_chunk_info (t_filehandle fh, t_uint32 *ckid, t_int32 *ckszie)`
Reads the info of a chunk from a JXF binary file.
- `t_jit_err jit_bin_write_header (t_filehandle fh, t_int32 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.

36.70.1 Detailed Description

36.70.2 Function Documentation

36.70.2.1 `t_jit_err jit_bin_read_chunk_info (t_filehandle fh, t_uint32 * ckid, t_int32 * ckszie)`

Reads the info of a chunk from a JXF binary file.

Parameters

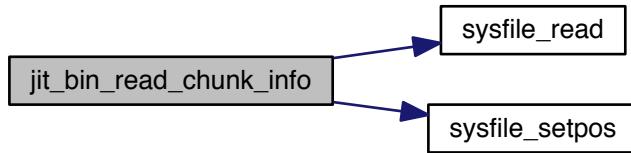
<code>fh</code>	<code>t_filehandle</code> file handle
<code>ckid</code>	chunk ID (ie JIT_BIN_CHUNK_CONTAINER, JIT_BIN_CHUNK_MATRIX)
<code>ckszie</code>	the size of the chunk

Returns

`t_jit_err` error code.

References `SYSFILE_FROMMARK`, `sysfile_read()`, and `sysfile_setpos()`.

Here is the call graph for this function:



36.70.2.2 `t_jit_err jit_bin_read_header (t_filehandle fh, t_uint32 * version, t_int32 * filesize)`

Reads the header of a JXF binary file.

Parameters

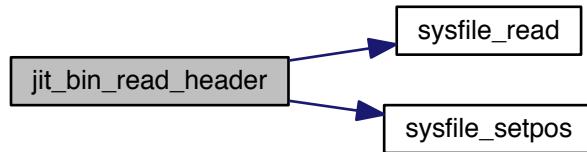
<code><i>fh</i></code>	<code>t_filehandle</code> file handle
<code><i>version</i></code>	version of the binary file format (ie <code>JIT_BIN_VERSION_1</code>)
<code><i>filesize</i></code>	the size of the file

Returns

`t_jit_err` error code.

References `SYSFILE_FROMSTART`, `sysfile_read()`, and `sysfile_setpos()`.

Here is the call graph for this function:



36.70.2.3 `t_jit_err jit_bin_read_matrix (t_filehandle fh, void * matrix)`

Reads matrix data from a JXF binary file.

Parameters

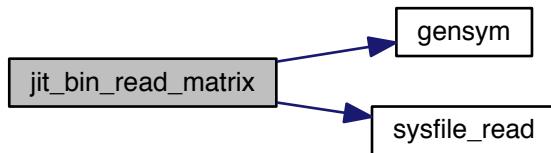
<i>fh</i>	t_filehandle file handle
<i>matrix</i>	the matrix data

Returns

t_jit_err error code.

References _jit_sym_char, _jit_sym_float32, _jit_sym_float64, _jit_sym_getdata, _jit_sym_getinfo, _jit_sym_lock, _jit_sym_long, _jit_sym_setinfo, t_jit_matrix_info::dim, t_jit_matrix_info::dimcount, t_jit_matrix_info::flags, gensym(), JIT_MATRIX_MAX_DIMCOUNT, t_jit_matrix_info::planecount, sysfile_read(), and t_jit_matrix_info::type.

Here is the call graph for this function:



36.70.2.4 t_jit_err jit_bin_write_header (t_filehandle *fh*, t_int32 *filesize*)

Writes the header of a JXF binary file.

Parameters

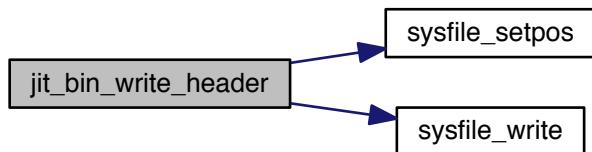
<i>fh</i>	t_filehandle file handle
<i>filesize</i>	the size of the file

Returns

t_jit_err error code.

References SYSFILE_FROMSTART, sysfile_setpos(), and sysfile_write().

Here is the call graph for this function:



36.70.2.5 t_jit_err jit_bin_write_matrix(t_filehandle *fh*, void * *matrix*)

Writes a matrix to a JXF binary file.

Parameters

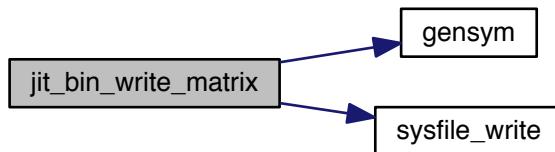
<i>fh</i>	t_filehandle file handle
<i>matrix</i>	the matrix data

Returns

t_jit_err error code.

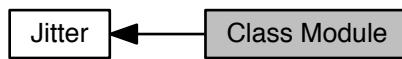
References _jit_sym_char, _jit_sym_float32, _jit_sym_float64, _jit_sym_long, t_jit_matrix_info::dim, t_jit_matrix_info::dimcount, gensym(), JIT_MATRIX_MAX_DIMCOUNT, t_jit_matrix_info::planecount, sysfile_write(), and t_jit_matrix_info::type.

Here is the call graph for this function:



36.71 Class Module

Collaboration diagram for Class Module:



Functions

- `t_max_err class_copy (t_symbol *src_name_space, t_symbol *src_classname, t_symbol *dst_name_space, t_symbol *dst_classname)`

Duplicates a previously registered object class, and registers a copy of this class.
- `void * jit_class_new (C74_CONST 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, const 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_addadornment (void *c, t_jit_object *o)`

Adds an adornment to a class.
- `t_jit_err jit_class_addinterface (void *c, void *interfaceclass, long byteoffset, long flags)`

Adds an interface 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.

36.71.1 Detailed Description

36.71.2 Function Documentation

36.71.2.1 `t_max_err class_copy (t_symbol * src_name_space, t_symbol * src_classname, t_symbol * dst_name_space, t_symbol * dst_classname)`

Duplicates a previously registered object class, and registers a copy of this class.

Parameters

<code>src_name_space</code>	The source class's name space.
<code>src_classname</code>	The source class's class name.
<code>dst_name_space</code>	The copied class's name space.
<code>dst_classname</code>	The copied class's class name.

Returns

This function returns the error code `MAX_ERR_NONE` if successful, or one of the other error codes defined in "ext_obex.h" if unsuccessful.

36.71.2.2 `t_jit_err jit_class_addadornment (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

<code>c</code>	class pointer
<code>o</code>	object to use as adornment

Returns

`t_jit_err` error code

References `jit_err_from_max_err()`.

Here is the call graph for this function:



36.71.2.3 t_jit_err jit_class_addattr(void * *c*, t_jit_object * *attr*)

Adds an attribute to a class.

Parameters

<i>c</i>	class pointer
<i>attr</i>	attribute object

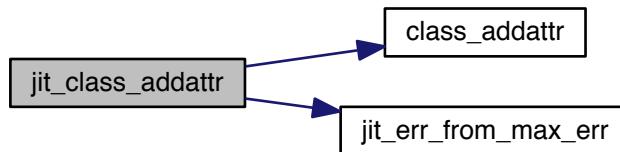
Returns

`t_jit_err` error code

References `class_addattr()`, and `jit_err_from_max_err()`.

Referenced by `jit_ob3d_setup()`.

Here is the call graph for this function:



36.71.2.4 `t_jit_err jit_class_addinterface(void * c, void * interfaceclass, long byteoffset, long flags)`

Adds an interface to a class.

Automatically expose methods and attributes of an interface class to a classes. Can also be used for class containers or subclassing behavior. If method or attribute is present in interface class prior to this call, the interface class' method or attribute will not be added. Use a nonzero byteoffset to contained class' object pointer in struct for container class. Use byte offset of zero for interface or subclassing behavior.

Parameters

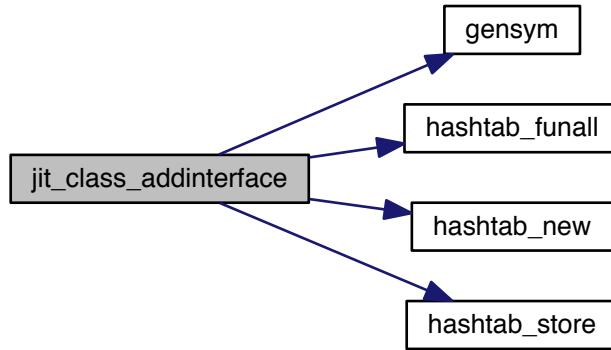
<i>c</i>	class pointer
<i>interfaceclass</i>	interface class pointer
<i>byteoffset</i>	byte offset (if for a contained object)
<i>flags</i>	reserved for future use

Returns

`t_jit_err` error code

References `gensym()`, `hashtab_funall()`, `hashtab_new()`, and `hashtab_store()`.

Here is the call graph for this function:



36.71.2.5 `t_jit_err jit_class_addmethod(void *c, method m, const char *name, ...)`

Adds a named method to a class.

Parameters

<code>c</code>	class pointer
<code>m</code>	function called when method is invoked
<code>name</code>	method name
...	type signature for the method in the standard Max type list format (see Chapter 3 of the Writing Externals in Max document for more information)

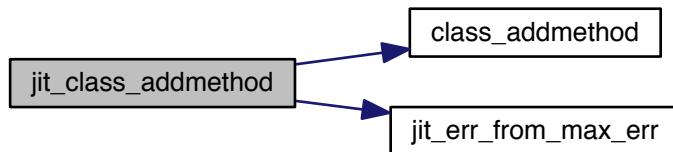
Returns

`t_jit_err` error code

References `class_addmethod()`, and `jit_err_from_max_err()`.

Referenced by `jit_ob3d_setup()`.

Here is the call graph for this function:



36.71.2.6 `t_jit_err jit_class_addtypedwrapper(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

<code>c</code>	class pointer
<code>m</code>	function called when method is invoked
<code>name</code>	method name
<code>...</code>	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

References `jit_err_from_max_err()`.

Here is the call graph for this function:



36.71.2.7 `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

<i>c</i>	class pointer
<i>classname</i>	classname of adornment to retrieve

Returns

`t_jit_err` error code

Referenced by `max_jit_classex_mop_mproc()`, `max_jit_classex_mop_wrap()`, and `max_jit_mop_setup()`.

36.71.2.8 `void* jit_class_attr_get (void * c, t_symbol * attrname)`

Retrieves attribute pointer associated with name provided.

Parameters

<i>c</i>	class pointer
<i>attrname</i>	attribute name

Returns

attribute object pointer

36.71.2.9 `void* jit_class_findbyname (t_symbol * classname)`

Retrieves class pointer associated with name provided.

Parameters

<i>classname</i>	class name
------------------	------------

Returns

class pointer

References `class_findbyname()`.

Here is the call graph for this function:



36.71.2.10 `t_jit_err jit_class_free (void * c)`

Frees a class.

Warning

This function is not typically used outside of jitlib.

Parameters

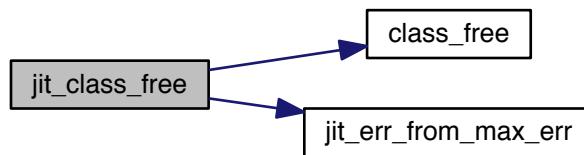
<i>c</i>	class pointer
----------	---------------

Returns

`t_jit_err` error code

References `class_free()`, and `jit_err_from_max_err()`.

Here is the call graph for this function:

**36.71.2.11 `t_messlist* jit_class_mess (t_jit_class * c, t_symbol * methodname)`**

Retrieves messlist entry for named method.

Parameters

<i>c</i>	class pointer
<i>methodname</i>	method name

Returns

`t_messlist` pointer.

36.71.2.12 `method jit_class_method (void * c, t_symbol * methodname)`

Retrieves method function pointer for named method.

Parameters

<i>c</i>	class pointer
<i>methodname</i>	method name

Returns

method function pointer.

36.71.2.13 `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

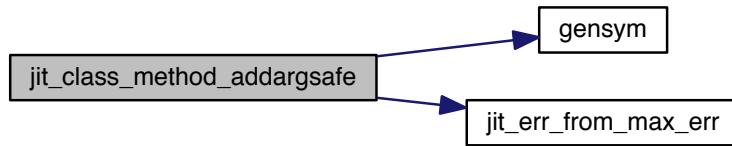
<code>c</code>	class pointer
<code>argname</code>	name as used via argument
<code>methodname</code>	name of method to map the argument name to

Returns

`t_jit_err` error code

References `gensym()`, and `jit_err_from_max_err()`.

Here is the call graph for this function:



36.71.2.14 `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

<code>c</code>	class pointer
<code>s</code>	name as used via argument

Returns

If successful, name of method to map the argument name to. Otherwise, NULL.

References gensym(), and `t_symbol::s_name`.

Referenced by `jit_object_method_argsafe_get()`.

Here is the call graph for this function:

**36.71.2.15 `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

References `class_nameget()`.

Referenced by `jit_class_symcompare()`.

Here is the call graph for this function:

**36.71.2.16 `void* jit_class_new (C74_CONST char * name, method mnew, method mfree, long size, ...)`**

Creates a new class with the name specified by the name argument.

Parameters

<i>name</i>	class name
<i>mnew</i>	class constructor
<i>mfree</i>	class destructor
<i>size</i>	object struct size in bytes
...	type signature for the constructor in the standard Max type list format (see Chapter 3 of the Writing Externals in Max document for more information)

Warning

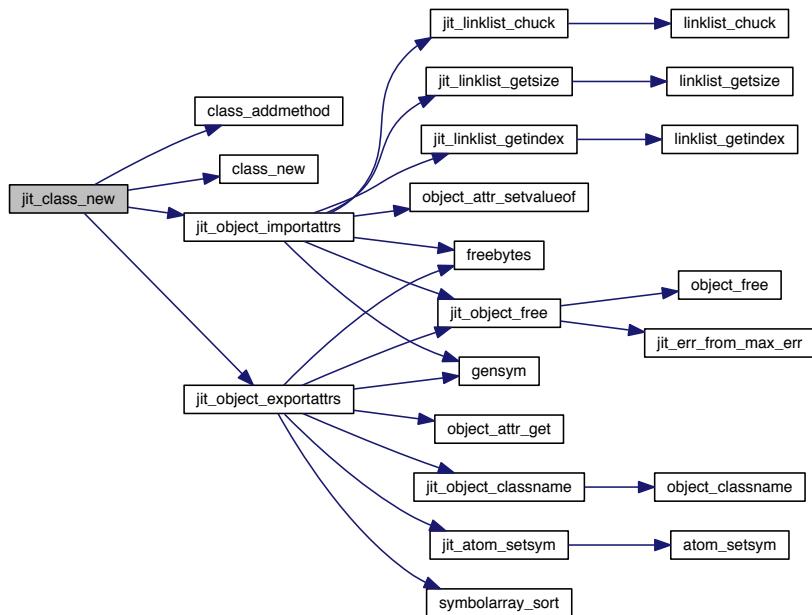
In order for the Jitter class to be exposed to JavaScript and Java, it is important that the constructor is typed, even if no arguments are provided—i.e. do not use the older strategy of defining Jitter constructors as private and untyped with A_CANT.

Returns

class pointer to be used in other class functions

References A_CANT, A_GIMME, class_addmethod(), class_new(), jit_object_exportattrs(), and jit_object_importattrs().

Here is the call graph for this function:

**36.71.2.17 t_jit_err jit_class_register(void * c)**

Registers class in the class registry.

Parameters

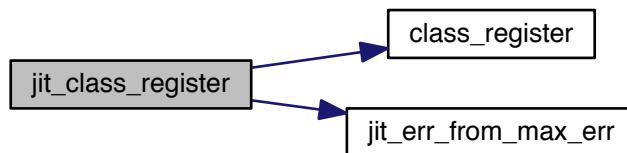
<i>c</i>	class pointer
----------	---------------

Returns

`t_jit_err` error code

References `class_register()`, and `jit_err_from_max_err()`.

Here is the call graph for this function:

**36.71.2.18 long jit_class_symcompare (void * *c*, t_symbol * *name*)**

Compares name of class with the name provided.

Parameters

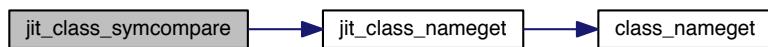
<i>c</i>	class pointer
<i>name</i>	name to compare with class name

Returns

1 if equal, 0 if not equal

References `jit_class_nameget()`.

Here is the call graph for this function:

**36.71.2.19 t_messlist* jit_class_typedwrapper_get (void * *c*, t_symbol * *s*)**

Retrieves typed wrapper messlist pointer associated with name provided.

Parameters

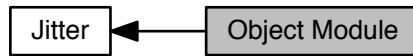
<i>c</i>	class pointer
<i>s</i>	name

Returns

[t_messlist](#) pointer

36.72 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_imp` (void *cn, void *p1, void *p2, void *p3, void *p4, void *p5, void *p6, void *p7, void *p8, void *dummy)

Instantiates an object specified by class name.
- void * `jit_object_method_imp` (void *x, void *s, void *p1, void *p2, void *p3, void *p4, void *p5, void *p6, void *p7, void *p8)

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.

36.72.1 Detailed Description

36.72.2 Function Documentation

36.72.2.1 void* jit_object_attach (t_symbol * s, void * x)

Attaches an object as a client of a named server object for notification.

Parameters

<code>s</code>	name of server object
<code>x</code>	client object pointer

Returns

If successful, server object pointer. Otherwise NULL.

References `object_attach()`.

Referenced by `max_jit_mop_inputs()`, `max_jit_mop_notify()`, `max_jit_mop_outputs()`, and `max_jit_ob3d_attach()`.

Here is the call graph for this function:



36.72.2.2 void* jit_object_attr_get (void * x, t_symbol * attrname)

Retrieves an object's attribute pointer specified by attribute name.

Parameters

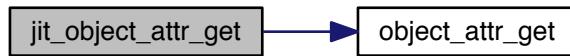
<code>x</code>	object pointer
<code>attrname</code>	attribute name

Returns

attribute object pointer

References `object_attr_get()`.

Here is the call graph for this function:

**36.72.2.3 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

References `object_attr_usercanget()`.

Here is the call graph for this function:

**36.72.2.4 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

References `object_attr_usercanset()`.

Referenced by `max_jit_attr_args()`.

Here is the call graph for this function:

**36.72.2.5 void* jit_object_class (void * x)**

Retrieves an object's class pointer.

Parameters

x	object pointer
---	----------------

Returns

class pointer

References `object_class()`.

Referenced by `jit_object_method_argsafe_get()`, and `max_jit_mop_setup()`.

Here is the call graph for this function:

**36.72.2.6 t_symbol* jit_object_classname (void * x)**

Retrieves an object's class name.

Parameters

x	object pointer
---	----------------

Returns

class name **t_symbol** pointer

References `object_classname()`.

Referenced by `jit_gl_begincapture()`, `jit_ob3d_free()`, `jit_ob3d_new()`, `jit_object_classname_compare()`, and `jit_object_exportattrs()`.

Here is the call graph for this function:



36.72.2.7 long jit_object_classname_compare (void * x, t_symbol * name)

Compares object's class name with the name provided.

Parameters

<i>x</i>	object pointer
<i>name</i>	name to compare with class name

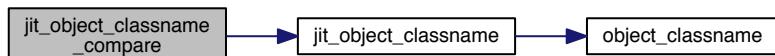
Returns

1 if equal, 0 if not equal

References `jit_object_classname()`.

Referenced by `max_jit_obex_adornment_get()`.

Here is the call graph for this function:



36.72.2.8 t_jit_err jit_object_detach (t_symbol * s, void * x)

Detaches a client object from a named server object.

Parameters

<i>s</i>	name of server object
<i>x</i>	client object pointer

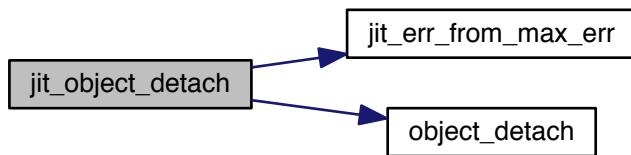
Returns

`t_jit_err` error code

References `jit_err_from_max_err()`, and `object_detach()`.

Referenced by `max_jit_mop_free()`, and `max_jit_ob3d_detach()`.

Here is the call graph for this function:



36.72.2.9 `t_jit_err jit_object_exportattrs(void *x, t_symbol *s, long argc, t_atom *argv)`

Exports object attributes to an XML file.

Parameters

<code>x</code>	object pointer
<code>s</code>	ignored
<code>argc</code>	argument count
<code>argv</code>	argument vector

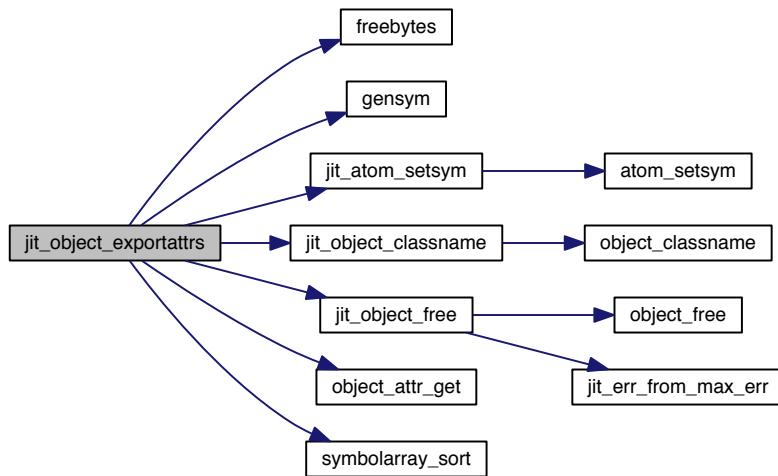
Returns

`t_jit_err` error code

References `_jit_sym_getname`, `freebytes()`, `gensym()`, `jit_atom_setsym()`, `jit_object_classname()`, `jit_object_free()`, `object_attr_get()`, `t_symbol::s_name`, and `symbolarray_sort()`.

Referenced by `jit_class_new()`, and `max_jit_classex_standard_wrap()`.

Here is the call graph for this function:



36.72.2.10 `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

<i>x</i>	object pointer
<i>s</i>	ignored
<i>argc</i>	argument count
<i>argv</i>	argument vector

Returns

`t_jit_err` error code

Referenced by `max_jit_classex_standard_wrap()`.

36.72.2.11 `void * jit_object_findregistered (t_symbol * s)`

Retrieves a registered object associated with name.

Parameters

<i>s</i>	registered name
----------	-----------------

Returns

If successful, object pointer. Otherwise NULL.

Referenced by `jit_gl_begincapture()`, `jit_matrix_jit_gl_texture()`, `jit_matrix_op()`, `max_jit_mop_jit_matrix()`, and `max_jit_mop_notify()`.

36.72.2.12 `t_symbol* jit_object_findregisteredbyptr (void * x)`

Retrieves a registered object's name.

Parameters

<code>x</code>	object pointer
----------------	----------------

Returns

If successful, `t_symbol` pointer name. Otherwise NULL.

References `object_findregisteredbyptr()`.

Here is the call graph for this function:



36.72.2.13 `t_jit_err jit_object_free (void * x)`

Frees an object.

Parameters

<code>x</code>	object pointer
----------------	----------------

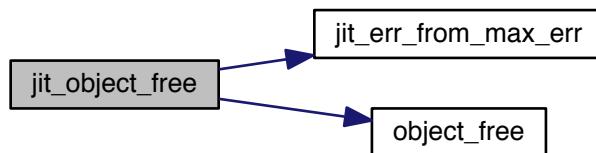
Returns

`t_jit_err` error code

References `jit_err_from_max_err()`, and `object_free()`.

Referenced by `jit_glchunk_delete()`, `jit_matrix_exprfill()`, `jit_matrix_new()`, `jit_matrix_newcopy()`, `jit_matrix_op()`, `jit_mop_free()`, `jit_mop_new()`, `jit_mop_newcopy()`, `jit_ob3d_free()`, `jit_object_exportattrs()`, `jit_object_importattrs()`, `max_jit_mop_adapt_matrix_all()`, `max_jit_mop_free()`, and `max_jit_obex_free()`.

Here is the call graph for this function:



36.72.2.14 method jit_object_getmethod (void * x, t_symbol * s)

Retrieves an object method specified by method name.

Parameters

<i>x</i>	object pointer
<i>s</i>	method name

Returns

method

References object_getmethod().

Referenced by jit_attr_filterget(), jit_attr_filterset(), and jit_ob3d_new().

Here is the call graph for this function:

**36.72.2.15 t_jit_err jit_object_importattrs (void * *x*, t_symbol * *s*, long *argc*, t_atom * *argv*)**

Imports object attributes from an XML file.

Parameters

<i>x</i>	object pointer
<i>s</i>	ignored
<i>argc</i>	argument count
<i>argv</i>	argument vector

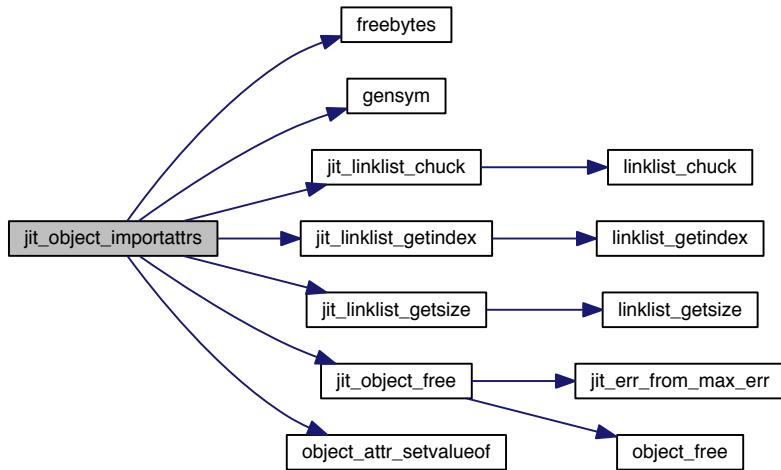
Returns

`t_jit_err` error code

References `_jit_sym_name`, `freebytes()`, `gensym()`, `jit_linklist_chuck()`, `jit_linklist_getindex()`, `jit_linklist_getsize()`, `jit_object_free()`, and `object_attr_setvalueof()`.

Referenced by `jit_class_new()`, and `max_jit_classex_standard_wrap()`.

Here is the call graph for this function:

**36.72.2.16 `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

<code>x</code>	object pointer
<code>s</code>	name as used via argument

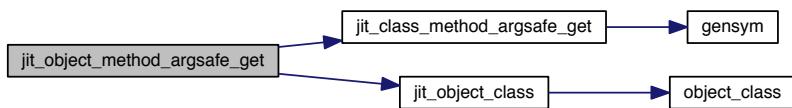
Returns

If successful, name of method to map the argument name to. Otherwise, NULL.

References `jit_class_method_argsafe_get()`, and `jit_object_class()`.

Referenced by `max_jit_attr_args()`.

Here is the call graph for this function:



36.72.2.17 void* jit_object_method_imp (void * *x*, void * *s*, void * *p1*, void * *p2*, void * *p3*, void * *p4*, void * *p5*, void * *p6*, void * *p7*, void * *p8*)

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

<i>x</i>	object pointer
<i>s</i>	method name
<i>p1</i>	untyped arguments passed on to the method
<i>p2</i>	untyped arguments passed on to the method
<i>p3</i>	untyped arguments passed on to the method
<i>p4</i>	untyped arguments passed on to the method
<i>p5</i>	untyped arguments passed on to the method
<i>p6</i>	untyped arguments passed on to the method
<i>p7</i>	untyped arguments passed on to the method
<i>p8</i>	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.

36.72.2.18 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

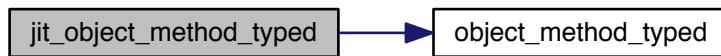
<i>x</i>	object pointer
<i>s</i>	method name
<i>ac</i>	argument count
<i>av</i>	argument vector
<i>rv</i>	return value for A_GIMMEBACK methods

Returns

method dependent, but uses void * as a super type.

References object_method_typed().

Here is the call graph for this function:



36.72.2.19 `void* jit_object_new_imp (void * cn, void * p1, void * p2, void * p3, void * p4, void * p5, void * p6, void * p7, void * p8, void * dummy)`

Instantiates an object specified by class name.

This function may be used to create instances of any Jitter object.

Parameters

<i>cn</i>	class name
<i>p1</i>	untyped arguments passed on to the constructor
<i>p2</i>	untyped arguments passed on to the constructor
<i>p3</i>	untyped arguments passed on to the constructor
<i>p4</i>	untyped arguments passed on to the constructor
<i>p5</i>	untyped arguments passed on to the constructor
<i>p6</i>	untyped arguments passed on to the constructor
<i>p7</i>	untyped arguments passed on to the constructor
<i>p8</i>	untyped arguments passed on to the constructor
<i>dummy</i>	unused

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.

36.72.2.20 `t_jit_err jit_object_notify (void * x, t_symbol * s, void * data)`

Notifies all client objects for a named server object.

Parameters

<i>x</i>	server object pointer
<i>s</i>	notification message
<i>data</i>	message specific data

Returns

`t_jit_err` error code

References `object_notify()`.

Referenced by `jit_matrix_setinfo()`, `jit_matrix_setinfo_ex()`, and `jit_ob3d_draw_chunk()`.

Here is the call graph for this function:



36.72.2.21 void* jit_object_register(void *x, t_symbol *s)

Registers an object in the named object registry.

Parameters

<code>x</code>	object pointer
<code>s</code>	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.

References `object_register()`.

Referenced by `max_jit_mop_inputs()`, `max_jit_mop_notify()`, and `max_jit_mop_outputs()`.

Here is the call graph for this function:



36.72.2.22 t_jit_err jit_object_unregister(void *x)

Unregisters an object from the named object registry.

Parameters

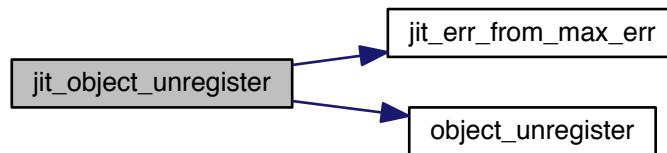
x	object pointer
---	----------------

Returns

t_jit_err error code

References jit_err_from_max_err(), and object_unregister().

Here is the call graph for this function:



36.73 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.

36.73.1 Detailed Description

36.73.2 Function Documentation

36.73.2.1 [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

References MAX_ERR_DUPLICATE, MAX_ERR_GENERIC, MAX_ERR_INVALID_PTR, MAX_ERR_NONE, and MAX_ERR_OUT_OF_MEM.

Referenced by `jit_attr_setchar_array()`, `jit_attr_setdouble_array()`, `jit_attr_setfloat()`, `jit_attr_setfloat_array()`, `jit_attr_setlong()`, `jit_attr_setlong_array()`, `jit_attr_setsym()`, `jit_attr_setsym_array()`, `jit_class_addadornment()`, `jit_class_addattr()`, `jit_class_addmethod()`, `jit_class_adddtypedwrapper()`, `jit_class_free()`, `jit_class_method_addargsafe()`, `jit_class_register()`, `jit_object_detach()`, `jit_object_free()`, and `jit_object_unregister()`.

36.73.2.2 void jit_error_code (void * x, t_jit_err v)

Sends error code based error message to Max console (safe from all threads)

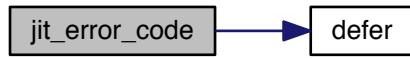
Parameters

<code>x</code>	object pointer
<code>v</code>	error code

References `defer()`.

Referenced by `max_jit_attr_getdump()`, and `max_jit_mop_jit_matrix()`.

Here is the call graph for this function:

**36.73.2.3 void jit_error_sym (void * x, t_symbol * s)**

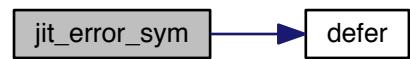
Sends symbol based error message to Max console (safe from all threads)

Parameters

<code>x</code>	object pointer
<code>s</code>	error message symbol

References `defer()`.

Here is the call graph for this function:



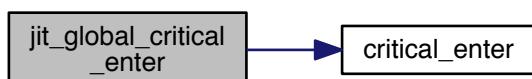
36.73.2.4 void jit_global_critical_enter (void)

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 Max'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.

References `critical_enter()`.

Here is the call graph for this function:



36.73.2.5 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 Max'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.

References `critical_exit()`.

Here is the call graph for this function:



36.73.2.6 void jit_post_sym (void * x, t_symbol * s)

Sends symbol based message to Max console (safe from all threads)

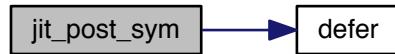
Parameters

x	object pointer
---	----------------

s	message symbol
---	----------------

References defer().

Here is the call graph for this function:



36.73.2.7 long jit_rand(void)

Generates a random value as a signed long integer.

Returns

random value

36.73.2.8 void jit_rand_setseed(long n)

Sets global random number generator seed.

Parameters

n	seed
---	------

References systime_ticks().

Here is the call graph for this function:



36.73.2.9 float swapf32(float f)

Byte swaps 32 bit floating point number.

Parameters

<code>f</code>	input float
----------------	-------------

Returns

byte swapped float

36.73.2.10 double swapf64 (double f)

Byte swaps 64 bit floating point number.

Parameters

<code>f</code>	input double
----------------	--------------

Returns

byte swapped double

36.74 Linked List Module

Collaboration diagram for Linked List Module:



Functions

- `void * jit_linklist_new (void)`
Constructs instance of t_jit_linklist.
- `t_atom_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.
- `t_atom_long jit_linklist_objptr2index (t_jit_linklist *x, void *p)`
Retrieves the list index for an object pointer.
- `t_atom_long jit_linklist_makearray (t_jit_linklist *x, void **a, long max)`
Flatten the linked list into an array.
- `t_atom_long jit_linklist_insertindex (t_jit_linklist *x, void *o, long index)`
Insert object at specified index.
- `t_atom_long jit_linklist_append (t_jit_linklist *x, void *o)`
Append object to the end of the linked list.
- `t_atom_long jit_linklist_deleteindex (t_jit_linklist *x, long index)`
Delete object at specified index, freeing the object.
- `t_atom_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.
- `t_atom_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.

36.74.1 Detailed Description

36.74.2 Function Documentation

36.74.2.1 [t_atom_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 recommended 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.

References [linklist_append\(\)](#).

Referenced by [jit_matrix_op\(\)](#), [max_jit_classex_addattr\(\)](#), and [max_jit_obex_proxy_new\(\)](#).

Here is the call graph for this function:



36.74.2.2 [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, freeing the objects, use the [jit_linklist_clear](#) method.

Parameters

<i>x</i>	t_jit_linklist object pointer
----------	-------------------------------

Warning

While exported, it is recommended 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.

References linklist_chuck().

Referenced by jit_matrix_op(), and jit_object_importattrs().

Here is the call graph for this function:

**36.74.2.3 t_atom_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

<i>x</i>	t_jit_linklist object pointer
<i>index</i>	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.

References linklist_chuckindex().

Here is the call graph for this function:

**36.74.2.4 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

<code>x</code>	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.

References linklist_clear().

Here is the call graph for this function:

**36.74.2.5 t_atom_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

<i>x</i>	t_jit_linklist object pointer
<i>index</i>	index to delete (zero based)

Returns

index deleted, or -1 if unsuccessful

Warning

While exported, it is recommended 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.

References linklist_deleteindex().

Here is the call graph for this function:



36.74.2.6 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

<i>x</i>	t_jit_linklist object pointer
<i>out</i>	pointer to linked list containing all objects found (set to NULL, if not found)
<i>cmpfn</i>	comparison function pointer (should return 1 if object matches data, otherwise 0)
<i>cmpdata</i>	opaque data used in comparison function

Warning

While exported, it is recommended 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.

References `linklist_findall()`.

Here is the call graph for this function:



36.74.2.7 `t_atom_long jit_linklist_findcount(t_jit_linklist *x, long cmpfnvoid *, void *, void *cmpdata)`

Retrieves the number of objects that satisfy the comparison function.

Parameters

<code>x</code>	<code>t_jit_linklist</code> object pointer
<code>cmpfn</code>	comparison function pointer (should returns 1 if object matches data, otherwise 0)
<code>cmpdata</code>	opaque data used in comparison function

Returns

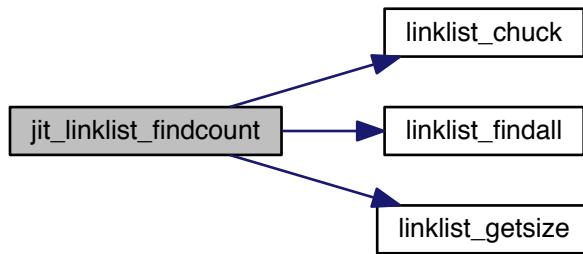
number of objects that satisfy the comparison function

Warning

While exported, it is recommended 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.

References linklist_chuck(), linklist_findall(), and linklist_getsize().

Here is the call graph for this function:



36.74.2.8 void jit_linklist_findfirst (t_jit_linklist * x, void ** o, long cmpfnvoid *, void *, void * cmpdata)

Retrieves the first object that satisfies the comparison function.

Parameters

<i>x</i>	t_jit_linklist object pointer
<i>o</i>	pointer to object pointer found (set to NULL, if not found)
<i>cmpfn</i>	comparison function pointer (should returns 1 if object matches data, otherwise 0)
<i>cmpdata</i>	opaque data used in comparison function

Warning

While exported, it is recommended 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.

References linklist_findfirst().

Referenced by max_jit_obex_attr_get(), and max_jit_obex_attr_set().

Here is the call graph for this function:



36.74.2.9 void* jit_linklist_getindex (t_jit_linklist * x, long index)

Retrieves the object at the specified list index.

Parameters

<i>x</i>	t_jit_linklist object pointer
<i>index</i>	list index ()

Returns

object pointer

Warning

While exported, it is recommended 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.

References linklist_getindex().

Referenced by jit_object_importattrs().

Here is the call graph for this function:



36.74.2.10 t_atom_long jit_linklist_getsize (t_jit_linklist * x)

Retrieves the linked list size.

Parameters

<i>x</i>	t_jit_linklist object pointer
----------	-------------------------------

Returns

linked list size

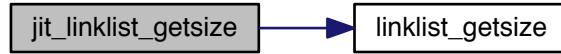
Warning

While exported, it is recommended 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 getsize method.

References linklist_getsize().

Referenced by jit_object_importattrs().

Here is the call graph for this function:



36.74.2.11 t_atom_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.

References linklist_insertindex().

Here is the call graph for this function:



36.74.2.12 t_atom_long jit_linklist_makearray (t_jit_linklist * x, void ** a, long max)

Flatten the linked list into an array.

Parameters

<i>x</i>	t_jit_linklist object pointer
<i>a</i>	array pointer
<i>max</i>	maximum array size

Returns

number of object pointers copied into array

Warning

While exported, it is recommended 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.

References linklist_makearray().

Here is the call graph for this function:



36.74.2.13 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

<i>x</i>	t_jit_linklist object pointer
<i>s</i>	method name
...	untyped arguments

Warning

While exported, it is recommended 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.

References `linklist_methodall()`.

Here is the call graph for this function:



36.74.2.14 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

<code>x</code>	<code>t_jit_linklist</code> object pointer
<code>i</code>	index
<code>s</code>	method name
...	untyped arguments

Returns

method return value

Warning

While exported, it is recommended 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.

References `linklist_methodindex()`.

Here is the call graph for this function:



36.74.2.15 void* jit_linklist_new (void)

Constructs instance of t_jit_linklist.

Returns

t_jit_linklist object pointer

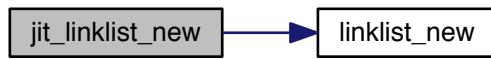
Warning

While exported, it is recommend to use jit_object_new to construct a t_jit_linklist object.

References linklist_new().

Referenced by jit_matrix_op(), max_jit_classex_addattr(), and max_jit_obex_proxy_new().

Here is the call graph for this function:



36.74.2.16 t_atom_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.

References linklist_objptr2index().

Here is the call graph for this function:



36.74.2.17 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.

References linklist_reverse().

Here is the call graph for this function:



36.74.2.18 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.

References linklist_rotate().

Here is the call graph for this function:



36.74.2.19 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.

References linklist_shuffle().

Here is the call graph for this function:

**36.74.2.20 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.

References linklist_sort().

Here is the call graph for this function:

**36.74.2.21 void jit_linklist_swap (t_jit_linklist * x, long a, long b)**

Swap list location of the indeces specified.

Parameters

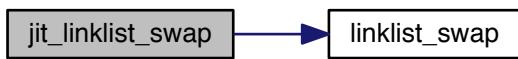
<i>x</i>	t_jit_linklist object pointer
<i>a</i>	index a
<i>b</i>	index b

Warning

While exported, it is recommended 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.

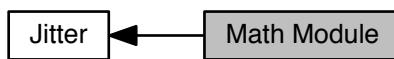
References linklist_swap().

Here is the call graph for this function:



36.75 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)`

- `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 $j_{1,0}$ Bessel function.
- `double jit_math_p1 (double x)`
Calcuates the p_1 Bessel function.
- `double jit_math_q1 (double x)`
Calcuates the q_1 Bessel function.
- `double jit_math_j1 (double x)`
Calcuates the j_1 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.

36.75.1 Detailed Description

36.75.2 Function Documentation

36.75.2.1 double jit_math_acos (double x)

Calculates the arccosine.

Parameters

x	input
---	-------

Returns

output

36.75.2.2 double jit_math_acosh (double x)

Calculates the hyperbolic arccosine.

Parameters

x	input
---	-------

Returns

output

36.75.2.3 double jit_math_asin (double x)

Calculates the arcsine.

Parameters

x	input
---	-------

Returns

output

36.75.2.4 double jit_math_asinh (double x)

Calculates the hyperbolic arcsine.

Parameters

x	input
---	-------

Returns

output

36.75.2.5 double jit_math_atan (double x)

Calculates the arctangent.

Parameters

x	input
---	-------

Returns

output

36.75.2.6 double jit_math_atan2 (double y, double x)

Calculates the four quadrant arctangent.

Parameters

y	input
x	input

Returns

output

36.75.2.7 double jit_math_atanh (double x)

Calculates the hyperbolic arctangent.

Parameters

x	input
---	-------

Returns

output

36.75.2.8 double jit_math_ceil (double x)

Calculates the ceiling.

Parameters

x	input
---	-------

Returns

output

36.75.2.9 double jit_math_cos (double x)

Calculates the cosine.

Parameters

x	input
---	-------

Returns

output

Referenced by jit_math_j1().

36.75.2.10 double jit_math_cosh (double x)

Calculates the hyperbolic cosine.

Parameters

x	input
---	-------

Returns

output

36.75.2.11 double jit_math_exp (double x)

Calculates the exponent.

Parameters

x	input
---	-------

Returns

output

36.75.2.12 double jit_math_exp2 (double x)

Calculates the exponent base 2.

Parameters

x	input
---	-------

Returns

output

36.75.2.13 double jit_math_expm1 (double x)

Calculates the exponent minus 1.

Parameters

x	input
---	-------

Returns

output

36.75.2.14 float jit_math_fast_acos (float x)

Calculates the arccosine by fast approximation.

Absolute error of 6.8e-05 for [0, 1]

Parameters

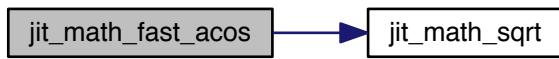
x	input
---	-------

Returns

output

References jit_math_sqrt().

Here is the call graph for this function:

**36.75.2.15 float jit_math_fast_asin (float x)**

Calculates the arcsine by fast approximation.

Absolute error of 6.8e-05 for [0, 1]

Parameters

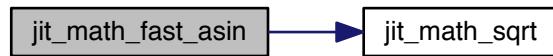
x	input
---	-------

Returns

output

References jit_math_sqrt().

Here is the call graph for this function:

**36.75.2.16 float jit_math_fast_atan (float x)**

Calculates the arctangent by fast approximation.

Absolute error of 1.43e-08 for [-1, 1]

Parameters

x	input
---	-------

Returns

output

36.75.2.17 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

36.75.2.18 float jit_math_fast_invsqrt (float x)

Calculates the inverse square root by fast approximation.

Parameters

x	input
---	-------

Returns

output

36.75.2.19 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

36.75.2.20 float jit_math_fast_sqrt (float n)

Calculates the square root by fast approximation.

Parameters

n	input
---	-------

Returns

output

36.75.2.21 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

36.75.2.22 double jit_math_floor (double x)

Calculates the floor.

Parameters

x	input
---	-------

Returns

output

36.75.2.23 double jit_math_fmod (double x, double y)

Calculates the floating point x modulo y.

Parameters

<i>x</i>	input
<i>y</i>	input

Returns

output

36.75.2.24 double jit_math_fold(double *x*, double *lo*, double *hi*)Calculates the fold of *x* between *lo* and *hi*.**Parameters**

<i>x</i>	input
<i>lo</i>	lower bound
<i>hi</i>	upper bound

Returns

output

36.75.2.25 double jit_math_hypot(double *x*, double *y*)

Calculates the hypotenuse.

Parameters

<i>x</i>	input
<i>y</i>	input

Returns

output

36.75.2.26 long jit_math_is_finite(float *v*)

Checks if input is finite.

Parameters

<i>v</i>	input
----------	-------

Returns

1 if finite. Otherwise, 0.

Referenced by jit_math_is_valid().

36.75.2.27 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.

Referenced by jit_math_is_valid().

36.75.2.28 long jit_math_is_poweroftwo (long x)

Checks if input is a power of two.

Parameters

x	input
---	-------

Returns

1 if finite. Otherwise, 0.

36.75.2.29 long jit_math_is_valid (float v)

Checks if input is both finite and a number.

Parameters

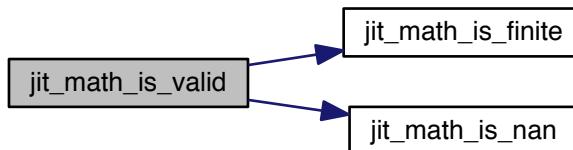
v	input
---	-------

Returns

1 if valid. Otherwise, 0.

References jit_math_is_finite(), and jit_math_is_nan().

Here is the call graph for this function:

**36.75.2.30 double jit_math_j1 (double x)**

Calculates the j1 Bessel function.

Parameters

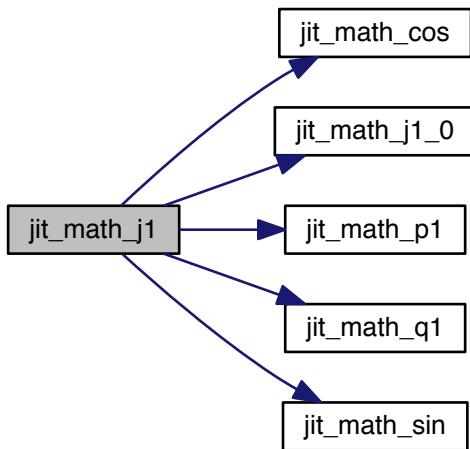
x	input
---	-------

Returns

output

References jit_math_cos(), jit_math_j1_0(), jit_math_p1(), jit_math_q1(), and jit_math_sin().

Here is the call graph for this function:

**36.75.2.31 double jit_math_j1_0 (double x)**

Calcuates the j1_0 Bessel function.

Parameters

x	input
---	-------

Returns

output

Referenced by jit_math_j1().

36.75.2.32 double jit_math_log (double x)

Calculates the logarithm.

Parameters

x	input
---	-------

Returns

output

36.75.2.33 double jit_math_log10 (double x)

Calculates the logarithm base 10.

Parameters

x	input
---	-------

Returns

output

36.75.2.34 double jit_math_log2 (double x)

Calculates the logarithm base 2.

Parameters

x	input
---	-------

Returns

output

36.75.2.35 double jit_math_p1 (double x)

Calcuates the p1 Bessel function.

Parameters

x	input
---	-------

Returns

output

Referenced by jit_math_j1().

36.75.2.36 double jit_math_pow (double x, double y)

Calculates x raised to the y power.

Parameters

x	input
y	input

Returns

output

36.75.2.37 double jit_math_q1 (double x)

Calcuates the q1 Bessel function.

Parameters

x	input
---	-------

Returns

output

Referenced by jit_math_j1().

36.75.2.38 double jit_math_round (double x)

Rounds the input.

Parameters

x	input
---	-------

Returns

output

36.75.2.39 unsigned long jit_math_roundup_poweroftwo (unsigned long x)

Rounds up to the nearest power of two.

Parameters

x	input
---	-------

Returns

output

36.75.2.40 double jit_math_sin (double x)

Calculates the sine.

Parameters

x	input
---	-------

Returns

output

Referenced by jit_math_j1().

36.75.2.41 double jit_math_sinh (double x)

Calculates the hyperbolic sine.

Parameters

x	input
---	-------

Returns

output

36.75.2.42 double jit_math_sqrt (double x)

Calculates the square root.

Parameters

x	input
---	-------

Returns

output

Referenced by jit_math_fast_acos(), and jit_math_fast_asin().

36.75.2.43 double jit_math_tan (double x)

Calculates the tangent.

Parameters

x	input
---	-------

Returns

output

36.75.2.44 double jit_math_tanh (double x)

Calculates the hyperbolic tangent.

Parameters

x	input
---	-------

Returns

output

36.75.2.45 double jit_math_trunc (double x)

Truncates the input.

Parameters

x	input
---	-------

Returns

output

36.75.2.46 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

36.76 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)`

- `t_jit_err jit_matrix_setcell (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_getcell (t_jit_matrix **x, t_symbol *s, long argc, t_atom *argv, long *rac, t_atom **rav)`

Sets 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)`

Gets cell at index to the value provided.
- `t_jit_err jit_matrix_fillplane (t_jit_matrix **x, t_symbol *s, long argc, t_atom *argv)`

Sets all cells to the value provided.
- `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.

36.76.1 Detailed Description

36.76.2 Function Documentation

36.76.2.1 void jit_linklist_free (`t_jit_linklist * x`)

Frees instance of `t_jit_linklist`.

should never be called directly

Parameters

<code>x</code>	<code>t_jit_linklist</code> object pointer
----------------	--

Returns

`t_jit_err` error code

Warning

Use `jit_object_free` instead.

References `linklist_clear()`.

Here is the call graph for this function:



36.76.2.2 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

<i>x</i>	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.

36.76.2.3 t_jit_err jit_matrix_data (t_jit_matrix * *x*, void * *data*)

Sets matrix data pointer.

Parameters

<i>x</i>	t_jit_matrix object pointer
<i>data</i>	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.

36.76.2.4 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

<i>x</i>	t_jit_matrix object pointer
<i>s</i>	message symbol pointer
<i>argc</i>	argument count
<i>argv</i>	argument vector

Returns

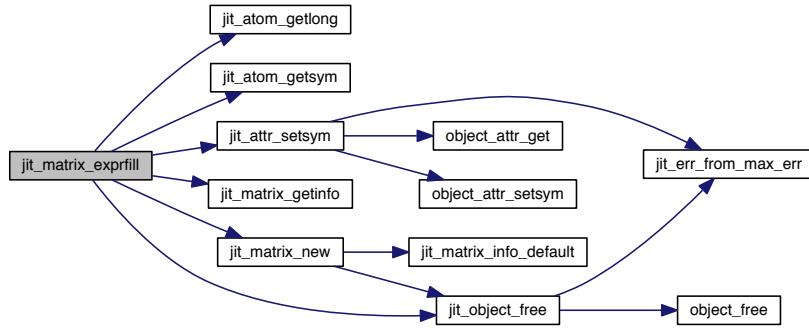
t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

References `_jit_sym_float32`, `_jit_sym_frommatrix`, `_jit_sym_matrix_calc`, `_jit_sym_nothing`, `A_SYM`, `t_matrix` ← `conv_info::flags`, `jit_atom_getlong()`, `jit_atom_getsym()`, `jit_attr_setsym()`, `jit_matrix_getinfo()`, `JIT_MATRIX_MA` ← `X_PLANECOUNT`, `jit_matrix_new()`, `jit_object_free()`, `t_jit_matrix_info::planecount`, `t_matrix_conv_info::planemap`, and `t_jit_matrix_info::type`.

Here is the call graph for this function:



36.76.2.5 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

<i>x</i>	t_jit_matrix object pointer
<i>s</i>	message symbol pointer
<i>argc</i>	argument count
<i>argv</i>	argument vector

Returns

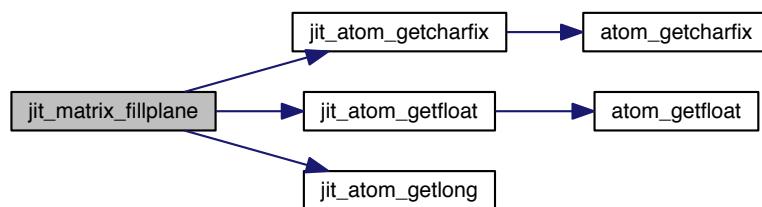
t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

References jit_sym_char, jit_sym_float32, jit_sym_float64, jit_sym_long, jit_atom_getcharfix(), jit_atom_getfloat(), and jit_atom_getlong().

Here is the call graph for this function:



36.76.2.6 t jit_err jit_matrix_free (t jit_matrix * x)

Frees instance of t jit_matrix.

Parameters

x	t_jit_matrix object pointer
---	-----------------------------

Returns

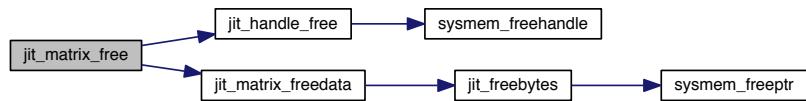
t_jit_err error code

Warning

Use jit_object_free instead.

References jit_handle_free(), JIT_MATRIX_DATA_HANDLE, JIT_MATRIX_DATA_REFERENCE, and jit_matrix_free().

Here is the call graph for this function:

**36.76.2.7 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

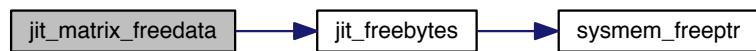
Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

References `jit_freebytes()`, and `JIT_MATRIX_DATA_REFERENCE`.

Referenced by `jit_matrix_free()`.

Here is the call graph for this function:



```
36.76.2.8 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

<i>dst_matrix</i>	destination t_jit_matrix object pointer
<i>src_matrix</i>	destination t_jit_matrix object pointer
<i>mcinfo</i>	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.

References _jit_sym_getdata, _jit_sym_getinfo, _jit_sym_lock, t_jit_matrix_info::dim, t_jit_matrix_info::dimcount, t_jit_matrix_conv_info::dstdimend, t_matrix_conv_info::dstdimstart, t_matrix_conv_info::flags, JIT_MATRIX_MAX_DIMCOUNT, MAX, t_jit_matrix_info::size, t_matrix_conv_info::srccdimend, and t_matrix_conv_info::srccdimstart.

36.76.2.9 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

<i>x</i>	t_jit_matrix object pointer
<i>s</i>	message symbol pointer
<i>argc</i>	argument count
<i>argv</i>	argument vector
<i>rac</i>	return value atom count
<i>rav</i>	return value atom vector

Returns

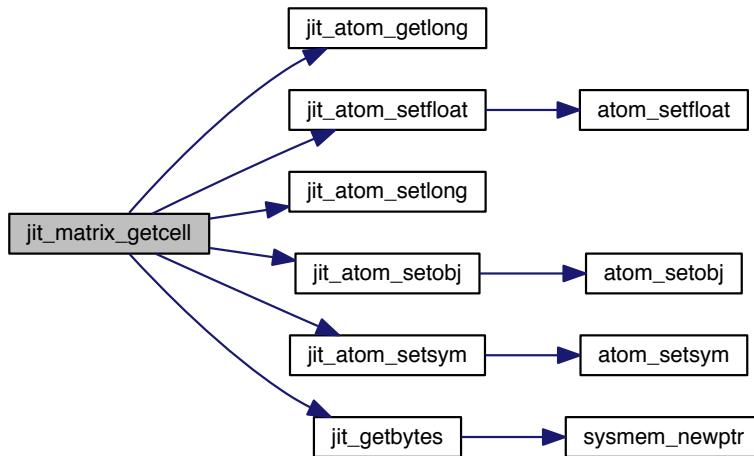
t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

References `_jit_sym_char`, `_jit_sym_float32`, `_jit_sym_float64`, `_jit_sym_long`, `_jit_sym_object`, `_jit_sym_symbol`, `_jit_sym_val`, `jit_atom_getlong()`, `jit_atom_setfloat()`, `jit_atom_setlong()`, `jit_atom_setobj()`, `jit_atom_setsym()`, `jit_getbytes()`, and `JIT_MATRIX_MAX_DIMCOUNT`.

Here is the call graph for this function:



36.76.2.10 `t_jit_err jit_matrix_getdata (t_jit_matrix *x, void ** data)`

Retrieves matrix data pointer.

Parameters

<code>x</code>	<code>t_jit_matrix</code> object pointer
<code>data</code>	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`.

36.76.2.11 `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

<i>x</i>	<code>t_jit_matrix</code> object pointer
<i>info</i>	<code>t_jit_matrix_info</code> 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`.

Referenced by `jit_matrix_exprfill()`, and `jit_matrix_op()`.

36.76.2.12 `t_jit_err jit_matrix_info_default (t_jit_matrix_info * info)`

Initializes matrix info struct to default values.

Parameters

<i>info</i>	<code>t_jit_matrix_info</code> 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`.

References `t_jit_matrix_info::dim`, `t_jit_matrix_info::dimcount`, `t_jit_matrix_info::dimstride`, `t_jit_matrix_info::flags`, `JIT_MATRIX_MAX_DIMCOUNT`, `t_jit_matrix_info::planecount`, `t_jit_matrix_info::size`, and `t_jit_matrix_info::type`.

Referenced by `jit_glchunk_grid_new()`, `jit_glchunk_new()`, `jit_matrix_new()`, `max_jit_mop_inputs()`, `max_jit_mop←matrix_args()`, and `max_jit_mop_outputs()`.

36.76.2.13 `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

<i>x</i>	<code>t_jit_matrix</code> object pointer
<i>s</i>	message symbol pointer
<i>argc</i>	argument count
<i>argv</i>	argument vector

Returns

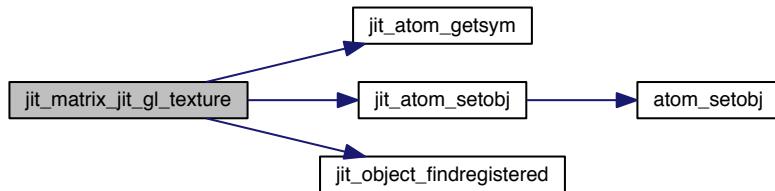
`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

References `_jit_sym_nothing`, `jit_atom_getsym()`, `jit_atom_setobj()`, and `jit_object_findregistered()`.

Here is the call graph for this function:



36.76.2.14 void * jit_matrix_new (t_jit_matrix_info * info)

Constructs instance of t_jit_matrix.

Parameters

<code>info</code>	<code>t_jit_matrix_info</code> struct pointer
-------------------	---

Returns

`t_jit_matrix` object pointer

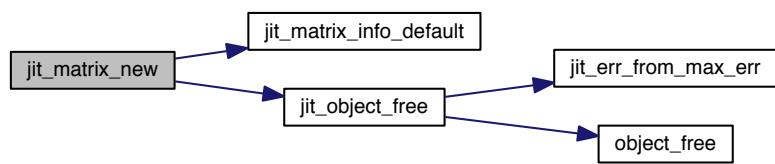
Warning

This function is not exported, but is provided for reference when calling via jit_object_new.

References `t_jit_matrix_info::flags`, `JIT_MATRIX_DATA_FLAGS_USE`, `JIT_MATRIX_DATA_HANDLE`, `JIT_MATRIX_DATA_REFERENCE`, `jit_matrix_info_default()`, and `jit_object_free()`.

Referenced by `jit_matrix_exprfill()`, and `jit_matrix_op()`.

Here is the call graph for this function:



36.76.2.15 void * jit_matrix_newcopy(t_jit_matrix * *copyme*)

Constructs instance of t_jit_matrix, copying from input.

Parameters

<i>copyme</i>	t_jit_matrix object pointer
---------------	-----------------------------

Returns

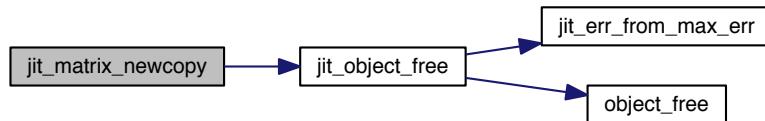
t_jit_matrix object pointer

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

References JIT_MATRIX_DATA_REFERENCE, and jit_object_free().

Here is the call graph for this function:

**36.76.2.16 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

<i>x</i>	t_jit_matrix object pointer
<i>s</i>	message symbol pointer
<i>argc</i>	argument count
<i>argv</i>	argument vector

Returns

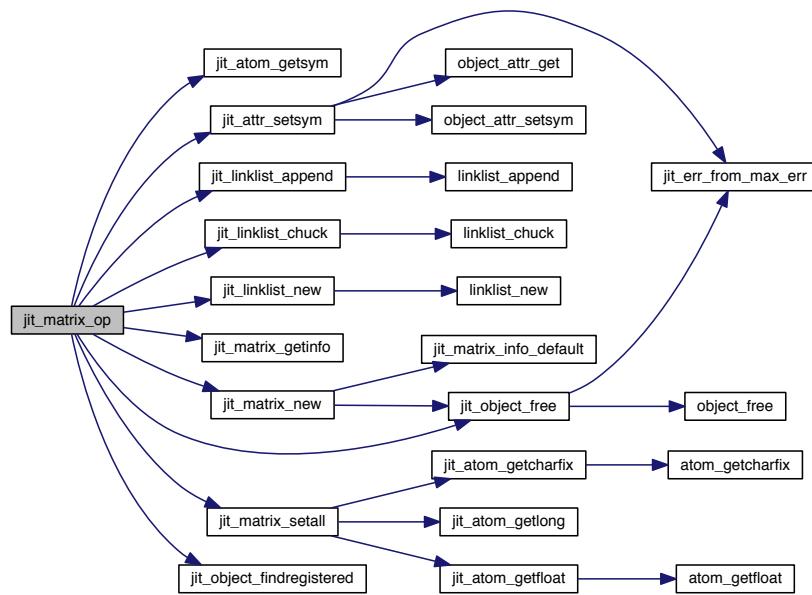
t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via jit_object_method on an instance of t_jit_matrix.

References _jit_sym_class_jit_matrix, _jit_sym_matrix_calc, _jit_sym_nothing, A_FLOAT, A_LONG, A_OBJ, A←_SYM, jit_atom_getsym(), jit_attr_setsym(), jit_linklist_append(), jit_linklist_chuck(), jit_linklist_new(), jit_matrix←getinfo(), jit_matrix_new(), jit_matrix_setall(), jit_object_findregistered(), jit_object_free(), and t_symbol::s_name.

Here is the call graph for this function:



36.76.2.17 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

<i>x</i>	t_jit_matrix object pointer
<i>s</i>	message symbol pointer
<i>argc</i>	argument count
<i>argv</i>	argument vector

Returns

t_jit_err error code

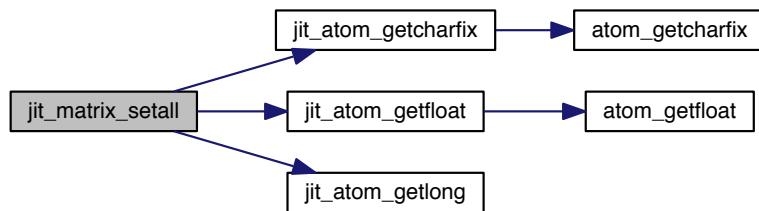
Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

References `_jit_sym_char`, `_jit_sym_float32`, `_jit_sym_float64`, `_jit_sym_long`, `jit_atom_getcharfix()`, `jit_atom_getfloat()`, `jit_atom_getlong()`, and `JIT_MATRIX_MAX_PLANECOUNT`.

Referenced by `jit_matrix_op()`.

Here is the call graph for this function:

**36.76.2.18 `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

<code>x</code>	<code>t_jit_matrix</code> object pointer
<code>s</code>	message symbol pointer
<code>argc</code>	argument count
<code>argv</code>	argument vector

Returns

`t_jit_err` error code

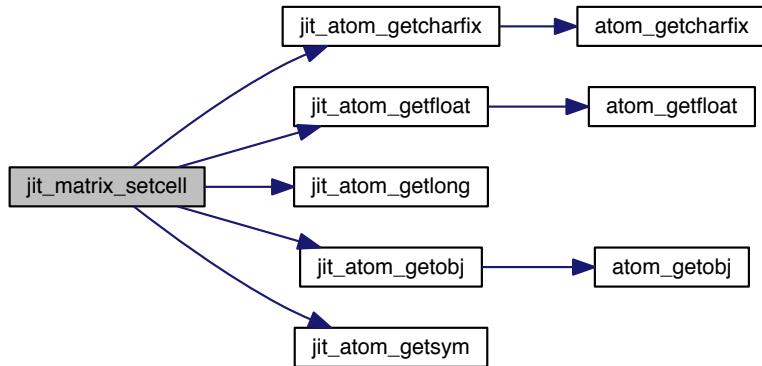
Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

References `_jit_sym_char`, `_jit_sym_float32`, `_jit_sym_float64`, `_jit_sym_long`, `_jit_sym_object`, `_jit_sym_plane`, `_jit_sym_symbol`, `_jit_sym_val`, `A_SYM`, `jit_atom_getcharfix()`, `jit_atom_getfloat()`, `jit_atom_getlong()`, `jit_atom_getobj()`, `jit_atom_getsym()`, `JIT_MATRIX_MAX_DIMCOUNT`, and `word:w_sym`.

Referenced by `jit_matrix_setcell1d()`, `jit_matrix_setcell2d()`, `jit_matrix_setcell3d()`, `jit_matrix_setplane1d()`, `jit_matrix_setplane2d()`, and `jit_matrix_setplane3d()`.

Here is the call graph for this function:



36.76.2.19 `t_jit_err jit_matrix_setcellId (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

<i>x</i>	<code>t_jit_matrix</code> object pointer
<i>s</i>	message symbol pointer
<i>argc</i>	argument count
<i>argv</i>	argument vector

Returns

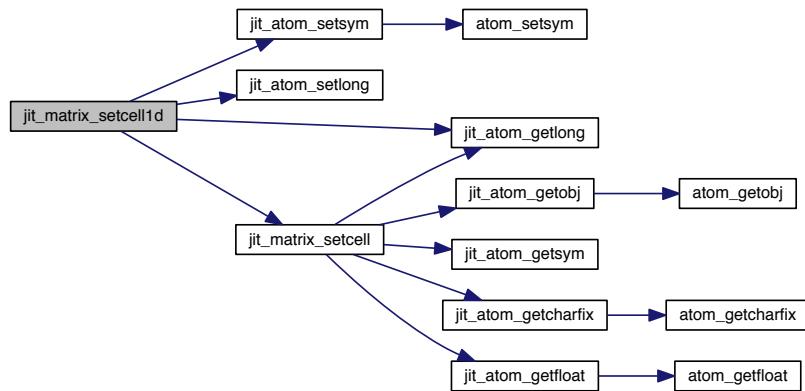
`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

References `_jit_sym_val`, `jit_atom_getlong()`, `jit_atom_setlong()`, `jit_atom_setsym()`, and `jit_matrix_setcell()`.

Here is the call graph for this function:



36.76.2.20 `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

<code>x</code>	<code>t_jit_matrix</code> object pointer
<code>s</code>	message symbol pointer
<code>argc</code>	argument count
<code>argv</code>	argument vector

Returns

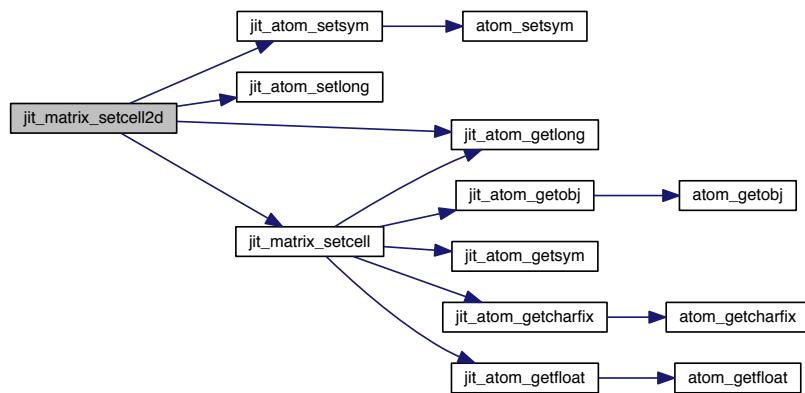
`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

References `_jit_sym_val`, `jit_atom_getlong()`, `jit_atom_setlong()`, `jit_atom_setsym()`, and `jit_matrix_setcell()`.

Here is the call graph for this function:



36.76.2.21 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

<code>x</code>	<code>t_jit_matrix</code> object pointer
<code>s</code>	message symbol pointer
<code>argc</code>	argument count
<code>argv</code>	argument vector

Returns

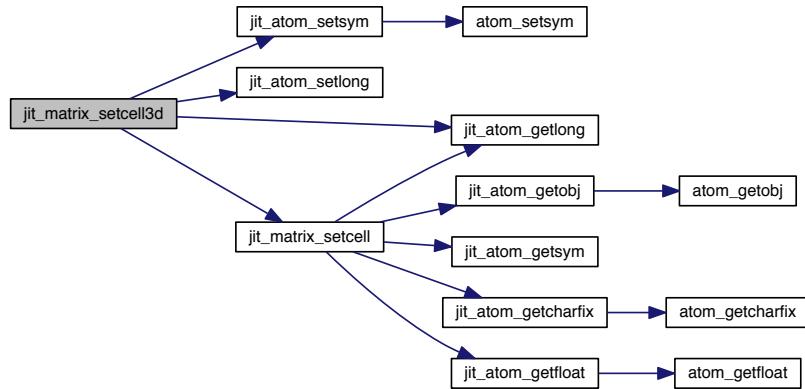
`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

References `_jit_sym_val`, `jit_atom_getlong()`, `jit_atom_setlong()`, `jit_atom_setsym()`, and `jit_matrix_setcell()`.

Here is the call graph for this function:



36.76.2.22 `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

<code>x</code>	<code>t_jit_matrix</code> object pointer
<code>info</code>	<code>t_jit_matrix_info</code> pointer

Returns

`t_jit_err` error code

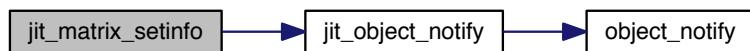
Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

References `_jit_sym_setinfo`, and `jit_object_notify()`.

Referenced by `jit_matrix_setinfo_ex()`.

Here is the call graph for this function:



36.76.2.23 `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

<code>x</code>	<code>t_jit_matrix</code> object pointer
<code>info</code>	<code>t_jit_matrix_info</code> pointer

Returns

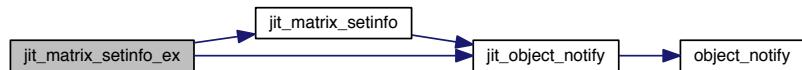
`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

References `_jit_sym_modified`, `_jit_sym_nothing`, `t_jit_matrix_info::flags`, `JIT_MATRIX_DATA_REFERENCE`, `jit_matrix_setinfo()`, `jit_object_notify()`, and `t_jit_matrix_info::size`.

Here is the call graph for this function:



36.76.2.24 `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

<code>x</code>	<code>t_jit_matrix</code> object pointer
<code>s</code>	message symbol pointer
<code>argc</code>	argument count
<code>argv</code>	argument vector

Returns

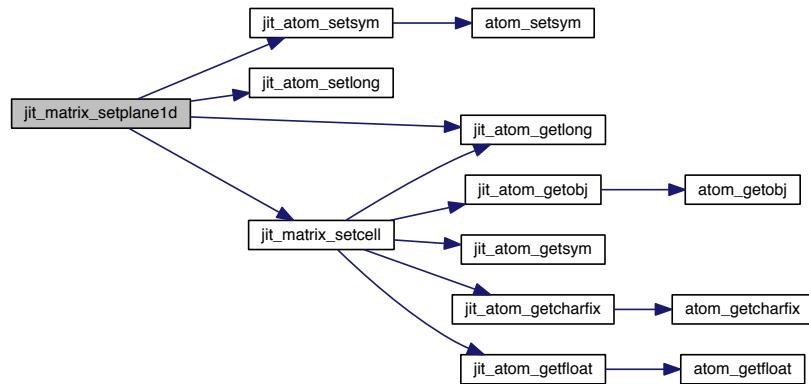
`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

References `_jit_sym_plane`, `_jit_sym_val`, `jit_atom_getlong()`, `jit_atom_setlong()`, `jit_atom_setsym()`, and `jit_matrix_setcell()`.

Here is the call graph for this function:



36.76.2.25 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

<i>x</i>	t_jit_matrix object pointer
<i>s</i>	message symbol pointer
<i>argc</i>	argument count
<i>argv</i>	argument vector

Returns

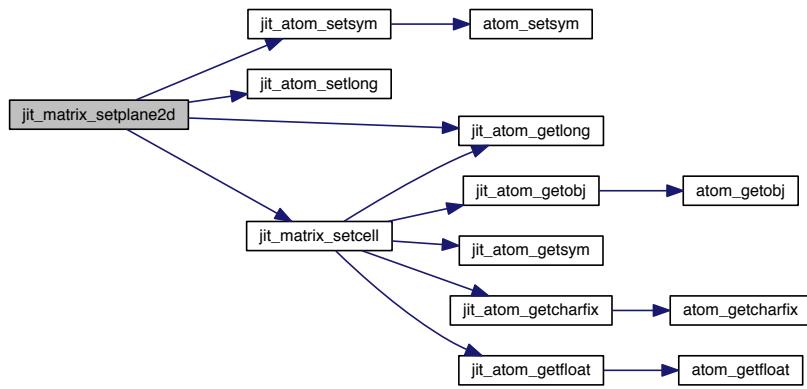
t_jit_err error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

References `_jit_sym_plane`, `_jit_sym_val`, `jit_atom_getlong()`, `jit_atom_setlong()`, `jit_atom_setsym()`, and `jit_matrix_setcell()`.

Here is the call graph for this function:



36.76.2.26 `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

<code>x</code>	<code>t_jit_matrix</code> object pointer
<code>s</code>	message symbol pointer
<code>argc</code>	argument count
<code>argv</code>	argument vector

Returns

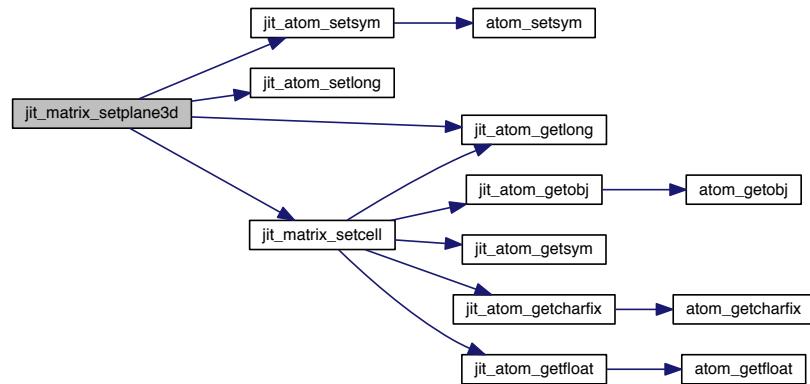
`t_jit_err` error code

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_matrix`.

References `_jit_sym_plane`, `_jit_sym_val`, `jit_atom_getlong()`, `jit_atom_setlong()`, `jit_atom_setsym()`, and `jit_matrix_setcell()`.

Here is the call graph for this function:



36.77 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 attribute 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 and 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)

- long `max_jit_obex_inletnumber_get` (void *x)

Sets the wrapped Jitter object for a Max wrapper object.
- void `max_jit_obex_inletnumber_set` (void *x, long inletnumber)

Retrieves the current inlet number used by inlet proxies.
- t_jit_err `max_jit_obex_proxy_new` (void *x, long c)

Sets the current inlet number used by inlet proxies.
- void `max_jit_obex_dumpout_set` (void *x, void *outlet)

Creates a new proxy inlet.
- void `max_jit_obex_dumpout_get` (void *x)

Sets 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)

Retrieves the Max wrapper object's dump outlet's outlet pointer.
- void * `max_jit_obex_adornment_get` (void *x, t_symbol *classname)

Sends a message and arguments out the dump outlet.
- void * `max_jit_obex_gimmeback` (void *x, t_symbol *s, long ac, t_atom *av)

Retrieves Max wrapper object adornment specified by class name.
- void `max_jit_obex_gimmeback_dumpout` (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.

36.77.1 Detailed Description

36.77.2 Function Documentation

36.77.2.1 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

<i>m</i>	method (function pointer)
<i>s</i>	method name

References A_CANT, A_GIMME, and addmess().

Referenced by `max_jit_classex_addattr()`.

Here is the call graph for this function:



36.77.2.2 void max_addmethod_defer_low (method m, char * s)

Adds method to Max class that calls defer_low rather than the method directly.

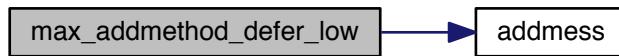
Parameters

<i>m</i>	method (function pointer)
<i>s</i>	method name

References A_CANT, A_GIMME, and addmess().

Referenced by max_jit_classex_addattr(), max_jit_classex_mop_wrap(), and max_jit_classex_standard_wrap().

Here is the call graph for this function:



36.77.2.3 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.

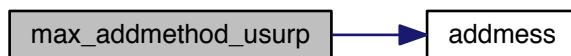
Parameters

<i>m</i>	method (function pointer)
<i>s</i>	method name

References A_CANT, A_GIMME, and addmess().

Referenced by max_jit_classex_addattr().

Here is the call graph for this function:



36.77.2.4 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

<i>m</i>	method (function pointer)
----------	---------------------------

<i>s</i>	method name
----------	-------------

References A_CANT, A_GIMME, and addmess().

Referenced by max_jit_classex_addattr(), and max_jit_classex_mop_wrap().

Here is the call graph for this function:



36.77.2.5 void max_jit_attr_args (void *x, short ac, t_atom *av)

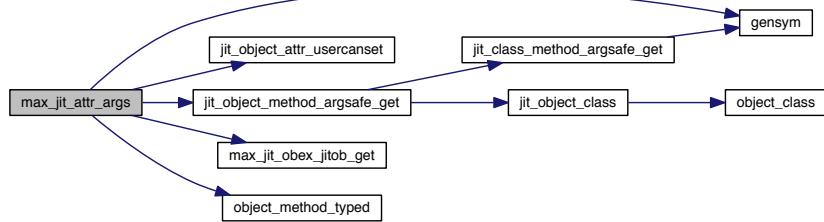
Processes attribute arguments.

Parameters

<i>x</i>	Max wrapper object pointer
<i>ac</i>	argument count
<i>av</i>	argument vector

References _jit_sym_nothing, A_SYM, gensym(), jit_object_attr_usercanset(), jit_object_method_argsafe_get(), max_jit_obex_jitob_get(), object_method_typed(), and t_symbol::s_name.

Here is the call graph for this function:



36.77.2.6 long max_jit_attr_args_offset (short ac, t_atom *av)

Determines argument offset to first attribute argument.

Parameters

<i>ac</i>	argument count
<i>av</i>	argument vector

Returns

argument offset

References attr_args_offset().

Referenced by max_jit_mop_matrix_args().

Here is the call graph for this function:



36.77.2.7 t_jit_err max_jit_attr_get(void *x, t_symbol *s, long *ac, t_atom **av)

Retrieves attribute value.

Parameters

<i>x</i>	Max wrapper object pointer
<i>s</i>	attribute name
<i>ac</i>	pointer atom count
<i>av</i>	pointer atom vector

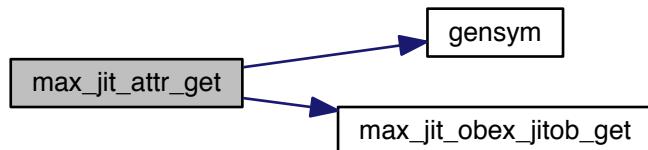
Returns

t_jit_err error code

References gensym(), max_jit_obex_jitob_get(), and t_symbol::s_name.

Referenced by max_jit_attr_getdump().

Here is the call graph for this function:



36.77.2.8 void max_jit_attr_getdump(void *x, t_symbol *s, short argc, t_atom *argv)

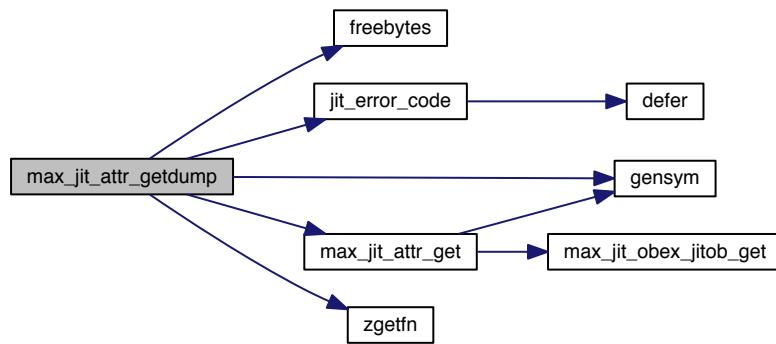
Retrieves attribute value and sends out dump outlet.

Parameters

<i>x</i>	Max wrapper object pointer
<i>s</i>	attribute name
<i>argc</i>	argument count (ignored)
<i>argv</i>	argument vector (ignored)

References freebytes(), gensym(), jit_error_code(), max_jit_attr_get(), t_symbol::s_name, and zgetfn().

Here is the call graph for this function:



36.77.2.9 void max_jit_attr_set (void * *x*, t_symbol * *s*, short *ac*, t_atom * *av*)

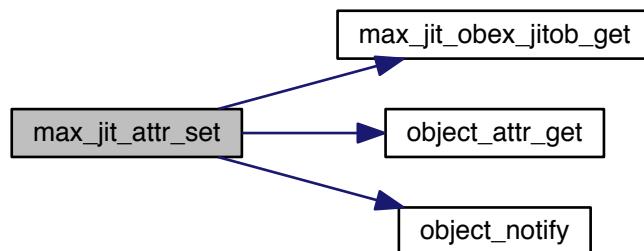
Sets attribute value.

Parameters

<i>x</i>	Max wrapper object pointer
<i>s</i>	attribute name
<i>ac</i>	atom count
<i>av</i>	atom vector

References max_jit_obex_jitob_get(), object_attr_get(), and object_notify().

Here is the call graph for this function:



36.77.2.10 long max_jit_classex_addattr(void * *x*, void * *attr*)

Adds an attribute to the Max wrapper class.

Parameters

<i>x</i>	pointer to t_max_jit_classex data (opaque)
<i>attr</i>	attribute object pointer

Returns

t_jit_err error code

Referenced by max_jit_classex_mop_wrap().

36.77.2.11 void * max_jit_classex_setup(long *obffset*)

Allocates and initializes special t_max_jit_classex data, used by the Max wrapper class.

Parameters

<i>obffset</i>	object struct byte offset to obex pointer
----------------	---

Returns

pointer to t_max_jit_classex data (opaque)

36.77.2.12 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

<i>mclass</i>	Max wrapper class pointer
<i>jclass</i>	jitter class pointer
<i>flags</i>	reserved for future use (currently ignored)

36.77.2.13 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

<i>x</i>	Max wrapper object pointer
<i>classname</i>	adornment classname

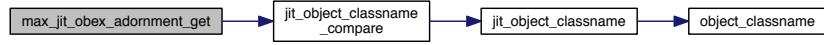
Returns

adornment pointer

References _jit_sym_findfirst, and jit_object_classname_compare().

Referenced by `max_jit_mop_adapt_matrix_all()`, `max_jit_mop_assist()`, `max_jit_mop_clear()`, `max_jit_mop_free()`, `max_jit_mop_get_io_by_name()`, `max_jit_mop_getinput()`, `max_jit_mop_getoutput()`, `max_jit_mop_getoutputmode()`, `max_jit_mop_inputs()`, `max_jit_mop jit_matrix()`, `max_jit_mop_matrix_args()`, `max_jit_mop_matrixout_new()`, `max_jit_mop_notify()`, `max_jit_mop_outputmatrix()`, `max_jit_mop_outputs()`, `max_jit_mop_variable_addinputs()`, and `max_jit_mop_variable_addoutputs()`.

Here is the call graph for this function:



36.77.2.14 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.

Parameters

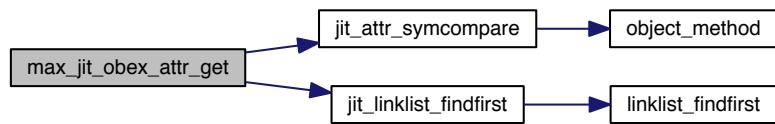
<code>x</code>	Max wrapper object pointer
<code>s</code>	attribute name
<code>ac</code>	pointer to atom count
<code>av</code>	pointer to atom vector

Returns

`t_jit_error` error code

References `_jit_sym_get`, `_jit_sym_getmethod`, `jit_attr_symcompare()`, and `jit_linklist_findfirst()`.

Here is the call graph for this function:



36.77.2.15 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

<code>x</code>	Max wrapper object pointer
<code>s</code>	attribute name

<i>ac</i>	atom count
<i>av</i>	atom vector

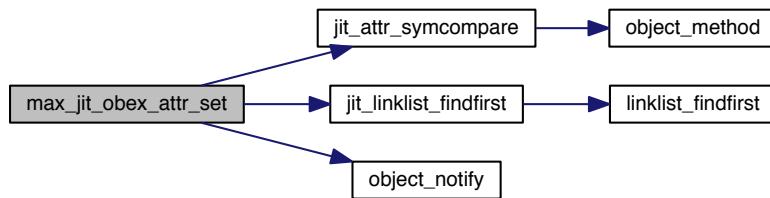
Returns

`t_jit_error` error code

References `_jit_sym_getmethod`, `_jit_sym_set`, `jit_attr_symcompare()`, `jit_linklist_findfirst()`, and `object_notify()`.

Referenced by `max_jit_classex_addattr()`.

Here is the call graph for this function:



36.77.2.16 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

<i>x</i>	Max wrapper object pointer
<i>s</i>	message symbol
<i>argc</i>	argument count
<i>argv</i>	argument vector

References `outlet_anything()`.

Referenced by `max_jit_classex_standard_wrap()`, and `max_jit_obex_gimmeback_dumpout()`.

Here is the call graph for this function:



36.77.2.17 void* max_jit_obex_dumpout_get(void * *x*)

Retrieves the Max wrapper object's dump outlet's outlet pointer.

Parameters

<i>x</i>	Max wrapper object pointer
----------	----------------------------

Returns

dump outlet pointer

36.77.2.18 void max_jit_obex_dumpout_set (void * *x*, void * *outlet*)

Sets the Max wrapper object's dump outlet's outlet pointer.

Parameters

<i>x</i>	Max wrapper object pointer
<i>outlet</i>	dump outlet pointer

Referenced by max_jit_mop_setup_simple(), and max_jit_obex_new().

36.77.2.19 void max_jit_obex_free (void * *x*)

Frees additional resources for the Max wrapper object instance.

Parameters

<i>x</i>	Max wrapper object pointer
----------	----------------------------

36.77.2.20 void max_jit_obex_gimmeback (void * *x*, t_symbol * *s*, long *ac*, t_atom * *av*)

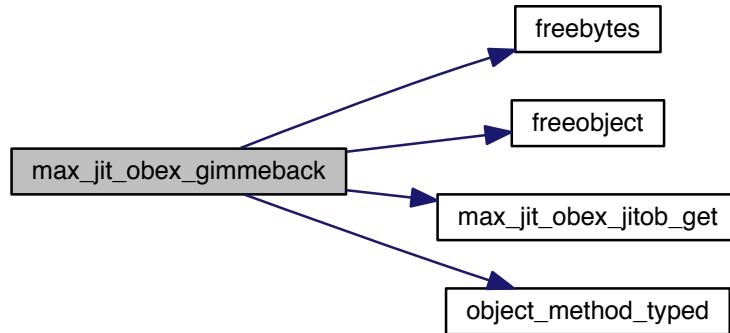
Calls gimmeback methods and frees any return value.

Parameters

<i>x</i>	Max wrapper object pointer
<i>s</i>	method name
<i>ac</i>	argument count
<i>av</i>	argument vector

References A NOTHING, A_OBJ, freebytes(), freeobject(), max_jit_obex_jitob_get(), object_method_typed(), and word::w_obj.

Here is the call graph for this function:



36.77.2.21 void max_jit_obex_gimmeback_dumpout (void * x, t_symbol * s, long ac, t_atom * av)

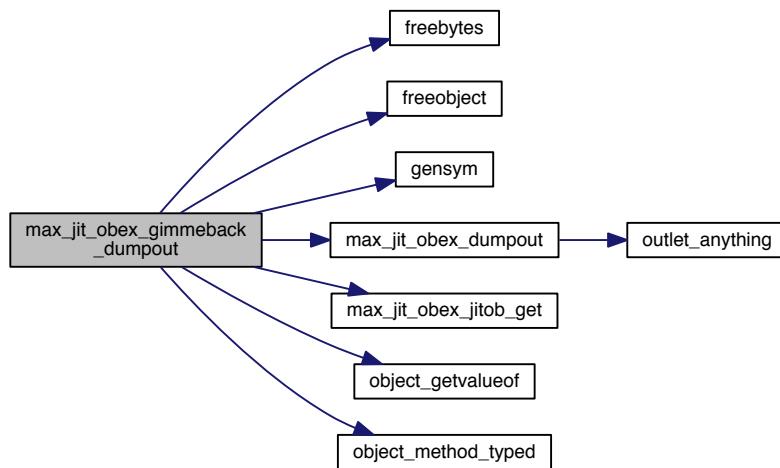
Calls gimmeback methods and outputs any return value through the Max wrapper class' dump outlet.

Parameters

x	Max wrapper object pointer
s	method name
ac	argument count
av	argument vector

References A NOTHING, A OBJ, freebytes(), freeobject(), gensym(), max_jit_obex_dumpout(), max_jit_obex jitob_get(), object_getvalueof(), object_method_typed(), t_symbol::s_name, and word::w_obj.

Here is the call graph for this function:



36.77.2.22 long max_jit_obex_inletnumber_get(void * *x*)

Retrieves the current inlet number used by inlet proxies.

Parameters

<i>x</i>	Max wrapper object pointer
----------	----------------------------

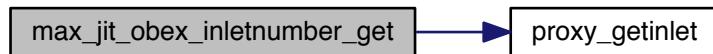
Returns

current inlet index

References proxy_getinlet().

Referenced by max_jit_mop_jit_matrix().

Here is the call graph for this function:



36.77.2.23 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

<i>x</i>	Max wrapper object pointer
<i>inletnumber</i>	inlet index

Referenced by max_jit_obex_new().

36.77.2.24 void* max_jit_obex_jitob_get(void * *x*)

Retrieves the wrapped Jitter object from a Max wrapper object.

Parameters

<i>x</i>	Max wrapper object pointer
----------	----------------------------

Returns

Jitter object pointer

Referenced by max_jit_attr_args(), max_jit_attr_get(), max_jit_attr_set(), max_jit_mop_jit_matrix(), max_jit_mop_setup(), max_jit_ob3d_detach(), max_jit_obex_gimmeback(), and max_jit_obex_gimmeback_dumpout().

36.77.2.25 void max_jit_obex_jitob_set(void * *x*, void * *jitob*)

Sets the wrapped Jitter object for a Max wrapper object.

Parameters

<i>x</i>	Max wrapper object pointer
<i>jitob</i>	Jitter object pointer

References gensym(), and OBJ_FLAG_REF.

Referenced by max_jit_mop_setup_simple(), and max_jit_obex_new().

Here is the call graph for this function:

**36.77.2.26 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

<i>mc</i>	Max class pointer
<i>classname</i>	Jitter class name to wrap

Returns

pointer to new Max wrapper object instance

36.77.2.27 t_jit_err max_jit_obex_proxy_new (void * *x*, long *c*)

Creates a new proxy inlet.

Parameters

<i>x</i>	Max wrapper object pointer
<i>c</i>	inlet index

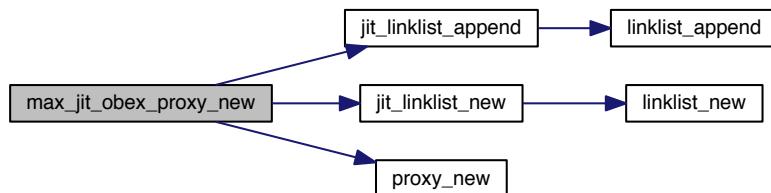
Returns

t_jit_err error code

References jit_linklist_append(), jit_linklist_new(), and proxy_new().

Referenced by max_jit_mop_inputs().

Here is the call graph for this function:



36.78 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.

36.78.1 Detailed Description

36.78.2 Function Documentation

36.78.2.1 `void jit_copy_bytes (void * dest, const void * src, long bytes)`

Copy bytes from source to destination pointer.

Parameters

<i>dest</i>	destination pointer
<i>src</i>	source pointer
<i>bytes</i>	byte count to copy

References `sysmem_copyptr()`.

Here is the call graph for this function:



36.78.2.2 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

<i>ptr</i>	pointer to memory
------------	-------------------

References `sysmem_freeptr()`.

Here is the call graph for this function:



36.78.2.3 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

<i>ptr</i>	pointer to memory
<i>size</i>	size in bytes allocated

References `sysmem_freeptr()`.

Referenced by `jit_glchunk_delete()`, `jit_matrix_freedata()`, `jit_ob3d_free()`, and `max_jit_obex_free()`.

Here is the call graph for this function:

**36.78.2.4 long jit_freemem(void)**

Reports free memory.

Warning

Obsolete. OS 9 only.

Returns

free bytes

36.78.2.5 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

<i>size</i>	size in bytes to allocate
-------------	---------------------------

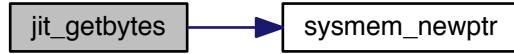
Returns

pointer to memory

References `sysmem_newptr()`.

Referenced by `jit_glchunk_copy()`, `jit_glchunk_grid_new()`, `jit_glchunk_new()`, `jit_matrix_getcell()`, `jit_ob3d_new()`, `jit_ob3d_setup()`, `max_jit_classex_setup()`, and `max_jit_obex_new()`.

Here is the call graph for this function:



36.78.2.6 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

<i>handle</i>	memory handle
---------------	---------------

References sysmem_freehandle().

Referenced by jit_matrix_free().

Here is the call graph for this function:



36.78.2.7 long jit_handle_lock(void ** handle, long lock)

Sets a memory handle's lock state.

Parameters

<i>handle</i>	memory handle
<i>lock</i>	state (1=locked, 0=unlocked)

Returns

lock state.

References sysmem_lockhandle().

Here is the call graph for this function:



36.78.2.8 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

References sysmem_newhandle().

Here is the call graph for this function:



36.78.2.9 long jit_handle_size_get (void ** handle)

Retrieves a memory handle's size in bytes.

Parameters

<code>handle</code>	memory handle
---------------------	---------------

Returns

size in bytes

References `sysmem_handlesize()`.

Here is the call graph for this function:

**36.78.2.10 t_jit_err jit_handle_size_set (void ** handle, long size)**

Sets a memory handle's size in bytes.

Parameters

<code>handle</code>	memory handle
<code>size</code>	new size in bytes

Returns

`t_jit_err` error code.

References `sysmem_resizehandle()`.

Here is the call graph for this function:

**36.78.2.11 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

<i>size</i>	size in bytes to allocate
-------------	---------------------------

Returns

pointer to memory

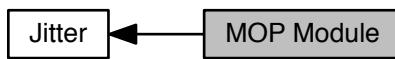
References `sysmem_newptr()`.

Here is the call graph for this function:



36.79 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.

36.79.1 Detailed Description

36.79.2 Function Documentation

36.79.2.1 [t_jit_err jit_mop_free \(t_jit_mop * x \)](#)

Frees instance of [t_jit_mop](#).

Parameters

x	t_jit_mop object pointer
---	--

Returns

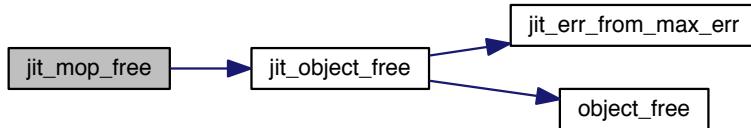
[t_jit_err](#) error code

Warning

Use `jit_object_free` instead.

References `t_jit_mop::inputlist`, `jit_object_free()`, and `t_jit_mop::outputlist`.

Here is the call graph for this function:

**36.79.2.2 void * jit_mop_getinput (t_jit_mop * x, long i)**

Retrieves input at input list index specified.

Parameters

<code>x</code>	<code>t_jit_mop</code> object pointer
<code>i</code>	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`.

References `_jit_sym_getindex`, and `t_jit_mop::inputlist`.

36.79.2.3 void * jit_mop_getinputlist (t_jit_mop * x)

Retrieves input list.

Parameters

<code>x</code>	<code>t_jit_mop</code> 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`.

References `t_jit_mop::inputlist`.

36.79.2.4 void * jit_mop_getoutput (t_jit_mop * x, long i)

Retrieves output at output list index specified.

Parameters

<i>x</i>	t_jit_mop object pointer
<i>i</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](#).

References [_jit_sym_getindex](#), and [t_jit_mop::outputlist](#).

36.79.2.5 void * jit_mop_getoutputlist (t_jit_mop * x)

Retrieves output list.

Parameters

<i>x</i>	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](#).

References [t_jit_mop::outputlist](#).

36.79.2.6 t_jit_err jit_mop_input_nolink (void * mop, long c)

Utility function to disable all linking attributes for a MOP input.

Parameters

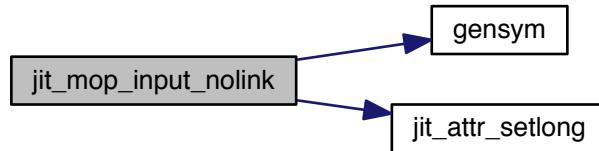
<i>mop</i>	t_jit_mop object pointer
<i>c</i>	input index

Returns

`t_jit_err` error code

References `_jit_sym_getindex`, `gensym()`, `t_jit_mop::inputlist`, and `jit_attr_setlong()`.

Here is the call graph for this function:

**36.79.2.7 `t_jit_err jit_mop_io_free (t_jit_mop * x)`**

Frees instance of `t_jit_mop_io`.

Parameters

<code>x</code>	<code>t_jit_mop_io</code> object pointer
----------------	--

Returns

`t_jit_err` error code

Warning

Use `jit_object_free` instead.

36.79.2.8 `method jit_mop_io_getioproc (t_jit_mop_io * x)`

Retrieves the I/O procedure used when handling incoming matrices.

Parameters

<code>x</code>	<code>t_jit_mop_io</code> 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`.

References `t_jit_mop_io::ioproc`.

36.79.2.9 `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](#).

References [t_jit_mop_io::matrix](#).

36.79.2.10 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](#).

References [t_jit_mop_io::ioproc](#).

36.79.2.11 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](#).

References [t_jit_mop_io::matrix](#).

36.79.2.12 `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`.

References `_jit_sym_char`, `_jit_sym_float32`, `_jit_sym_float64`, `_jit_sym_long`, `t_jit_mop_io::dimlink`, `gensym()`, `t_jit_mop_io::ioname`, `t_jit_mop_io::ioproc`, `JIT_MATRIX_MAX_DIMCOUNT`, `JIT_MATRIX_MAX_PLANECOUNT`, `T`, `t_jit_mop_io::matrix`, `t_jit_mop_io::matrixname`, `t_jit_mop_io::maxdim`, `t_jit_mop_io::maxdimcount`, `t_jit_mop_io::maxplanecount`, `t_jit_mop_io::mindim`, `t_jit_mop_io::mindimcount`, `t_jit_mop_io::minplanecount`, `t_jit_mop_io::panelink`, `t_jit_mop_io::special`, `t_jit_mop_io::typelink`, `t_jit_mop_io::types`, and `t_jit_mop_io::typescount`.

Here is the call graph for this function:

36.79.2.13 `t_jit_object * jit_mop_io_newcopy(t_jit_mop_io * x)`

Constructs instance of `t_jit_mop_io`, copying settings of input.

Parameters

<code>x</code>	<code>t_jit_mop_io</code> 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`.

36.79.2.14 `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

<i>x</i>	t_jit_mop_io object pointer
<i>info</i>	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](#).

References `t_jit_matrix_info::dim`, `t_jit_matrix_info::dimcount`, `JIT_MATRIX_MAX_DIMCOUNT`, `MAX`, `t_jit_mop_io::maxdim`, `t_jit_mop_io::maxdimcount`, `t_jit_mop_io::mindim`, and `t_jit_mop_io::mindimcount`.

36.79.2.15 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

<i>x</i>	t_jit_mop_io object pointer
<i>info</i>	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](#).

References `t_jit_mop_io::maxplaneCount`, `t_jit_mop_io::minplaneCount`, and `t_jit_matrix_info::planeCount`.

36.79.2.16 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, overwriting value in [t_jit_matrix_info](#) struct.

Parameters

<i>x</i>	t_jit_mop_io object pointer
<i>info</i>	t_jit_matrix_info pointer

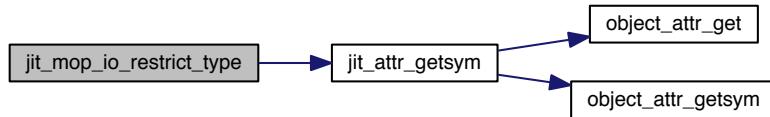
Returns[t_jit_err](#) error code

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`.

References `_jit_sym_type`, `jit_attr_getsym()`, `t_jit_mop_io::matrix`, `t_jit_matrix_info::type`, `t_jit_mop_io::types`, and `t_jit_mop_io::typescount`.

Here is the call graph for this function:



36.79.2.17 `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

<code>mop</code>	<code>t_jit_mop</code> object pointer
<code>mop_io</code>	<code>t_jit_mop_io</code> object pointer
<code>matrix</code>	<code>t_jit_matrix</code> object pointer

Returns

`t_jit_err` error code

```

1 void *m;
2 t_jit_matrix_info info;
3
4 if (matrix&&(m= jit_object_method(mop_io,_jit_sym_getmatrix))) {
5     jit_object_method(matrix,_jit_sym_getinfo,&info);
6     jit_object_method(mop_io,_jit_sym_restrict_type,&info);
7     jit_object_method(mop_io,_jit_sym_restrict_dim,&info);
8     jit_object_method(mop_io,_jit_sym_restrict_planecount,&info);
9     jit_object_method(m,_jit_sym_setinfo,&info);
10    jit_object_method(m,_jit_sym_frommatrix,matrix,NULL);
11 }
12
13 return JIT_ERR_NONE;
  
```

References `_jit_sym_frommatrix`, `_jit_sym_getinfo`, `_jit_sym_getmatrix`, `_jit_sym_restrict_dim`, `_jit_sym_restrict_planecount`, `_jit_sym_restrict_type`, and `_jit_sym_setinfo`.

Referenced by `jit_mop_ioproc_tosym()`.

36.79.2.18 `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

<i>mop</i>	t_jit_mop object pointer
<i>mop_io</i>	t_jit_mop_io object pointer
<i>matrix</i>	t_jit_matrix object pointer

Returns[t_jit_err](#) error code

```

1 void *m;
2 t_jit_matrix_info info;
3
4 if (matrix&&(m= jit_object_method(mop_io, _jit_sym_getmatrix))) {
5     jit_object_method(m, _jit_sym_frommatrix_trunc, matrix);
6 }
7
8 return JIT_ERR_NONE;

```

References [_jit_sym_frommatrix_trunc](#), and [_jit_sym_getmatrix](#).Referenced by [jit_mop_ioproc_tosym\(\)](#).**36.79.2.19 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

<i>mop</i>	t_jit_mop object pointer
<i>mop_io</i>	t_jit_mop_io object pointer
<i>matrix</i>	t_jit_matrix object pointer

Returns[t_jit_err](#) error code

```

1 void *m;
2 t_jit_matrix_info info;
3
4 if (matrix&&(m= jit_object_method(mop_io, _jit_sym_getmatrix))) {
5     jit_object_method(m, _jit_sym_clear);
6     jit_object_method(m, _jit_sym_frommatrix_trunc, matrix);
7 }
8
9 return JIT_ERR_NONE;

```

References [_jit_sym_clear](#), [_jit_sym_frommatrix_trunc](#), and [_jit_sym_getmatrix](#).Referenced by [jit_mop_ioproc_tosym\(\)](#).**36.79.2.20 t_symbol* jit_mop_ioproc_tosym(void * *ioproc*)**

Utility to convert MOP I/O procedure function to a human-readable type name.

Parameters

<i>ioproc</i>	t_jit_mop_io procedure pointer
---------------	--

Returns

[t_symbol](#) pointer

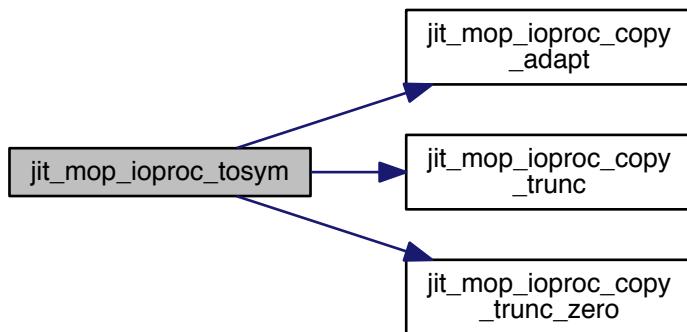
```

1 if (ioproc==NULL) {
2 return ps_resamp;
3 } else if (ioproc==jit_mop_ioproc_copy_adapt) {
4 return ps_adapt;
5 } else if (ioproc==jit_mop_ioproc_copy_trunc) {
6 return ps_trunc;
7 } else if (ioproc==jit_mop_ioproc_copy_trunc_zero) {
8 return ps_trunc_zero;
9 } else {
10 return ps_custom;
11 }
12 return ps_resamp;

```

References `jit_mop_ioproc_copy_adapt()`, `jit_mop_ioproc_copy_trunc()`, and `jit_mop_ioproc_copy_trunc_zero()`.

Here is the call graph for this function:



36.79.2.21 `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

<code>mop</code>	t_jit_mop object pointer
<code>s</code>	method symbol
<code>...</code>	untyped arguments

Returns

`t_jit_err` error code

References `_jit_sym_getindex`, `t_jit_mop::inputcount`, `t_jit_mop::inputlist`, `t_jit_mop::outputcount`, and `t_jit_mop::outputlist`.

36.79.2.22 `t_jit_object * jit_mop_new (long inputcount, long outputcount)`

Constructs instance of [t_jit_mop](#).

Returns

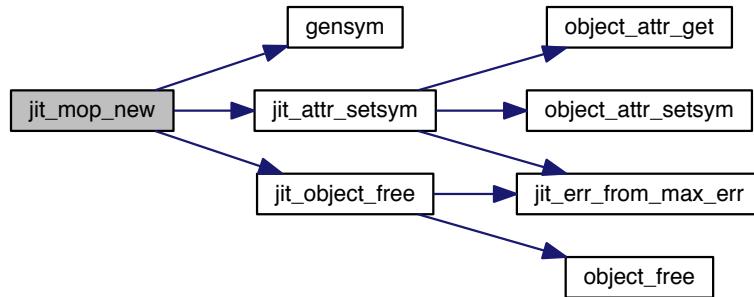
`t_jit_mop` object pointer

Warning

This function is not exported, but is provided for reference when calling via `jit_object_new`.

References `_jit_sym_append`, `t_jit_mop::adapt`, `t_jit_mop::caninplace`, `gensym()`, `t_jit_mop::inputcount`, `t_jit_mop::inputlist`, `jit_attr_setsym()`, `jit_object_free()`, `t_jit_mop::outputcount`, `t_jit_mop::outputlist`, `t_jit_mop::outputmode`, and `t_jit_mop::special`.

Here is the call graph for this function:

**36.79.2.23 `t_jit_object * jit_mop_newcopy (t_jit_mop * x)`**

Constructs instance of `t_jit_mop`, copying settings of input.

Parameters

<code>x</code>	<code>t_jit_mop</code> object pointer
----------------	---------------------------------------

Returns

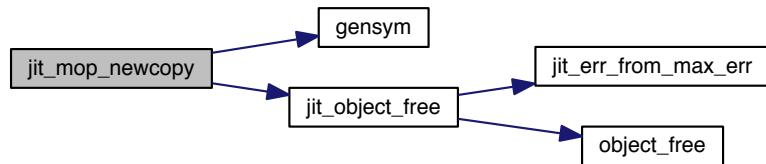
`t_jit_mop` object pointer

Warning

This function is not exported, but is provided for reference when calling via `jit_object_method` on an instance of `t_jit_mop`.

References `_jit_sym_append`, `_jit_sym_getindex`, `_jit_sym_newcopy`, `gensym()`, `t_jit_mop::inputcount`, `t_jit_mop::inputlist`, `jit_object_free()`, `t_jit_mop::outputcount`, and `t_jit_mop::outputlist`.

Here is the call graph for this function:



36.79.2.24 `t_jit_err jit_mop_output_nolink(void *mop, long c)`

Utility function to disable all linking attributes for a MOP output.

Parameters

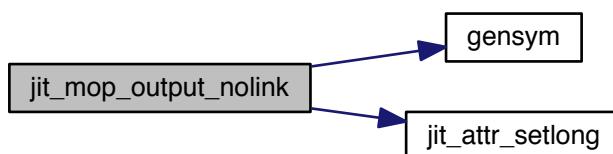
<code>mop</code>	<code>t_jit_mop</code> object pointer
<code>c</code>	output index

Returns

`t_jit_err` error code

References `_jit_sym_getindex`, `gensym()`, `jit_attr_setlong()`, and `t_jit_mop::outputlist`.

Here is the call graph for this function:



36.79.2.25 `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

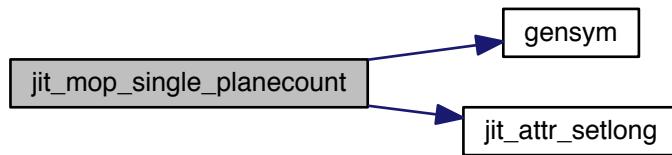
<i>mop</i>	t_jit_mop object pointer
<i>c</i>	plane count

Returns

[t_jit_err](#) error code

References [_jit_sym_getindex](#), [gensym\(\)](#), [t_jit_mop::inputcount](#), [t_jit_mop::inputlist](#), [jit_attr_setlong\(\)](#), [t_jit_mop::outputcount](#), and [t_jit_mop::outputlist](#).

Here is the call graph for this function:

36.79.2.26 [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

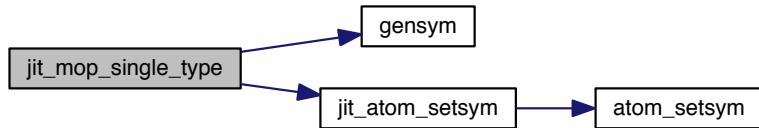
<i>mop</i>	t_jit_mop object pointer
<i>s</i>	type symbol

Returns

[t_jit_err](#) error code

References [_jit_sym_getindex](#), [gensym\(\)](#), [t_jit_mop::inputcount](#), [t_jit_mop::inputlist](#), [jit_atom_setsym\(\)](#), [t_jit_mop::outputcount](#), and [t_jit_mop::outputlist](#).

Here is the call graph for this function:



36.80 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.

36.80.1 Detailed Description

36.80.2 Function Documentation

36.80.2.1 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

<i>p</i>	parallel ndim calc data
----------	-------------------------

Referenced by `jit_parallel_ndim_simplecalc1()`, `jit_parallel_ndim_simplecalc2()`, `jit_parallel_ndim_simplecalc3()`, and `jit_parallel_ndim_simplecalc4()`.

36.80.2.2 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 calcuations to multiple threads if appropriate.

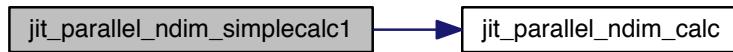
This function fills out the `t_jit_parallel_ndim` struct for a one input/output N-dimensional matrix calc method, and calls `jit_parallel_ndim_calc`. This function does not distinguish between what is an input or output.

Parameters

<i>fn</i>	N-dimensional matrix calc method
<i>data</i>	user defined pointer (typically object)
<i>dimcount</i>	master number of dimensions to iterate
<i>dim</i>	master pointer to dimension sizes
<i>planeCount</i>	master number of planes
<i>minfo1</i>	matrix info for first input/output
<i>bp1</i>	matrix data pointer for first input/output
<i>flags1</i>	parallel flags for first input/output

References `jit_parallel_ndim_calc()`.

Here is the call graph for this function:



36.80.2.3 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 calcuations 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.

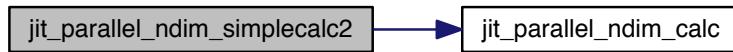
Parameters

<i>fn</i>	N-dimensional matrix calc method
<i>data</i>	user defined pointer (typically object)
<i>dimcount</i>	master number of dimensions to iterate
<i>dim</i>	master pointer to dimension sizes
<i>planeCount</i>	master number of planes

<i>minfo1</i>	matrix info for first input/output
<i>bp1</i>	matrix data pointer for first input/output
<i>flags1</i>	parallel flags for first input/output
<i>minfo2</i>	matrix info for second input/output
<i>bp2</i>	matrix data pointer for second input/output
<i>flags2</i>	parallel flags for second input/output

References `t_jit_matrix_info::flags`, and `jit_parallel_ndim_calc()`.

Here is the call graph for this function:



36.80.2.4 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.

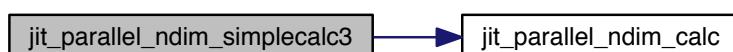
This function fills out the `t_jit_parallel_ndim` struct for a three input/output N-dimensional matrix calc method, and calls `jit_parallel_ndim_calc`. This function does not distinguish between what is an input or output.

Parameters

<i>fn</i>	N-dimensional matrix calc method
<i>data</i>	user defined pointer (typically object)
<i>dimcount</i>	master number of dimensions to iterate
<i>dim</i>	master pointer to dimension sizes
<i>planeCount</i>	master number of planes
<i>minfo1</i>	matrix info for first input/output
<i>bp1</i>	matrix data pointer for first input/output
<i>flags1</i>	parallel flags for first input/output
<i>minfo2</i>	matrix info for second input/output
<i>bp2</i>	matrix data pointer for second input/output
<i>flags2</i>	parallel flags for second input/output
<i>minfo3</i>	matrix info for third input/output
<i>bp3</i>	matrix data pointer for third input/output
<i>flags3</i>	parallel flags for third input/output

References `t_jit_matrix_info::flags`, and `jit_parallel_ndim_calc()`.

Here is the call graph for this function:



```
36.80.2.5 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

<i>fn</i>	N-dimensional matrix calc method
<i>data</i>	user defined pointer (typically object)
<i>dimcount</i>	master number of dimensions to iterate
<i>dim</i>	master pointer to dimension sizes
<i>planeCount</i>	master number of planes
<i>minfo1</i>	matrix info for first input/output
<i>bp1</i>	matrix data pointer for first input/output
<i>flags1</i>	parallel flags for first input/output
<i>minfo2</i>	matrix info for second input/output
<i>bp2</i>	matrix data pointer for second input/output
<i>flags2</i>	parallel flags for second input/output
<i>minfo3</i>	matrix info for third input/output
<i>bp3</i>	matrix data pointer for third input/output
<i>flags3</i>	parallel flags for third input/output
<i>minfo4</i>	matrix info for fourth input/output
<i>bp4</i>	matrix data pointer for fourth input/output
<i>flags4</i>	parallel flags for fourth input/output

References t_jit_matrix_info::flags, and jit_parallel_ndim_calc().

Here is the call graph for this function:



36.81 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)`

- `void * max_jit_mop_getinput (void *x, long c)`

Retrieves input `t_jit_mop_io` object pointer by name.
- `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.

36.81.1 Detailed Description

36.81.2 Function Documentation

36.81.2.1 `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

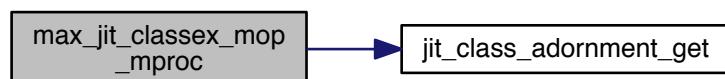
<code>mclass</code>	max jit classex pointer returned from <code>max_jit_classex_setup</code>
<code>jclass</code>	<code>t_jit_class</code> pointer, typically returned from <code>jit_class_findbyname</code>
<code>mproc</code>	matrix procedure

Returns

`t_jit_err` error code

References `_jit_sym_jit_mop`, `_jit_sym_special`, and `jit_class_adornment_get()`.

Here is the call graph for this function:



36.81.2.2 `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

<code>mclass</code>	max jit classex pointer returned from <code>max_jit_classex_setup</code>
<code>jclass</code>	<code>t_jit_class</code> pointer, typically returned from <code>jit_class_findbyname</code>

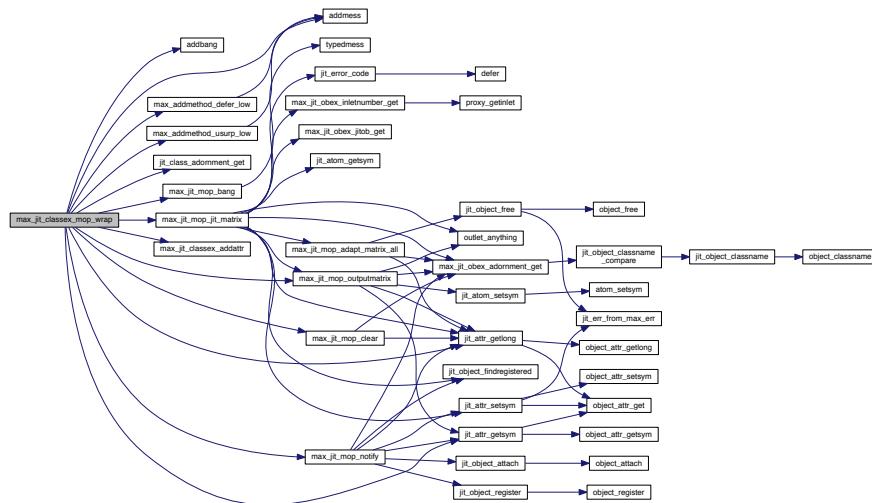
<i>flags</i>	flags to override default MOP Max wrapper behavior
--------------	--

Returns

`t_jit_err` error code

References `_jit_sym_atom`, `_jit_sym_char`, `_jit_sym_getinput`, `_jit_sym_getoutput`, `_jit_sym_inputcount`, `_jit_sym_ioname`, `_jit_sym_jit_attr_offset`, `_jit_sym_jit_attr_offset_array`, `_jit_sym_jit_mop`, `_jit_sym_long`, `_jit_sym_outputcount`, `_jit_sym_symbol`, `A_CANT`, `A_GIMME`, `addbang()`, `addmess()`, `JIT_ATTR_GET_DEFER_LOW`, `jit_attr_getlong()`, `jit_attr_getsym()`, `JIT_ATTR_SET_USURP_LOW`, `jit_class_adornment_get()`, `JIT_MATRIX_MAXDIMCOUNT`, `max_addmethod_defer_low()`, `max_addmethod_usurp_low()`, `max_jit_classex_addattr()`, `max_jit_mop_bang()`, `max_jit_mop_clear()`, `MAX_JIT_MOP_FLAGS_ONLY_MATRIX_PROBE`, `MAX_JIT_MOP_FLAGS_OWN_ADAPT`, `MAX_JIT_MOP_FLAGS_OWN_BANG`, `MAX_JIT_MOP_FLAGS_OWN_CLEAR`, `MAX_JIT_MOP_FLAGS_OWN_DIM`, `MAX_JIT_MOP_FLAGS_OWN_JIT_MATRIX`, `MAX_JIT_MOP_FLAGS_OWN_NAME`, `MAX_JIT_MOP_FLAGS_OWN_NOTIFY`, `MAX_JIT_MOP_FLAGS_OWN_OUTPUTMATRIX`, `MAX_JIT_MOP_FLAGS_OWN_OUTPUTMODE`, `MAX_JIT_MOP_FLAGS_OWN_PLANECOUNT`, `MAX_JIT_MOP_FLAGS_OWN_TYPE`, `max_jit_mop_jit_matrix()`, `max_jit_mop_notify()`, `max_jit_mop_outputmatrix()`, and `t_symbol::s_name`.

Here is the call graph for this function:

**36.81.2.3 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

<code>x</code>	Max object pointer
<code>y</code>	matrix to adapt to

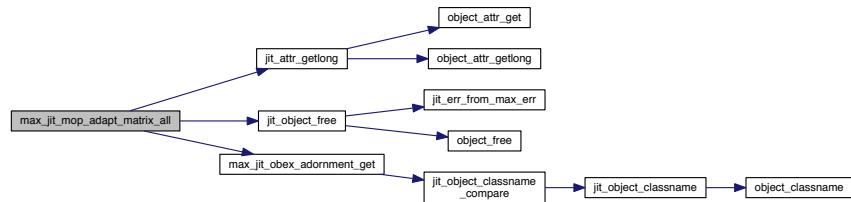
Returns

`t_jit_err` error code

References `_jit_sym_dimlink`, `_jit_sym_frommatrix`, `_jit_sym_getinfo`, `_jit_sym_getinput`, `_jit_sym_getioproc`, `_jit_sym_getmatrix`, `_jit_sym_getoutput`, `_jit_sym_inputcount`, `_jit_sym_jit_matrix`, `_jit_sym_jit_mop`, `_jit_sym_outputcount`, `_jit_sym_planelink`, `_jit_sym_setinfo`, `_jit_sym_typerlink`, `t_jit_matrix_info::dim`, `t_jit_matrix_info::dimcount`, `jit_attr_getlong()`, `jit_object_free()`, `max_jit_obex_adornment_get()`, `t_jit_matrix_info::planecount`, and `t_jit_matrix_info::type`.

Referenced by max_jit_mop_jit_matrix().

Here is the call graph for this function:



36.81.2.4 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

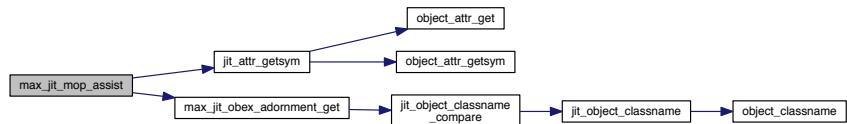
<i>x</i>	Max object pointer
<i>b</i>	ignored
<i>m</i>	inlet or outlet type
<i>a</i>	index
<i>s</i>	output string

Returns

t_jit_err error code

References _jit_sym_getinput, _jit_sym_getoutput, _jit_sym_ioname, _jit_sym_jit_mop, jit_attr_getsym(), and max_jit_obex_adornment_get().

Here is the call graph for this function:



36.81.2.5 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

<i>x</i>	Max object pointer
----------	--------------------

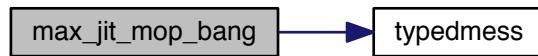
Returns

`t_jit_err` error code

References `_jit_sym_outputmatrix`, and `typedmess()`.

Referenced by `max_jit_classex_mop_wrap()`.

Here is the call graph for this function:

**36.81.2.6 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

<code>x</code>	Max object pointer
----------------	--------------------

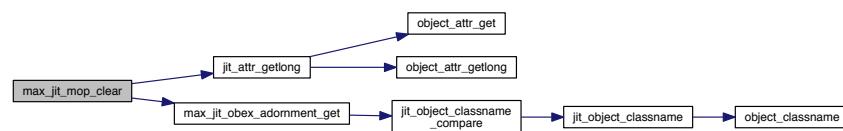
Returns

`t_jit_err` error code

References `_jit_sym_clear`, `_jit_sym_getinput`, `_jit_sym_getmatrix`, `_jit_sym_getoutput`, `_jit_sym_inputcount`, `_jit_sym_jit_mop`, `_jit_sym_outputcount`, `jit_attr_getlong()`, and `max_jit_obex_adornment_get()`.

Referenced by `max_jit_classex_mop_wrap()`.

Here is the call graph for this function:

**36.81.2.7 void max_jit_mop_free (void * x)**

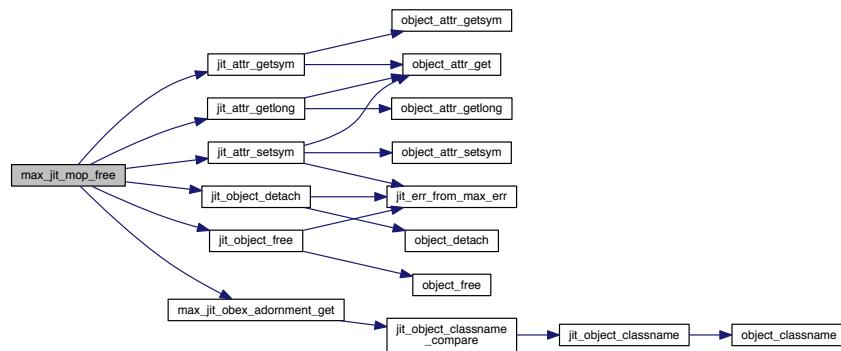
Frees additional resources used by a MOP Max wrapper object.

Parameters

x	Max object pointer
---	--------------------

References `_jit_sym_getinput`, `_jit_sym_getmatrix`, `_jit_sym_getoutput`, `_jit_sym_inputcount`, `_jit_sym_jit_mop`, `_jit_sym_matrix`, `_jit_sym_matrixname`, `_jit_sym_outputcount`, `jit_attr_getlong()`, `jit_attr_getsym()`, `jit_attr_setsym()`, `jit_object_detach()`, `jit_object_free()`, and `max_jit_obex_adornment_get()`.

Here is the call graph for this function:



36.81.2.8 void* max_jit_mop_get_io_by_name (void * x, t_symbol * s)

Retrieves `t_jit_mop_io` object pointer by name.

Parameters

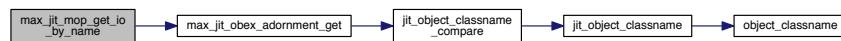
x	Max object pointer
s	input/output name (e.g. in, in2, out, out2, etc.)

Returns

`t_jit_err` error code

References `_jit_sym_getinput`, `_jit_sym_getoutput`, `_jit_sym_jit_mop`, `JIT_MOP_INPUT`, `JIT_MOP_OUTPUT`, and `max_jit_obex_adornment_get()`.

Here is the call graph for this function:



36.81.2.9 void* max_jit_mop_getinput (void * x, long c)

Retrieves input `t_jit_mop_io` object pointer index.

Parameters

<i>x</i>	Max object pointer
<i>c</i>	input index

Returns

`t_jit_err` error code

References `_jit_sym_getinput`, `_jit_sym_jit_mop`, and `max_jit_obex_adornment_get()`.

Here is the call graph for this function:

**36.81.2.10 void* max_jit_mop_getoutput (void * *x*, long *c*)**

Retrieves output `t_jit_mop_io` object pointer index.

Parameters

<i>x</i>	Max object pointer
<i>c</i>	output index

Returns

`t_jit_err` error code

References `_jit_sym_getoutput`, `_jit_sym_jit_mop`, and `max_jit_obex_adornment_get()`.

Here is the call graph for this function:

**36.81.2.11 long max_jit_mop_getoutputmode (void * *x*)**

Retrieves current MOP Max wrapper class output mode.

Parameters

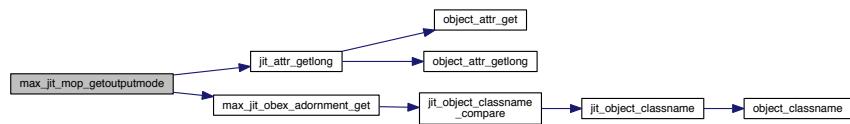
<i>x</i>	Max object pointer
----------	--------------------

Returns

`t_jit_err` error code

References `_jit_sym_jit_mop`, `_jit_sym_outputmode`, `jit_attr_getlong()`, and `max_jit_obex_adornment_get()`.

Here is the call graph for this function:



36.81.2.12 t_jit_err max_jit_mop_inputs (void * x)

Creates input resources for a MOP Max wrapper object.

Parameters

x	Max object pointer
---	--------------------

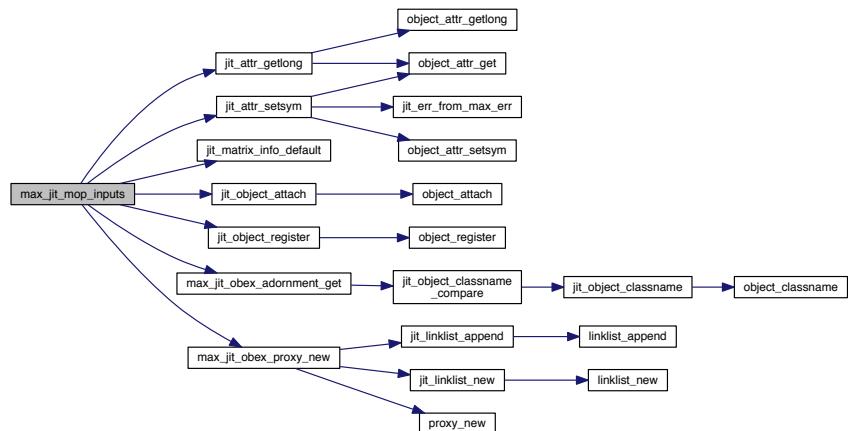
Returns

t_jit_err error code

References _jit_sym_getinput, _jit_sym_inputcount, _jit_sym_jit_matrix, _jit_sym_jit_mop, _jit_sym_matrix, _← jit_sym_matrixname, jit_attr_getlong(), jit_attr_setsym(), jit_matrix_info_default(), jit_object_attach(), jit_object← register(), max_jit_obex_adornment_get(), and max_jit_obex_proxy_new().

Referenced by max_jit_mop_setup_simple().

Here is the call graph for this function:



36.81.2.13 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

<i>x</i>	Max object pointer
<i>s</i>	message symbol ("jit_matrix")
<i>argc</i>	argument count
<i>argv</i>	argument vector

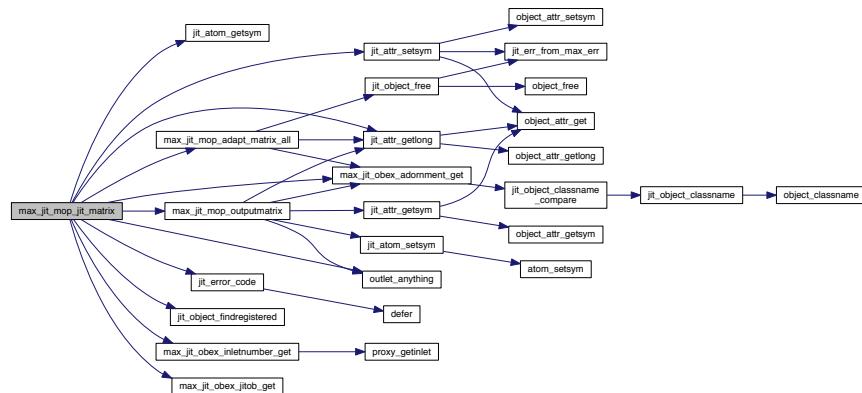
Returns

t_jit_err error code

References `_jit_sym_adapt`, `_jit_sym_class_jit_matrix`, `_jit_sym_frommatrix`, `_jit_sym_getinput`, `_jit_sym_getinputlist`, `_jit_sym_getioproc`, `_jit_sym_getmatrix`, `_jit_sym_getoutput`, `_jit_sym_getoutputlist`, `_jit_sym_jit_matrix`, `_jit_sym_jit_mop`, `_jit_sym_matrix`, `_jit_sym_matrix_calc`, `_jit_sym_matrixname`, `_jit_sym_nothing`, `_jit_sym_outputmode`, `jit_atom_getsym()`, `jit_attr_getlong()`, `jit_attr_setsym()`, `jit_error_code()`, `jit_object_findregistered()`, `max_jit_mop_adapt_matrix_all()`, `max_jit_mop_outputmatrix()`, `max_jit_obex_adornment_get()`, `max_jit_obex_inletnumber_get()`, `max_jit_obex_jitob_get()`, and `outlet_anything()`.

Referenced by max_jit_classex_mop_wrap().

Here is the call graph for this function:



36.81.2.14 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

<i>x</i>	Max object pointer
<i>argc</i>	argument count
<i>argv</i>	argument vector

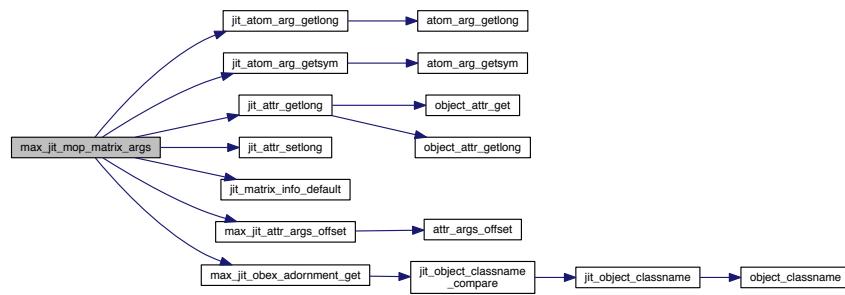
Returns

t jit err error code

References `_jit_sym_adapt`, `_jit_sym_dimlink`, `_jit_sym_getinfo`, `_jit_sym_getinput`, `_jit_sym_getmatrix`, `_jit_sym_getoutput`, `_jit_sym_inputcount`, `_jit_sym_jit_mop`, `_jit_sym_outputcount`, `_jit_sym_outputmode`, `_jit_sym_planelink`, `_jit_sym_setinfo`, `_jit_sym_typelink`, `t_jit_matrix_info::dim`, `t_jit_matrix_info::dimcount`, `jit_atom_arg_getlong()`, `jit_atom_arg_getsym()`, `jit_attr_getlong()`, `jit_attr_setlong()`, `jit_matrix_info_default()`, `max_jit_attr_args_offset()`, `max_jit_obex_adornment_get()`, `t_jit_matrix_info::planecount`, and `t_jit_matrix_info::type`.

Referenced by max_jit_mop_setup_simple().

Here is the call graph for this function:



36.81.2.15 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)

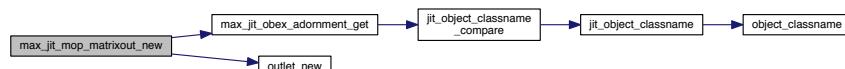
Returns

t_jit_err error code

References `_jit_sym_getoutput`, `_jit_sym_jit_mop`, `_jit_sym_special`, `max_jit_obex_adornment_get()`, and `outlet_new()`.

Referenced by `max_jit_mop_outputs()`.

Here is the call graph for this function:



36.81.2.16 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

<i>msg</i>	notification message
------------	----------------------

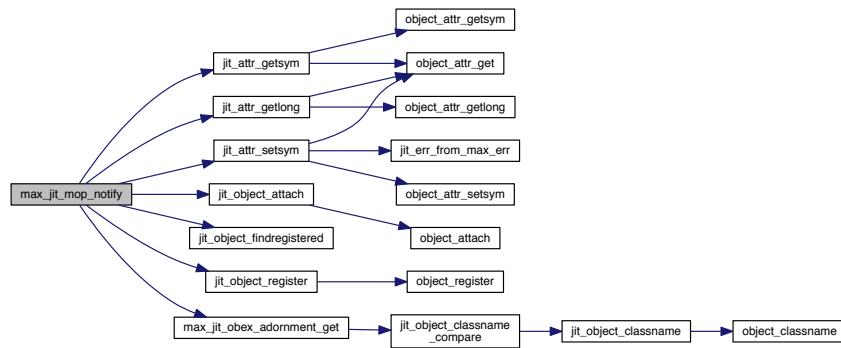
Returns

t_jit_err error code

References *_jit_sym_free*, *_jit_sym_getinfo*, *_jit_sym_getinput*, *_jit_sym_getoutput*, *_jit_sym_inputcount*, *_jit_sym_jit_matrix*, *_jit_sym_jit_mop*, *_jit_sym_matrixname*, *_jit_sym_outputcount*, *jit_attr_getlong()*, *jit_attr_getsym()*, *jit_attr_setsym()*, *jit_object_attach()*, *jit_object_findregistered()*, *jit_object_register()*, and *max_jit_obex_adornment_get()*.

Referenced by *max_jit_classex_mop_wrap()*.

Here is the call graph for this function:

**36.81.2.17 *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

<i>x</i>	Max object pointer
----------	--------------------

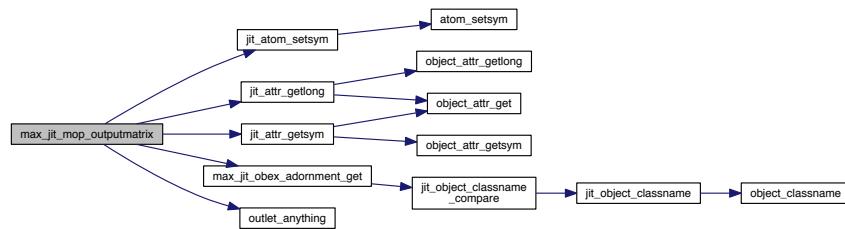
Returns

`t_jit_err` error code

References `_jit_sym_getoutput`, `_jit_sym_jit_matrix`, `_jit_sym_jit_mop`, `_jit_sym_matrixname`, `_jit_sym_outputcount`, `_jit_sym_outputmode`, `jit_atom_setsym()`, `jit_attr_getlong()`, `jit_attr_getsym()`, `max_jit_obex_adornment_get()`, and `outlet_anything()`.

Referenced by `max_jit_classex_mop_wrap()`, and `max_jit_mop_jit_matrix()`.

Here is the call graph for this function:

**36.81.2.18 t_jit_err max_jit_mop_outputs (void * x)**

Creates output resources for a MOP Max wrapper object.

Parameters

x	Max object pointer
---	--------------------

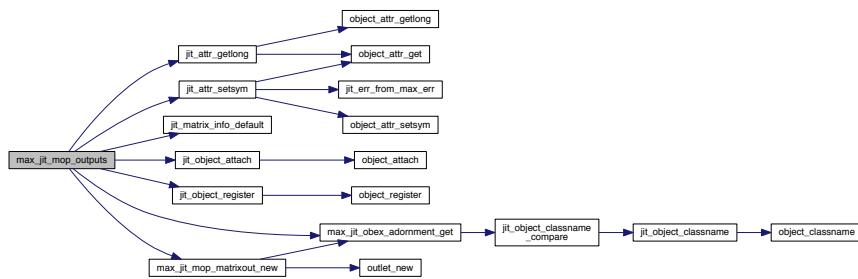
Returns

`t_jit_err` error code

References `_jit_sym_getoutput`, `_jit_sym_jit_matrix`, `_jit_sym_jit_mop`, `_jit_sym_matrix`, `_jit_sym_matrixname`, `_jit_sym_outputcount`, `jit_attr_getlong()`, `jit_attr_setsym()`, `jit_matrix_info_default()`, `jit_object_attach()`, `jit_object_register()`, `max_jit_mop_matrixout_new()`, and `max_jit_obex_adornment_get()`.

Referenced by `max_jit_mop_setup_simple()`.

Here is the call graph for this function:

**36.81.2.19 t_jit_err max_jit_mop_setup (void * x)**

Sets up necessary resources for MOP Max wrapper object.

Parameters

x	Max object pointer
---	--------------------

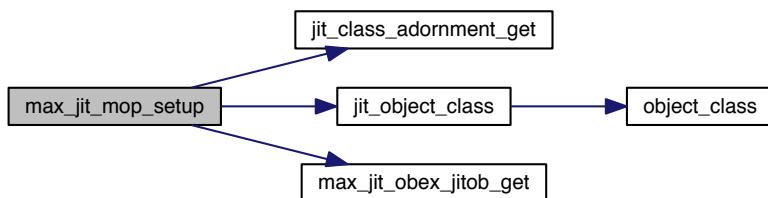
Returns

t_jit_err error code

References `_jit_sym_jit_mop`, `_jit_sym_newcopy`, `jit_class_adornment_get()`, `jit_object_class()`, and `max_jit_obex_jitob_get()`.

Referenced by `max_jit_mop_setup_simple()`.

Here is the call graph for this function:



36.81.2.20 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

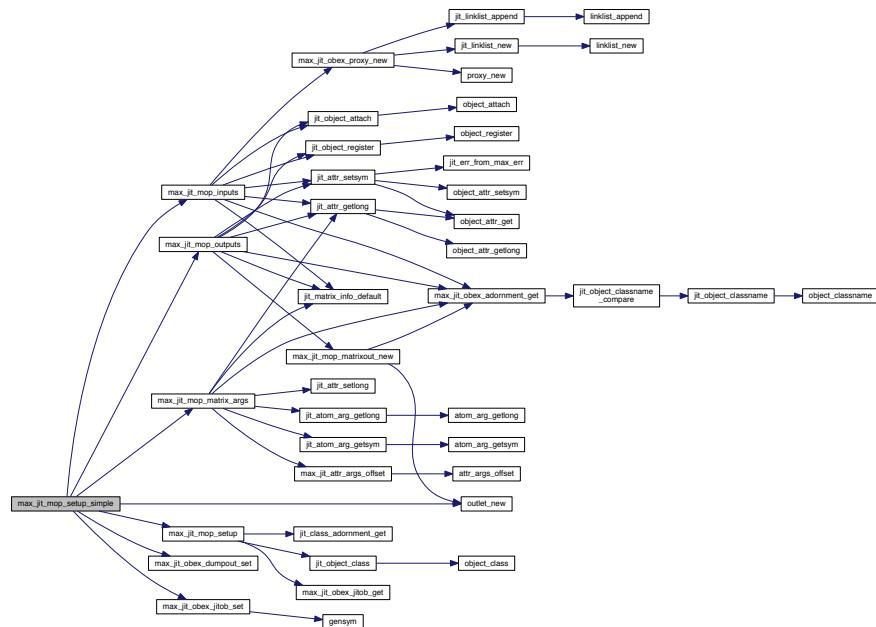
```

1 max_jit_obex_jitob_set(x,o);
2 max_jit_obex_dumpout_set(x,outlet_new(x,NULL));
3 max_jit_mop_setup(x);
4 max_jit_mop_inputs(x);
5 max_jit_mop_outputs(x);
6 max_jit_mop_matrix_args(x,argc,argv);
7
8 return JIT_ERR_NONE;

```

References `max_jit_mop_inputs()`, `max_jit_mop_matrix_args()`, `max_jit_mop_outputs()`, `max_jit_mop_setup()`, `max_jit_obex_dumpout_set()`, `max_jit_obex_jitob_set()`, and `outlet_new()`.

Here is the call graph for this function:



36.81.2.21 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

References `_jit_sym_jit_mop`, `gensym()`, and `max jit_obex_adornment_get()`.

Here is the call graph for this function:



36.81.2.22 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

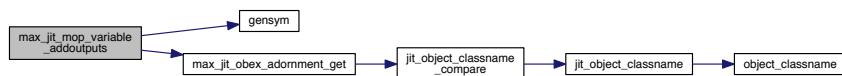
<i>x</i>	Max object pointer
<i>c</i>	inlet count

Returns

`t_jit_err` error code

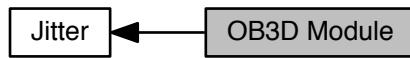
References `_jit_sym_jit_mop`, `gensym()`, and `max_jit_obex_adornment_get()`.

Here is the call graph for this function:



36.82 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.
- const char * [jit_gl_get_glsL_version](#) ()

Retrieves OpenGL GLSL version 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)

- 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.
- void * **ob3d_patcher_get** (void *v)
Retrieves containing patcher 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_class_ob3d_wrap** (**t_class** *c)
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.*

36.82.1 Detailed Description

36.82.2 Function Documentation

36.82.2.1 void jit_gl_begincapture (**t_jit_gl_drawinfo** * drawinfo, **t_symbol** * s, long i)

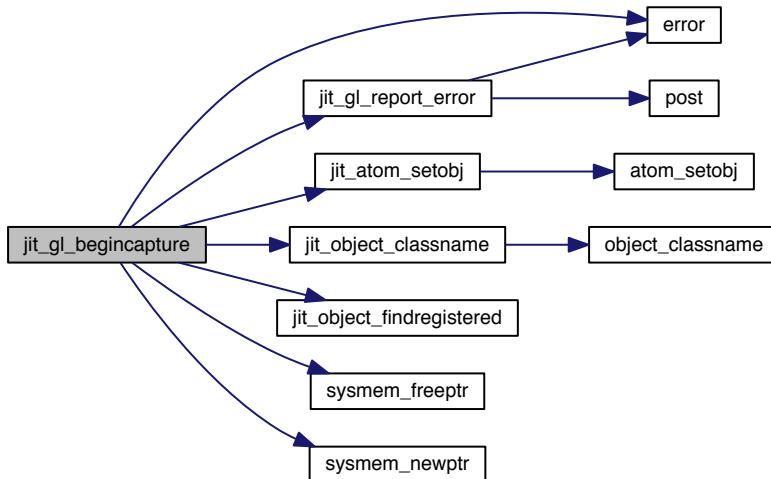
Begin texture capture.

Parameters

<i>drawinfo</i>	t_jit_gl_drawinfo pointer
<i>s</i>	texture name
<i>i</i>	ignored

References error(), jit_atom_setobj(), jit_gl_report_error(), jit_object_classname(), jit_object_findregistered(), **t_jit_gl_drawinfo::ob3d**, sysmem_freeptr(), and sysmem_newptr().

Here is the call graph for this function:



36.82.2.2 void jit_gl_bindtexture (*t_jit_gl_drawinfo* * *drawinfo*, *t_symbol* * *s*, long *i*)

Bind texture for specified texture unit.

Parameters

<i>drawinfo</i>	t_jit_gl_drawinfo pointer
<i>s</i>	texture name
<i>i</i>	texture unit

References [t_jit_gl_drawinfo::ob3d](#).

36.82.2.3 long jit_gl_drawinfo_active_textures ([t_jit_gl_drawinfo](#) * *drawinfo*)

Determine the number of active texture units to use.

Parameters

<i>drawinfo</i>	t_jit_gl_drawinfo pointer
-----------------	---

Returns

number of active texture units

References [t_jit_gl_drawinfo::ctx](#), and [t_jit_gl_drawinfo::ob3d](#).

36.82.2.4 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

<i>x</i>	Jitter object pointer
<i>drawinfo</i>	t_jit_gl_drawinfo pointer

Returns

[t_jit_err](#) error code

References [t_jit_gl_drawinfo::ctx](#), and [t_jit_gl_drawinfo::ob3d](#).

36.82.2.5 void jit_gl_endcapture ([t_jit_gl_drawinfo](#) * *drawinfo*, [t_symbol](#) * *s*, long *i*)

End texture capture.

Parameters

<i>drawinfo</i>	t_jit_gl_drawinfo pointer
<i>s</i>	texture name
<i>i</i>	ignored

References [t_jit_gl_drawinfo::ob3d](#).

36.82.2.6 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

36.82.2.7 const char* jit_gl_get_glsl_version()

Retrieves OpenGL GLSL version string.

Equivalent to glGetString(GL_SHADING_LANGUAGE_VERSION). Assumes a valid context has been set and the OpenGL renderer supports GLSL.

Returns

OpenGL GL GLSL version string

36.82.2.8 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

36.82.2.9 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

36.82.2.10 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

36.82.2.11 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

Referenced by jit_gl_is_min_version().

36.82.2.12 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

<i>ctx</i>	t_jit_gl_context pointer
<i>ext</i>	extension string

Returns

1 if true, 0 if false.

36.82.2.13 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

<i>major</i>	major version number
<i>minor</i>	minor version number
<i>release</i>	release version number

Returns

1 if true, 0 if false.

References jit_gl_get_version().

Here is the call graph for this function:

**36.82.2.14 long jit_gl_report_error (char * *prefix*)**

Tests for OpenGL error and reports to Max window.

Parameters

<i>prefix</i>	prefix string
---------------	---------------

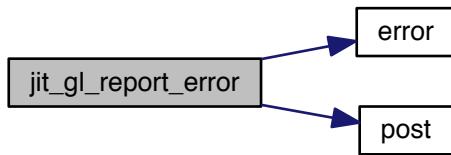
Returns

OpenGL error code

References error(), and post().

Referenced by jit_gl_begincapture().

Here is the call graph for this function:



36.82.2.15 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

<i>drawinfo</i>	t_jit_gl_drawinfo pointer
<i>s</i>	s texture coordinate

References *t_jit_gl_drawinfo*::ctx, MIN, and *t_jit_gl_drawinfo*::ob3d.

36.82.2.16 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

<i>drawinfo</i>	t_jit_gl_drawinfo pointer
<i>v</i>	texture coordinate vector

References *t_jit_gl_drawinfo*::ctx, MIN, and *t_jit_gl_drawinfo*::ob3d.

36.82.2.17 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

<i>drawinfo</i>	t_jit_gl_drawinfo pointer
<i>s</i>	s texture coordinate
<i>t</i>	t texture coordinate

References *t_jit_gl_drawinfo*::ctx, MAX, MIN, and *t_jit_gl_drawinfo*::ob3d.

36.82.2.18 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

<i>drawinfo</i>	t_jit_gl_drawinfo pointer
<i>v</i>	texture coordinate vector

References [t_jit_gl_drawinfo::ctx](#), [MIN](#), and [t_jit_gl_drawinfo::ob3d](#).

36.82.2.19 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

<i>drawinfo</i>	t_jit_gl_drawinfo pointer
<i>s</i>	<i>s</i> texture coordinate
<i>t</i>	<i>t</i> texture coordinate
<i>r</i>	<i>r</i> texture coordinate

References [t_jit_gl_drawinfo::ctx](#), [MIN](#), and [t_jit_gl_drawinfo::ob3d](#).

36.82.2.20 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

<i>drawinfo</i>	t_jit_gl_drawinfo pointer
<i>v</i>	texture coordinate vector

References [t_jit_gl_drawinfo::ctx](#), [MIN](#), and [t_jit_gl_drawinfo::ob3d](#).

36.82.2.21 void jit_gl_unbindtexture ([t_jit_gl_drawinfo](#) * *drawinfo*, [t_symbol](#) * *s*, long *i*)

Unbind texture for specified texture unit.

Parameters

<i>drawinfo</i>	t_jit_gl_drawinfo pointer
<i>s</i>	texture name
<i>i</i>	texture unit

References [t_jit_gl_drawinfo::ob3d](#).

36.82.2.22 [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

<i>new</i>	pointer to new t_jit_glchunk pointer
<i>orig</i>	priginal t_jit_glchunk pointer

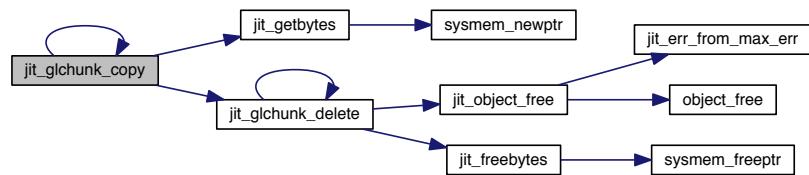
Returns

`t_jit_err` error code

References `_jit_sym_frommatrix`, `_jit_sym_getinfo`, `_jit_sym_jit_matrix`, `_jit_sym_nothing`, `jit_getbytes()`, `jit->glchunk_copy()`, `jit_glchunk_delete()`, `t_jit_glchunk::m_flags`, `t_jit_glchunk::m_index`, `t_jit_glchunk::m_index_name`, `t_jit_glchunk::m_vertex`, `t_jit_glchunk::m_vertex_name`, `t_jit_glchunk::next_chunk`, and `t_jit_glchunk::prim`.

Referenced by `jit_glchunk_copy()`, and `jit_ob3d_draw_chunk()`.

Here is the call graph for this function:



36.82.2.23 void jit_glchunk_delete (`t_jit_glchunk` * *x*)

Disposes `t_jit_glchunk` struct.

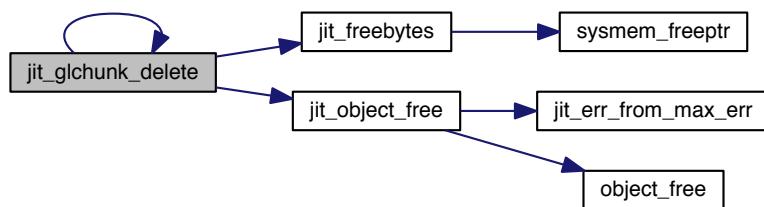
Parameters

<i>x</i>	<code>t_jit_glchunk</code> pointer
----------	------------------------------------

References `jit_freebytes()`, `jit_glchunk_delete()`, `jit_object_free()`, `t_jit_glchunk::m_index`, `t_jit_glchunk::m_vertex`, and `t_jit_glchunk::next_chunk`.

Referenced by `jit_glchunk_copy()`, `jit_glchunk_delete()`, `jit_glchunk_grid_new()`, `jit_glchunk_new()`, `jit_ob3d_draw->-chunk()`, and `jit_ob3d_free()`.

Here is the call graph for this function:



36.82.2.24 `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

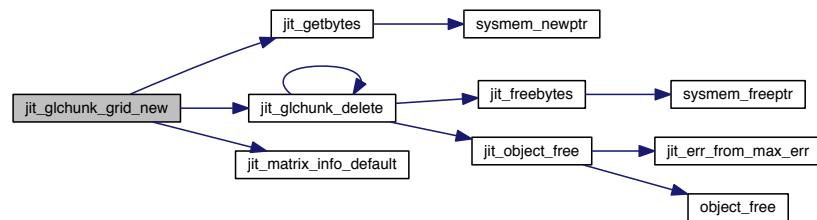
<i>prim</i>	drawing primitive name
<i>planes</i>	number of planes to allocate in vertex matrix
<i>width</i>	width of vertex matrix to allocate
<i>height</i>	height of vertex matrix to allocate

Returns

[t_jit_glchunk](#) pointer

References `_jit_sym_float32`, `_jit_sym_jit_matrix`, `_jit_sym_nothing`, `t_jit_matrix_info::dim`, `t_jit_matrix_info::dimcount`, `jit_getbytes()`, `jit_glchunk_delete()`, `jit_matrix_info_default()`, `t_jit_glchunk::m_flags`, `t_jit_glchunk::m_index`, `t_jit_glchunk::m_index_name`, `t_jit_glchunk::m_vertex`, `t_jit_glchunk::m_vertex_name`, `t_jit_glchunk::next_chunk`, `t_jit_matrix_info::planecount`, `t_jit_glchunk::prim`, and `t_jit_matrix_info::type`.

Here is the call graph for this function:



36.82.2.25 `t_jit_glchunk* jit_glchunk_new (t_symbol * prim, int planes, int vertices, int indices)`

Allocates and initializes a [t_jit_glchunk](#) struct.

Parameters

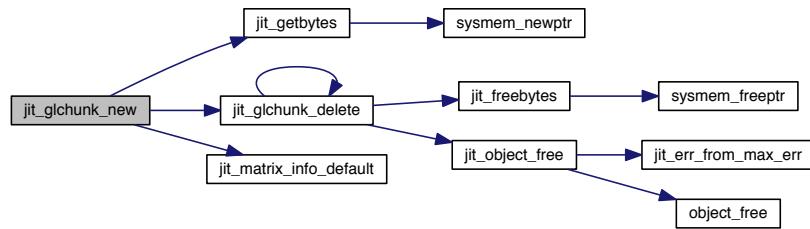
<i>prim</i>	drawing primitive name
<i>planes</i>	number of planes to allocate in vertex matrix
<i>vertices</i>	number of vertices to allocate in vertex matrix
<i>indices</i>	number of indices to allocate in index matrix, if used

Returns

`t_jit_glchunk` pointer

References `_jit_sym_float32`, `_jit_sym_jit_matrix`, `_jit_sym_long`, `_jit_sym_nothing`, `t_jit_matrix_info::dim`, `t_jit_matrix_info::dimcount`, `jit_getbytes()`, `jit_glchunk_delete()`, `jit_matrix_info_default()`, `t_jit_glchunk::m_flags`, `t_jit_glchunk::m_index`, `t_jit_glchunk::m_index_name`, `t_jit_glchunk::m_vertex`, `t_jit_glchunk::m_vertex_name`, `t_jit_glchunk::next_chunk`, `t_jit_matrix_info::planecount`, `t_jit_glchunk::prim`, and `t_jit_matrix_info::type`.

Here is the call graph for this function:

36.82.2.26 `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

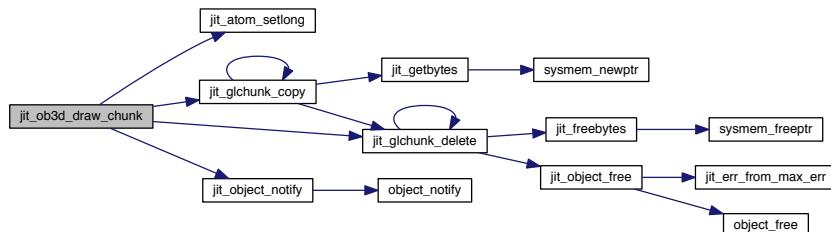
<code>v</code>	<code>t_jit_ob3d</code> pointer
<code>chunk</code>	<code>t_jit_glchunk</code> pointer

Returns

`t_jit_err` error code

References `_jit_sym_nothing`, `_jit_sym_register`, `jit_atom_setlong()`, `jit_glchunk_copy()`, `jit_glchunk_delete()`, and `jit_object_notify()`.

Here is the call graph for this function:

36.82.2.27 `void jit_ob3d_free(void * jit_ob)`

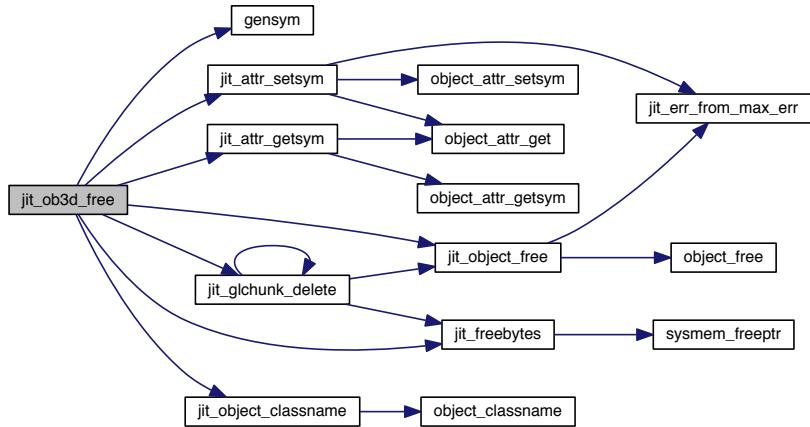
Disposes OB3D resources.

Parameters

<i>jit_obj</i>	Jitter object pointer
----------------	-----------------------

References `_jit_sym_nothing`, `gensym()`, `jit_attr_getsym()`, `jit_attr_setsym()`, `jit_freebytes()`, `jit_glchunk_delete()`, `jit_object_classname()`, and `jit_object_free()`.

Here is the call graph for this function:



36.82.2.28 void* jit_obj3d_new (void * *x*, t_symbol * *dest_name*)

Allocates and initializes OB3D resources.

Parameters

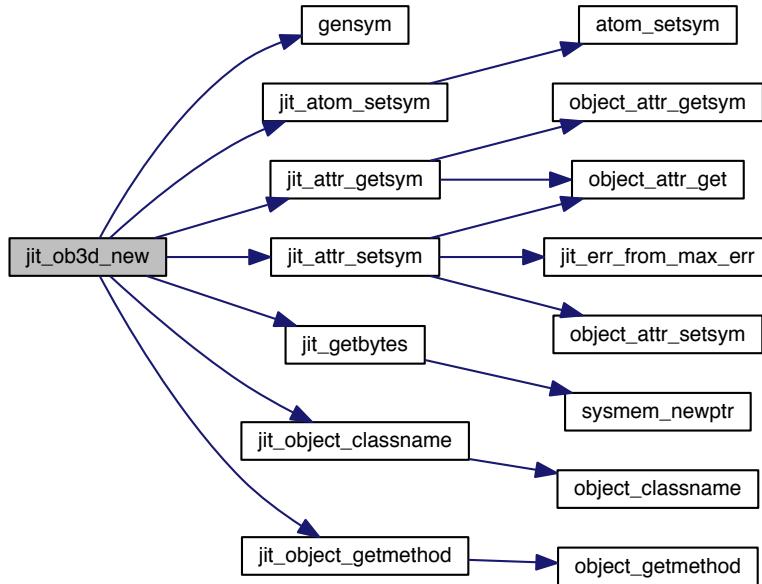
<i>x</i>	Jitter object pointer
<i>dest_name</i>	drawing destination name

Returns

`t_jit_obj3d` pointer (opaque)

References `_jit_sym_name`, `_jit_sym_nothing`, `gensym()`, `jit_atom_setsym()`, `jit_attr_getsym()`, `jit_attr_setsym()`, `jit_getbytes()`, `jit_object_classname()`, and `jit_object_getmethod()`.

Here is the call graph for this function:



36.82.2.29 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

<i>jit_ob</i>	Jitter object pointer
---------------	-----------------------

Returns

t_jit_err error code

36.82.2.30 void* jit_ob3d_setup(void * *jit_class*, long *oboffset*, long *flags*)

Adds default methods and attributes to the OB3D class.

Parameters

<i>jit_class</i>	Jitter class pointer
------------------	----------------------

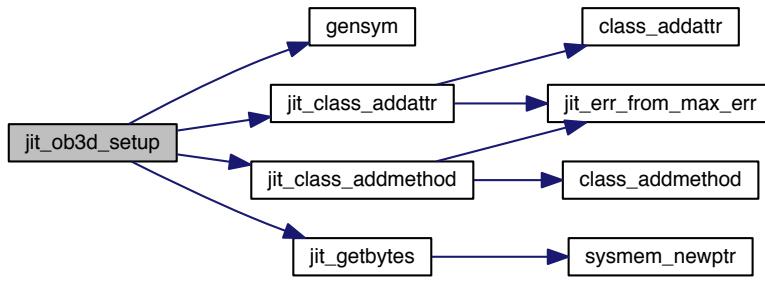
<i>oboffset</i>	object struct byte offset for t_jit_ob3d pointer
<i>flags</i>	flags to override default OB3D behavior

Returns

t_jit_class3d pointer (opaque)

References _jit_sym_atom, _jit_sym_float32, _jit_sym_jit_attr_offset, _jit_sym_jit_attr_offset_array, _jit_sym_long, _jit_sym_symbol, A_CANT, A_DEFER_LOW, calcoffset, CLASS_ATTR_BASIC, CLASS_ATTR_ENUM, CLAS←S_METHOD_ATTR_PARSE, gensym(), JIT_ATTR_GET_DEFER_LOW, JIT_ATTR_GET_OPAQUE_USER, JI←T_ATTR_SET_OPAQUE_USER, JIT_ATTR_SET_USURP_LOW, jit_class_addattr(), jit_class_addmethod(), jit←_getbytes(), JIT_OB3D_AUTO_ONLY, JIT_OB3D_HAS_CAMERA, JIT_OB3D_HAS_LIGHTS, JIT_OB3D_IS_R←ENDERER, JIT_OB3D_IS_SLAB, JIT_OB3D_NO_ANTIALIAS, JIT_OB3D_NO_BLEND, JIT_OB3D_NO_COLOR, JIT_OB3D_NO_DEPTH, JIT_OB3D_NO_FOG, JIT_OB3D_NO_LIGHTING_MATERIAL, JIT_OB3D_NO_MATTR←IXOUTPUT, JIT_OB3D_NO_POLY_VARS, JIT_OB3D_NO_ROTATION_SCALE, and JIT_OB3D_NO_TEXTURE.

Here is the call graph for this function:



36.82.2.31 t_jit_err max_jit_ob3d_assist (void * x, void * b, long m, long a, char * s)

Default OB3D Max wrapper assistance method.

Parameters

<i>x</i>	Max object pointer
<i>b</i>	ignored
<i>m</i>	inlet or outlet type
<i>a</i>	index
<i>s</i>	output string

Returns

t_jit_err error code

36.82.2.32 void max_jit_ob3d_attach (void * x, t_jit_object * jit_ob, void * outlet)

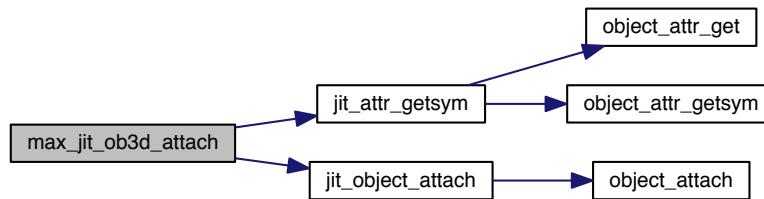
Allocates and initializes OB3D Max wrapper related resources.

Parameters

<i>x</i>	Max wrapper object pointer
<i>jit_ob</i>	Jitter object pointer
<i>outlet</i>	matrix outlet pointer

References `_jit_sym_name`, `jit_attr_getsym()`, and `jit_object_attach()`.

Here is the call graph for this function:



36.82.2.33 void max_jit_ob3d_detach (void * x)

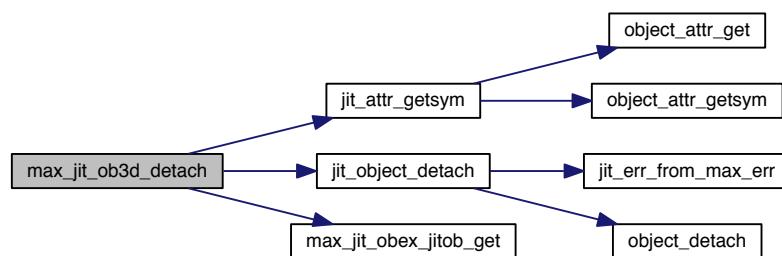
Disposes OB3D Max wrapper related resources.

Parameters

<i>x</i>	Max wrapper object pointer
----------	----------------------------

References `_jit_sym_name`, `jit_attr_getsym()`, `jit_object_detach()`, and `max_jit_obex_jitob_get()`.

Here is the call graph for this function:



36.82.2.34 void max_ob3d_bang (t_max_object * x)

Default OB3D Max wrapper bang method.

Parameters

x	Max object pointer
---	--------------------

References typedmess().

Referenced by max_jit_class_ob3d_wrap(), and max_ob3d_setup().

Here is the call graph for this function:



36.82.2.35 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

Referenced by max_jit_class_ob3d_wrap(), and max_ob3d_setup().

36.82.2.36 long ob3d_auto_get (void *v)

Retrieves automatic flag from opaque t_jit_ob3d struct.

Parameters

v	t_jit_ob3d pointer
---	--------------------

Returns

automatic flag

36.82.2.37 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

36.82.2.38 void ob3d_dest_dim_set (void *v, long width, long height)

Sets destination dimensions in opaque t_jit_ob3d struct.

Parameters

<i>v</i>	t_jit_ob3d pointer
<i>width</i>	destination dimensions width
<i>height</i>	destination dimensions height

36.82.2.39 long ob3d_dirty_get(void * *v*)

Retrieves dirty flag from opaque t_jit_ob3d struct.

Parameters

<i>v</i>	t_jit_ob3d pointer
----------	--------------------

Returns

dirty flag

36.82.2.40 void ob3d_dirty_set(void * *v*, long *c*)

Sets dirty flag from opaque t_jit_ob3d struct.

Parameters

<i>v</i>	t_jit_ob3d pointer
<i>c</i>	dirty flag state

36.82.2.41 long ob3d_enable_get(void * *v*)

Retrieves enable flag from opaque t_jit_ob3d struct.

Parameters

<i>v</i>	t_jit_ob3d pointer
----------	--------------------

Returns

enable flag

36.82.2.42 void* ob3d_jitob_get(void * *v*)

Retrieves parent Jitter object from opaque t_jit_ob3d struct.

Parameters

<i>v</i>	t_jit_ob3d pointer
----------	--------------------

Returns

parent Jitter object pointer

36.82.2.43 void* ob3d_outlet_get(void * *v*)

Retrieves matrix outlet from opaque t_jit_ob3d struct.

Parameters

v	t_jit_ob3d pointer
---	--------------------

Returns

matrix outlet

36.82.2.44 void* ob3d_patcher_get(void * v)

Retrieves containing patcher object from opaque t_jit_ob3d struct.

Parameters

v	t_jit_ob3d pointer
---	--------------------

Returns

containing patcher object

36.82.2.45 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

36.82.2.46 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

36.82.2.47 long ob3d_ui_get(void * v)

Retrieves UI flag from opaque t_jit_ob3d struct.

Parameters

v	t_jit_ob3d pointer
---	--------------------

Returns

UI flag

36.83 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: 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)*

- 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)*

- 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)

- 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: arctangetn (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)

36.83.1 Detailed Description

36.83.2 Function Documentation

36.83.2.1 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.2 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.3 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.4 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.5 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.6 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.7 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.8 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.9 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.10 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.11 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.12 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.13 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.14 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.15 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.16 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.17 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.18 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.19 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.20 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.21 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.22 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.23 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.24 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.25 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.26 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.27 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.28 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.29 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.30 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.31 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.32 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.33 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.34 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.35 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.36 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.37 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.38 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.39 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.40 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.41 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.42 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.43 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.44 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.45 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.46 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.47 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.48 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.49 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by jit_op_vector_flipdiv_char().

36.83.2.50 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by jit_op_vector_flipdiv_float32().

36.83.2.51 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by jit_op_vector_flipdiv_float64().

```
36.83.2.52 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by jit_op_vector_flipdiv_long().

```
36.83.2.53 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

```
36.83.2.54 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

```
36.83.2.55 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.56 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.57 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.58 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.59 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.60 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.61 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.62 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.63 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.64 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.65 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References jit_op_vector_div_char().

Here is the call graph for this function:



36.83.2.66 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*)

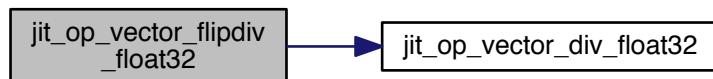
Binary operator: flipped division (float32)

Parameters

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References jit_op_vector_div_float32().

Here is the call graph for this function:



36.83.2.67 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*)

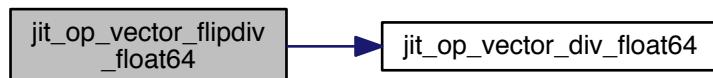
Binary operator: flipped division (float64)

Parameters

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References jit_op_vector_div_float64().

Here is the call graph for this function:



36.83.2.68 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References jit_op_vector_div_long().

Here is the call graph for this function:



36.83.2.69 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*)

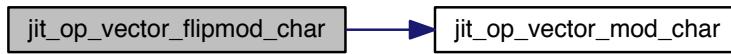
Binary operator: flipped modulo (char)

Parameters

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References jit_op_vector_mod_char().

Here is the call graph for this function:



36.83.2.70 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*)

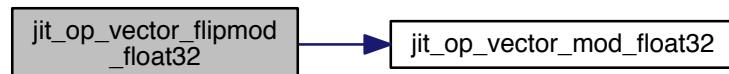
Binary operator: flipped modulo (float32)

Parameters

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References `jit_op_vector_mod_float32()`.

Here is the call graph for this function:



36.83.2.71 `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)`

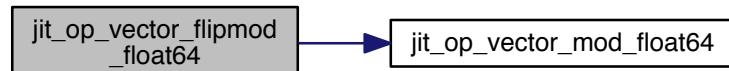
Binary operator: flipped modulo (float64)

Parameters

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References `jit_op_vector_mod_float64()`.

Here is the call graph for this function:



36.83.2.72 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References `jit_op_vector_mod_long()`.

Here is the call graph for this function:



36.83.2.73 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References `jit_op_vector_pass_char()`.

Here is the call graph for this function:



36.83.2.74 `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)`

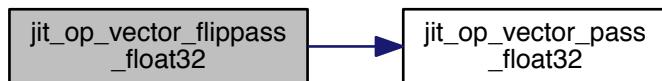
Unary operator: flipped pass (float32)

Parameters

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References jit_op_vector_pass_float32().

Here is the call graph for this function:



36.83.2.75 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*)

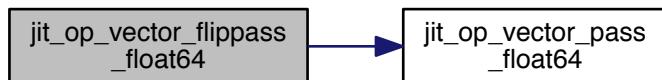
Unary operator: flipped pass (float64)

Parameters

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References jit_op_vector_pass_float64().

Here is the call graph for this function:



36.83.2.76 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References `jit_op_vector_pass_long()`.

Here is the call graph for this function:



36.83.2.77 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References `jit_op_vector_subs_char()`.

Here is the call graph for this function:



36.83.2.78 `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)`

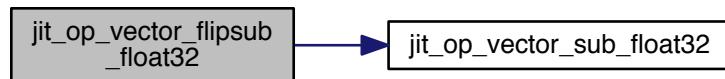
Binary operator: flipped subtraction (float32)

Parameters

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References jit_op_vector_sub_float32().

Here is the call graph for this function:



36.83.2.79 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*)

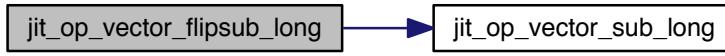
Binary operator: flipped subtraction (long)

Parameters

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References jit_op_vector_sub_long().

Here is the call graph for this function:



36.83.2.80 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.81 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.82 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.83 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.84 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.85 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.86 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.87 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.88 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.89 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.90 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.91 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.92 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.93 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.94 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.95 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.96 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.97 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.98 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.99 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.100 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.101 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.102 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.103 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.104 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.105 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.106 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.107 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.108 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.109 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.110 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.111 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.112 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.113 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.114 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.115 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.116 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.117 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.118 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.119 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.120 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.121 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.122 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.123 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.124 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.125 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.126 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.127 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.128 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.129 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.130 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.131 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.132 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.133 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.134 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by `jit_op_vector_flipmod_char()`.

36.83.2.135 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by `jit_op_vector_flipmod_float32()`.

36.83.2.136 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by jit_op_vector_flipmod_float64().

```
36.83.2.137 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by jit_op_vector_flipmod_long().

```
36.83.2.138 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

```
36.83.2.139 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

```
36.83.2.140 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.141 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.142 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.143 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.144 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.145 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.146 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.147 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.148 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.149 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.150 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.151 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.152 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.153 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.154 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.155 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.156 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.157 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.158 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by `jit_op_vector_flippass_char()`.

36.83.2.159 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by `jit_op_vector_flippass_float32()`.

36.83.2.160 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

References `t_jit_op_info::p`, and `t_jit_op_info::stride`.

Referenced by `jit_op_vector_flippass_float64()`.

36.83.2.161 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by `jit_op_vector_flippass_long()`.

36.83.2.162 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.163 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.164 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

```
36.83.2.165 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

```
36.83.2.166 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

```
36.83.2.167 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

```
36.83.2.168 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.169 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.170 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.171 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.172 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.173 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.174 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.175 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by `jit_op_vector_flipsub_float32()`.

36.83.2.176 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.177 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by `jit_op_vector_flipsub_long()`.

36.83.2.178 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Referenced by `jit_op_vector_flipsub_char()`.

36.83.2.179 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.180 `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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.181 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.182 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.183 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

36.83.2.184 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

```
36.83.2.185 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

```
36.83.2.186 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

<i>n</i>	length of vectors
<i>vecdata</i>	ignored
<i>in0</i>	left input pointer and stride
<i>in1</i>	right input pointer and stride
<i>out</i>	output pointer and stride

Chapter 37

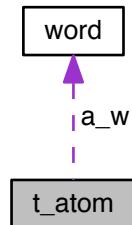
Data Structure Documentation

37.1 t_atom Struct Reference

An atom is a typed datum.

```
#include <ext_mess.h>
```

Collaboration diagram for t_atom:



37.1.1 Detailed Description

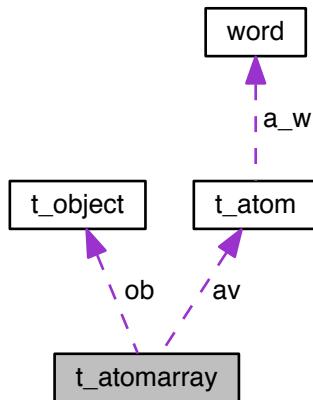
An atom is a typed datum.

37.2 t_atomarray Struct Reference

The atomarray object.

```
#include <ext_atomarray.h>
```

Collaboration diagram for t_atomarray:



37.2.1 Detailed Description

The atomarray object.

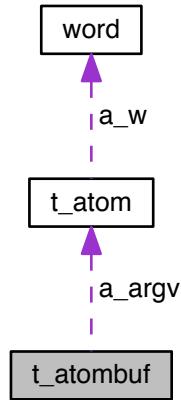
This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

37.3 t_atombuf Struct Reference

The atombuf struct provides a way to pass a collection of atoms.

```
#include <ext_atombuf.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

37.3.1 Detailed Description

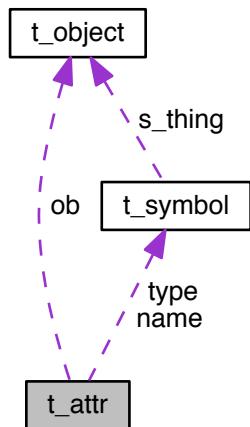
The atombuf struct provides a way to pass a collection of atoms.

37.4 t_attr Struct Reference

Common attr struct.

```
#include <ext_obex.h>
```

Collaboration diagram for t_attr:



37.4.1 Detailed Description

Common attr struct.

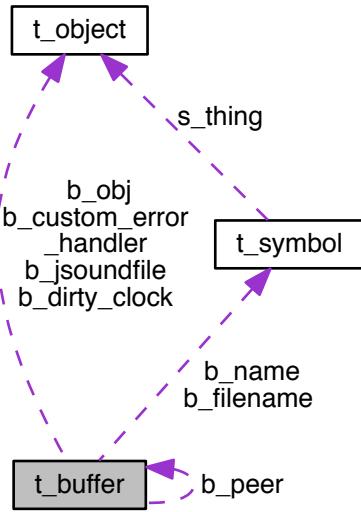
This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

37.5 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_1oversr`**
 $1 / sr$
- **`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`**

- 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 info~
- long **b_olchnchans**
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)
- **t_bool 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
- long **b_padding**
amount of padding (number of samples) in b_memory before b_samples starts
- long **b_paddingchanged**
flag indicating that b_padding has changed and needs to be allocated
- **t_object * b_jsoundfile**
internal instance for reading/writing FLAC format
- **t_systhread_mutex b_mutex**
mutex to use when locking and performing operations anywhere except perform method
- long **b_wasvalid**
internal flag used by replacement or editing operation
- **t_object * b_custom_error_handler**
used to return error numbers to a caller if this object is embedded inside of another object (e.g. playlist~)
- **t_clock * b_dirty_clock**
used to return error numbers to a caller if this object is embedded inside of another object (e.g. playlist~)
- **t_qelem * b_dirty_qelem**
used to move buffer dirty notifications to the main thread
- **t_bool b_dirty_done**
a buffer is not only dirty, but needs the 'done' message sent out its b_doneout outlet
- **t_filepath b_filevol**
path of last file read (not written)

37.5.1 Detailed Description

Data structure for the buffer~ object.

Deprecatated. Use [t_buffer_ref](#) and [t_buffer_obj](#) instead.

37.5.2 Field Documentation

37.5.2.1 `t_clock* t_buffer::b_dirty_clock`

used to return error numbers to a caller if this object is embedded inside of another object (e.g. `playlist~`)

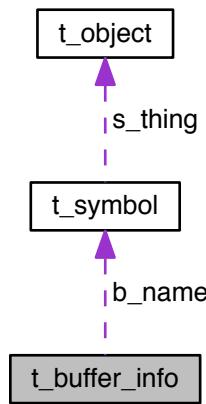
used to move buffer dirty notifications to the main thread

37.6 t_buffer_info Struct Reference

Common buffer~ data/metadata.

```
#include <ext_buffer.h>
```

Collaboration diagram for `t_buffer_info`:



Data Fields

- `t_symbol * b_name`
name of the buffer
- `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
- `long b_modtime`
last modified time ("dirty" method)
- `long b_rfu [57]`
reserved for future use (total struct size is 64x4 = 256 bytes)

37.6.1 Detailed Description

Common buffer~ data/metadata.

This info can be retrieved from a buffer~ using the buffer_getinfo() call.

37.7 t_celldesc Struct Reference

A dataview cell description.

```
#include <jdataview.h>
```

37.7.1 Detailed Description

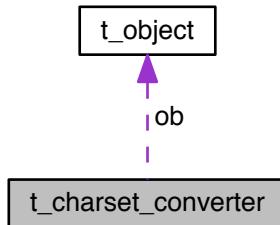
A dataview cell description.

37.8 t_charset_converter Struct Reference

The charset_converter object.

```
#include <ext_charset.h>
```

Collaboration diagram for t_charset_converter:



37.8.1 Detailed Description

The charset_converter object.

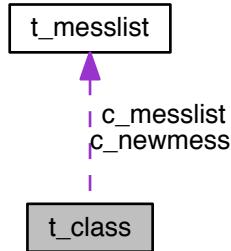
This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

37.9 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

37.9.1 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.

37.10 t_datetime Struct Reference

The Systime data structure.

```
#include <ext_systime.h>
```

Data Fields

- **t_uint32 year**
year
- **t_uint32 month**
month
- **t_uint32 day**
day
- **t_uint32 hour**
hour
- **t_uint32 minute**
minute
- **t_uint32 second**
second

- `t_uint32 millisecond`
(reserved for future use)

37.10.1 Detailed Description

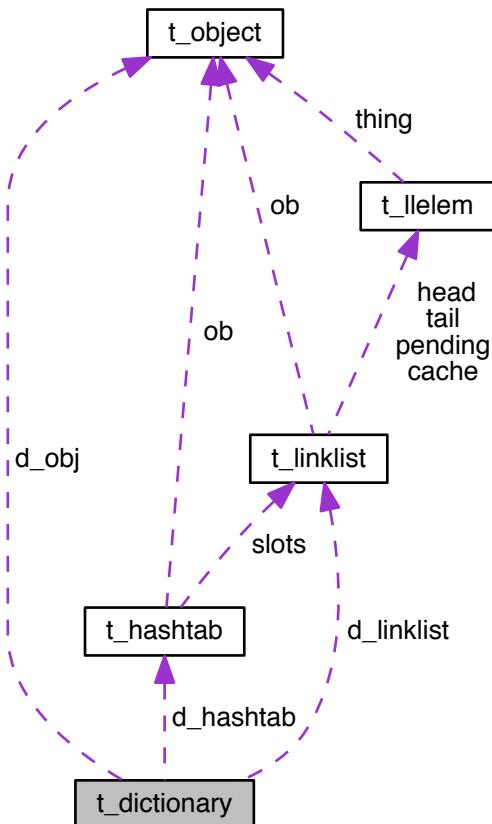
The Systime data structure.

37.11 t_dictionary Struct Reference

The dictionary object.

```
#include <ext_dictionary.h>
```

Collaboration diagram for t_dictionary:



37.11.1 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

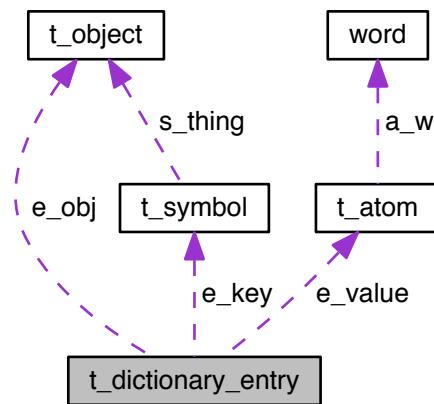
[t_dictionary](#)

37.12 t_dictionary_entry Struct Reference

A dictionary entry.

```
#include <ext_dictionary.h>
```

Collaboration diagram for t_dictionary_entry:



37.12.1 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](#)

37.13 t_ex_ex Struct Reference

`ex_ex`.

```
#include <ext_expr.h>
```

Data Fields

- union {
- } [ex_cont](#)

content

- long `ex_type`
type of the node

37.13.1 Detailed Description

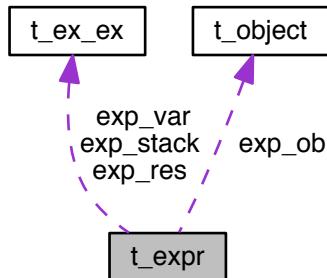
`ex_ex.`

37.14 t_expr Struct Reference

Struct for an instance of expr.

```
#include <ext_expr.h>
```

Collaboration diagram for t_expr:



Data Fields

- `t_ex_ex exp_res`
the result of last evaluation

37.14.1 Detailed Description

Struct for an instance of expr.

37.15 t_fileinfo Struct Reference

Information about a file.

```
#include <ext_path.h>
```

Data Fields

- `t_fourcc type`
type (four-char-code)
- `t_fourcc creator`

- Mac-only creator (four-char-code)*
- `t_uint32 unused`
this was date but it wasn't populated and it wasn't used
 - `t_int32 flags`
One of the values defined in `e_max_fileinfo_flags`.

37.15.1 Detailed Description

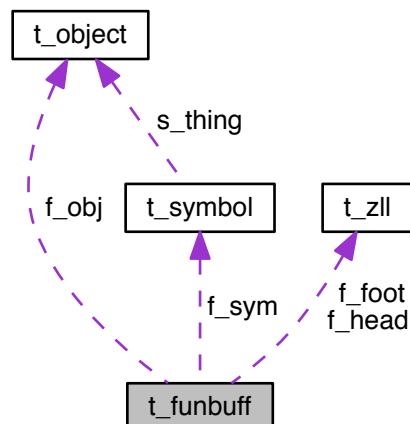
Information about a file.

37.16 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

37.16.1 Detailed Description

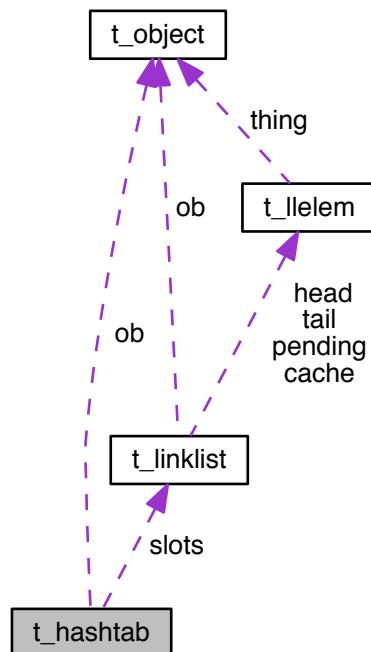
The structure of a funbuff object.

37.17 t_hashtab Struct Reference

The hashtab object.

```
#include <ext_hashtab.h>
```

Collaboration diagram for t_hashtab:



37.17.1 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

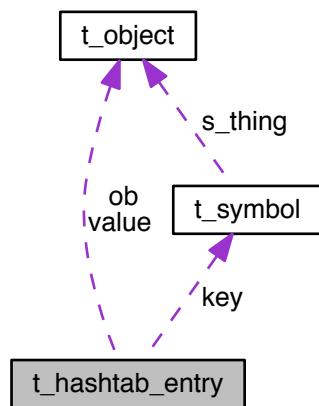
[t_hashtab](#)

37.18 t_hashtab_entry Struct Reference

A hashtab entry.

```
#include <ext_hashtab.h>
```

Collaboration diagram for t_hashtab_entry:



37.18.1 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

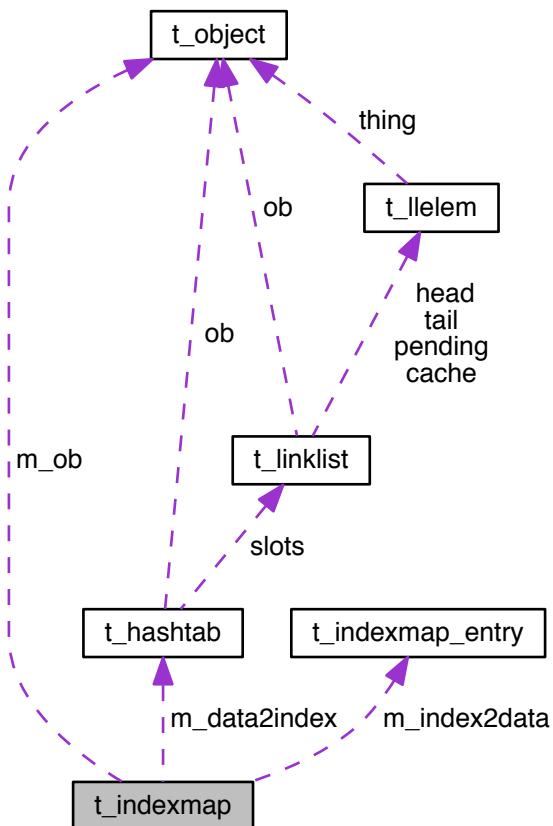
t hashtable

37.19 t_indexmap Struct Reference

An indexmap object.

```
#include <indexmap.h>
```

Collaboration diagram for t_indexmap:



37.19.1 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

37.20 t_indexmap_entry Struct Reference

An indexmap element.

```
#include <indexmap.h>
```

37.20.1 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

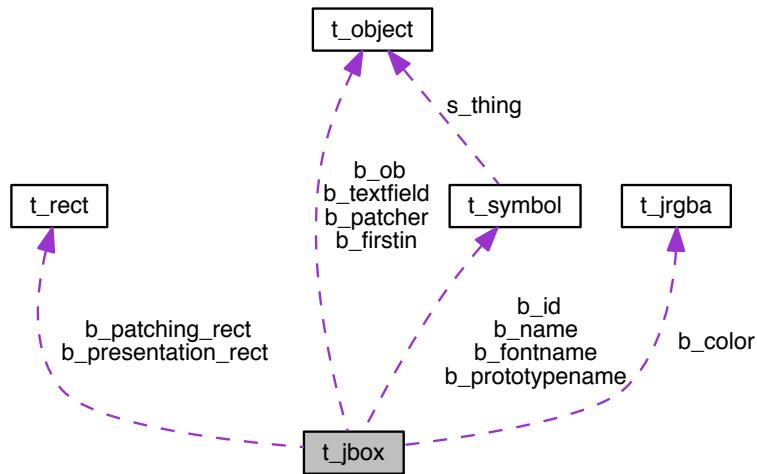
[t_indexmap](#)

37.21 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](#):



37.21.1 Detailed Description

The [t_jbox](#) struct provides the header for a Max user-interface object.

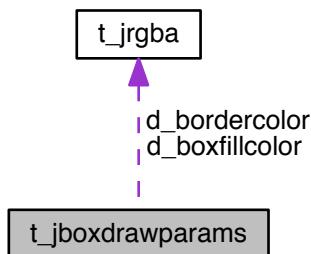
This struct should be considered opaque and is subject to change without notice. Do not access its members directly in any code.

37.22 t_jboxdrawparams Struct Reference

The [t_jboxdrawparams](#) structure.

```
#include <jpatcher_api.h>
```

Collaboration diagram for t_jboxdrawparams:



37.22.1 Detailed Description

The [t_jboxdrawparams](#) structure.

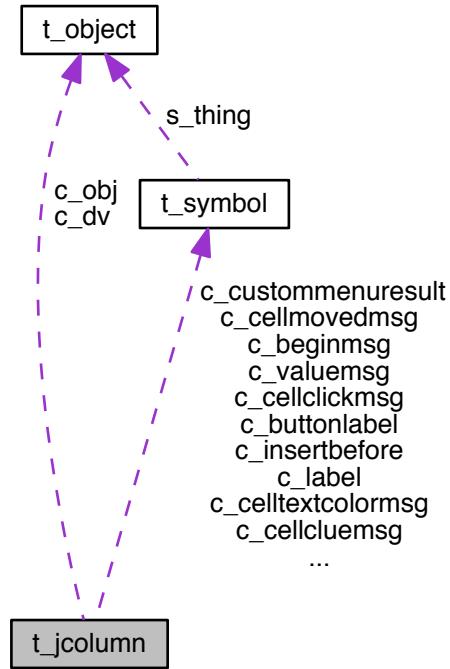
This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

37.23 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_FORWARDS 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)
- char `c_showinfo`
show info button for cell clue on mouse over
- 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
- `t_symbol * c_cellentermsg`
message for cell enter
- `t_symbol * c_cellexitmsg`
message for cell exit
- `t_symbol * c_cellmovedmsg`
message for cell mouse move
- `t_symbol * c_cellclickmsg`
message for custom cell click action

37.23.1 Detailed Description

A datatable column.

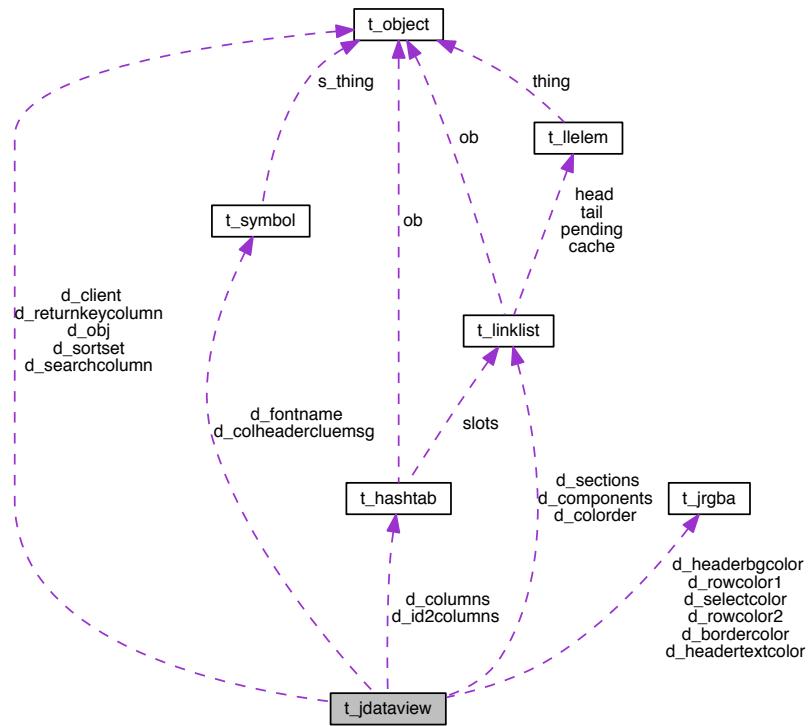
Columns for a given datatable are stored in a `t_hashtab` and accessed by name.

37.24 t_jdatatable Struct Reference

The datatable object.

```
#include <jdatatable.h>
```

Collaboration diagram for t_jdataview:



Data Fields

- `t_linklist * 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
- `void * d_rowmap_obsolete`
no longer used
- 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_rgba 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_overridefocus**
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)
- **t_linklist * d_sections**
list of sections
- char **d_paintcellseparator**
should paint a line below a cell (grayish)
- **t_object * d_sortset**
sort col saved when dv is invisible

37.24.1 Detailed Description

The dataview object.

37.25 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 height.
- double **max_x_advance**

- **double max_y_advance**
Unused / Not valid.

37.25.1 Detailed Description

A structure for holding information related to how much space the rendering of a given font will use.

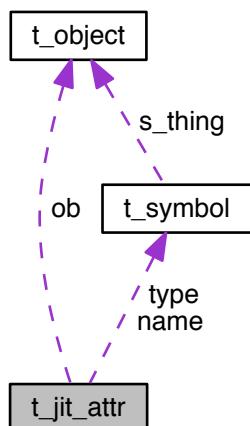
The units for these measurements is in pixels.

37.26 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

37.26.1 Detailed Description

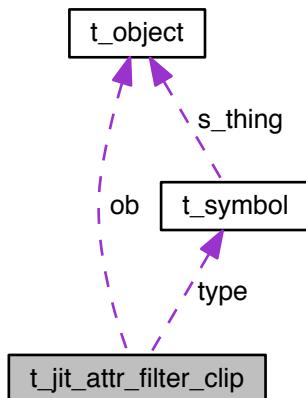
Common attribute struct.

Shared by all built in attribute classes.

37.27 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

37.27.1 Detailed Description

[t_jit_attr_clip](#) object struct.

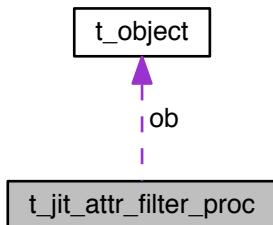
Warning

This struct should not be accessed directly, but is provided for reference when calling Jitter attribute functions.

37.28 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

37.28.1 Detailed Description

[t_jit_attr_filter_proc](#) object struct.

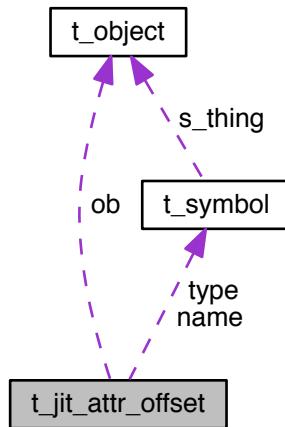
Warning

This struct should not be accessed directly, but is provided for reference when calling Jitter attribute functions.

37.29 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

37.29.1 Detailed Description

[t_jit_attr_offset](#) object struct.

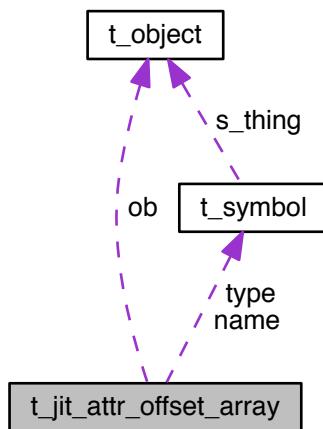
Warning

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.

37.30 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

37.30.1 Detailed Description

[t_jit_attr_offset_array](#) object struct.

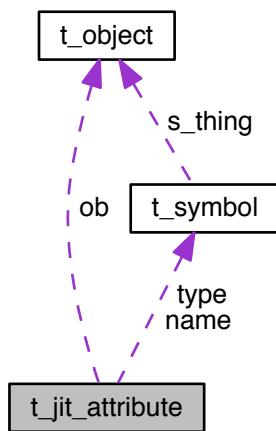
Warning

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.

37.31 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

37.31.1 Detailed Description

[t_jit_attribute](#) object struct.

Warning

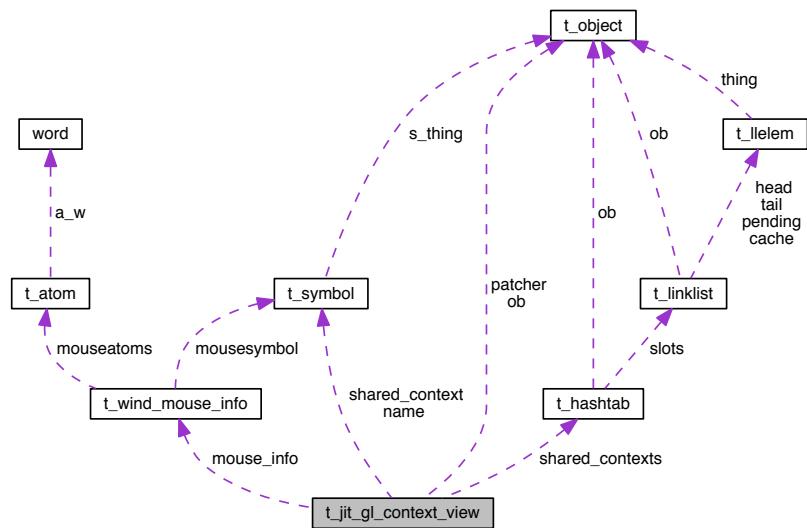
This struct should not be accessed directly, but is provided for reference. Attribute objects do not typically use attributes themselves to access members, but rather accessor methods—i.e. use `jit_object_method` in place of the `jit_attr_*` functions to access attribute state.

37.32 t_jit_gl_context_view Struct Reference

`t_jit_gl_context_view` object struct.

```
#include <jit/gl/context/view.h>
```

Collaboration diagram for t_jit_gl_context_view:



Data Fields

- `t_object ob`
 - jitter object*
 - long `rebuild`
 - rebuild flag*
 - `t_jit_gl_context context`
 - OpenGL context.*
 - `t_symbol * shared_context`
 - shared context name*
 - `t_wind_mouse_info mouse_info`
 - data for mouse events*
 - `t_wind_mousewheel_info mousewheel_info`
 - data for mouse wheel events*
 - `t_wind_key_info key_info`
 - data for key events*
 - long `canrebuild`
 - flag for whether the context can rebuild or not*

- long `doublebuffer`
double buffer flag
- long `depthbuffer`
depth buffer flag
- long `stereo`
active stereo flag
- `t_jit_rect frame`
frame of context
- long `fsaa`
FSAA flag.
- long `sync`
V-sync flag.
- long `shared`
Shader context flag.
- `t_hashtab * shared_contexts`
Hashtab of shared context names.
- long `idlemouse`
Idlemouse flag (events on mouse move)
- long `mousewheel`
mosuewheel flag (events on mouse wheel)
- void * `target`
target object we're controlled by
- long `targettype`
target type we're controlled by
- `t_symbol * name`
name of the view
- long `reshaping`
flag for breaking cycles on reshape notification
- long `ownerreshape`
flag for if the owner handles reshaping the context
- `t_object * patcher`
patcher the context view is in (if there is one)
- long `freeing`
in the process of freeing flag
- long `creating`
in the process of creating flag
- float `scalefactor`
scaling factor when drawing to retina display
- long `allow_hi_res`
allows for high resolution drawing when available

37.32.1 Detailed Description

`t_jit_gl_context_view` object struct.

Manages an OpenGL context within a rectangle. Objects that use a `t_jit_gl_context_view` to manage an OpenGL context should attach themselves to the object for its lifetime and implement an "update" method in order to handle modifications to the `t_jit_gl_context_view` that may require a rebuild or further response within the embedding object.

37.33 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

37.33.1 Detailed Description

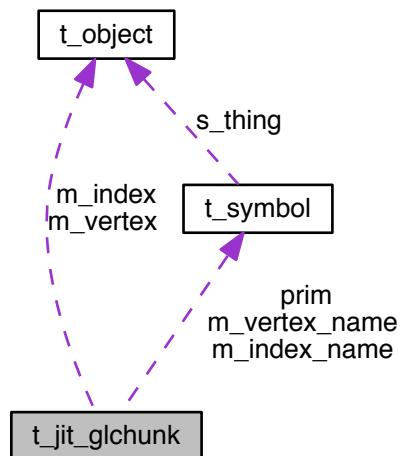
[t_jit_gl_drawinfo](#) struct used for tasks such as multi texture unit binding.

37.34 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

37.34.1 Detailed Description

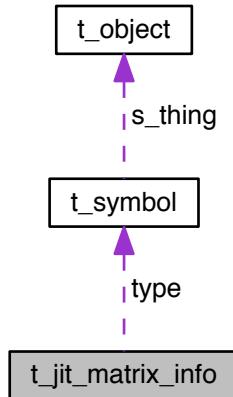
`t_jit_glchunk` is a public structure to store one gl-command's-worth of data, in a format which can be passed easily to `glDrawRangeElements`, and `matrixoutput`.

37.35 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

37.35.1 Detailed Description

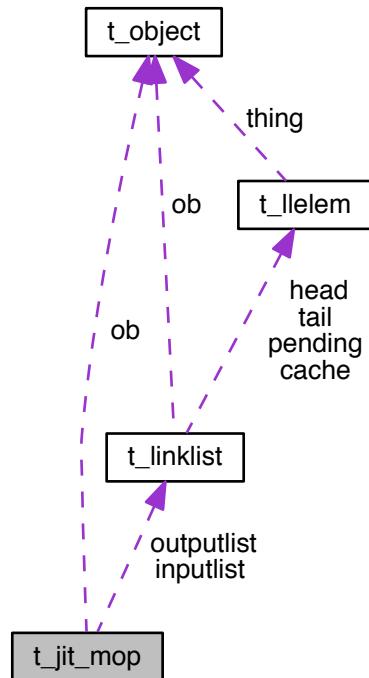
Matrix information struct.

Used to get/set multiple matrix attributes at once.

37.36 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`
"inputcount" attribute
- `t_jit_linklist * inputlist`
linked list of inputs, accessed via methods
- `t_jit_linklist * outputlist`
linked list of inputs, accessed via methods
- `char caninplace`
deprecated
- `char adapt`
"adapt" attribute
- `char outputmode`
"outputmode" attribute

37.36.1 Detailed Description

`t_jit_mop` object struct.

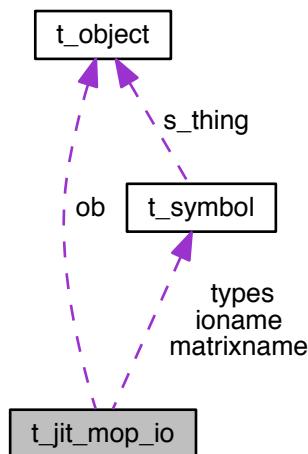
Warning

This struct should not be accessed directly, but is provided for reference when calling Jitter attribute functions.

37.37 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`

- 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.

37.37.1 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.

37.38 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)

37.38.1 Detailed Description

Provides base pointer and stride for vector operator functions.

37.39 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

37.39.1 Detailed Description

An affine transformation (such as scale, shear, etc).

37.40 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].

37.40.1 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](#)

37.41 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].

37.41.1 Detailed Description

A color composed of red, green, blue, and alpha components.

37.42 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

37.42.1 Detailed Description

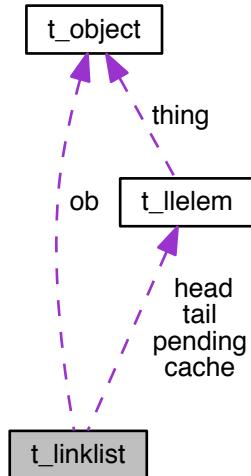
Line or line segment in 3D space (GLfloat)

37.43 t_linklist Struct Reference

The linklist object.

```
#include <ext_linklist.h>
```

Collaboration diagram for t_linklist:



37.43.1 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

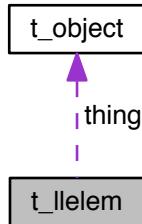
[t_llelem](#)

37.44 t_llelem Struct Reference

A linklist element.

```
#include <ext_linklist.h>
```

Collaboration diagram for t_llelem:



37.44.1 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](#)

37.45 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

37.45.1 Detailed Description

Matrix conversion struct.

Used to copy data from one matrix to another with special characteristics.

37.46 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](#) [MSG_MAXARG+1]
Argument type information.

37.46.1 Detailed Description

A list of symbols and their corresponding methods, complete with typechecking information.

37.47 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 -1 entry of the message list of an object contains a pointer to its [t_class](#) entry.
- [t_ptr_int o_magic](#)
magic number
- struct inlet * [o_inlet](#)
list of inlets
- struct outlet * [o_outlet](#)
list of outlets

37.47.1 Detailed Description

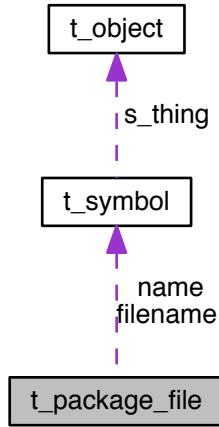
The structure for the head of any object which wants to have inlets or outlets, or support attributes.

37.48 t_package_file Struct Reference

A container for a path/filename pair, together with additional meta fields for special cases.

```
#include <ext_packages.h>
```

Collaboration diagram for t_package_file:



37.48.1 Detailed Description

A container for a path/filename pair, together with additional meta fields for special cases.

37.49 t_path Struct Reference

The path data structure.

```
#include <ext_path.h>
```

37.49.1 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.

37.50 t_pathlink Struct Reference

The pathlink data structure.

```
#include <ext_path.h>
```

37.50.1 Detailed Description

The pathlink data structure.

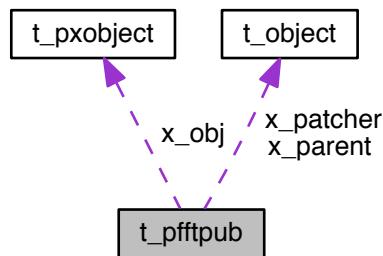
This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

37.51 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_ftoffset`
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)?

37.51.1 Detailed Description

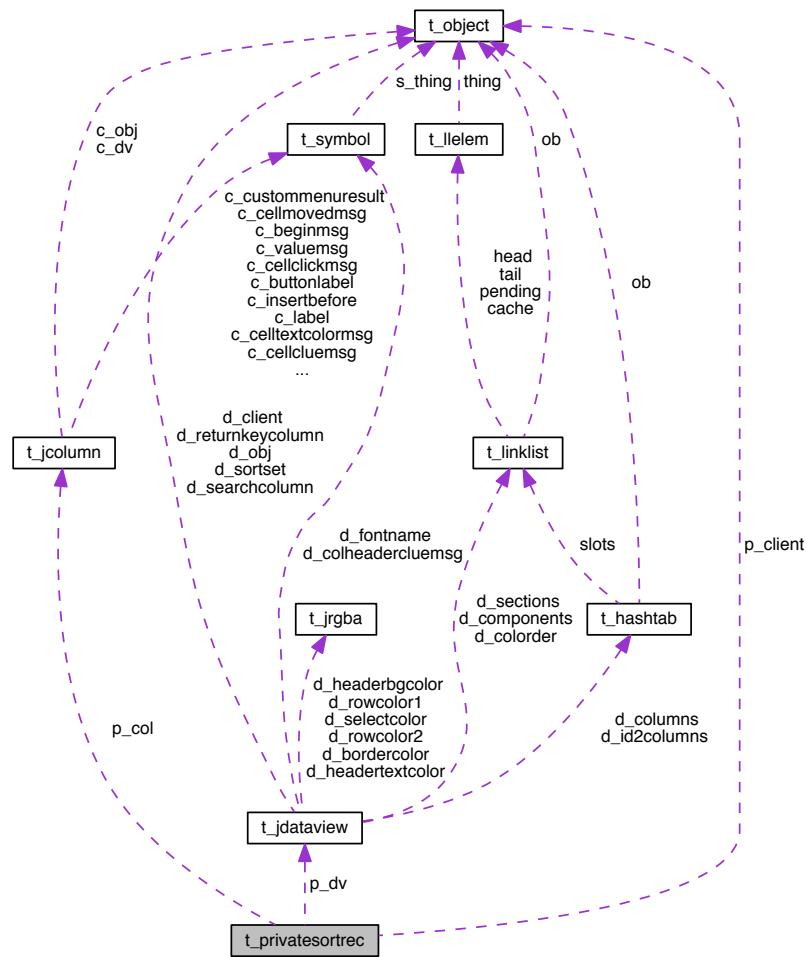
Public FFT Patcher struct.

37.52 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`
1 if sorting "forwards"
- `t_object * p_client`
pointer to the client object
- `t_jdataview * p_dv`
pointer to the dataview

37.52.1 Detailed Description

used to pass data to a client sort function

37.53 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.

37.53.1 Detailed Description

Coordinates for specifying a point.

See also

[t_rect](#)
[t_size](#)

37.54 t_pxdata Struct Reference

Common struct for MSP objects.

```
#include <z_dsp.h>
```

Data Fields

- 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](#)

37.54.1 Detailed Description

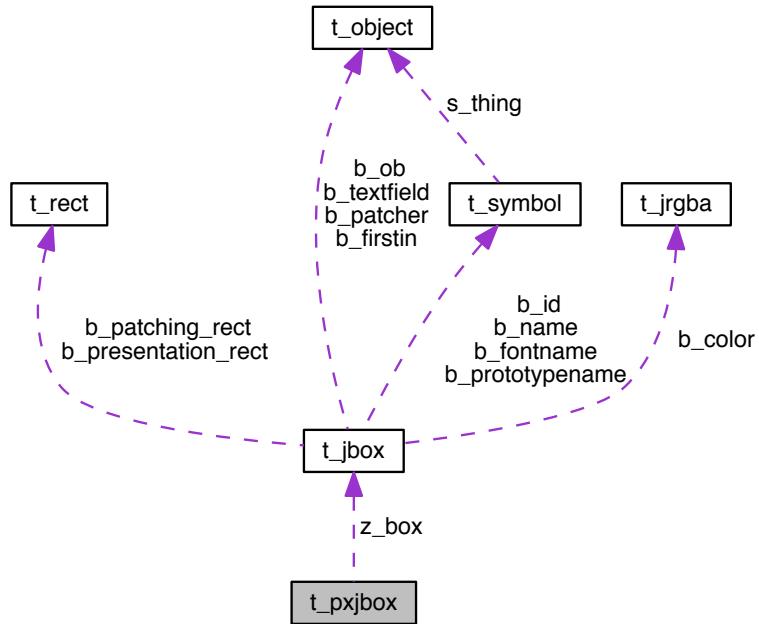
Common struct for MSP objects.

37.55 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`

37.55.1 Detailed Description

Header for any ui signal processing object.

For non-ui objects use `t_pxobject`.

37.56 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](#)

37.56.1 Detailed Description

Header for any non-ui signal processing object.

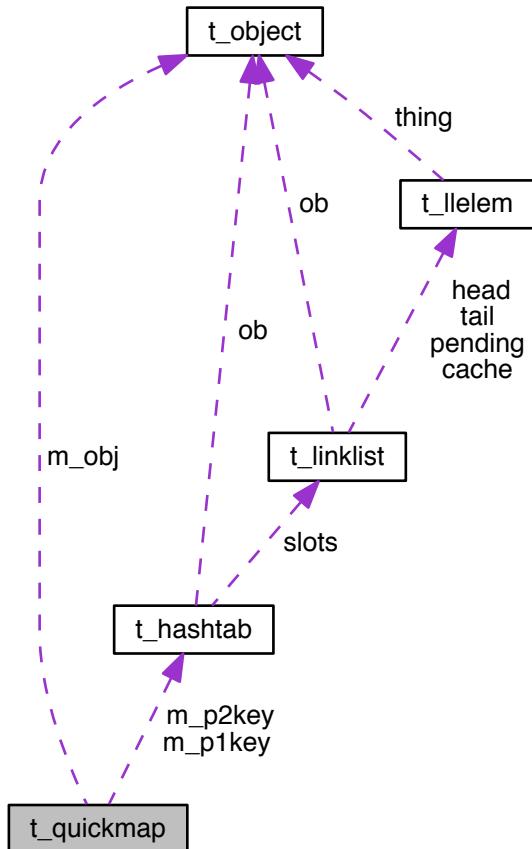
For ui objects use [t_pxjbox](#).

37.57 t_quickmap Struct Reference

The quickmap object.

```
#include <ext_quickmap.h>
```

Collaboration diagram for t_quickmap:



37.57.1 Detailed Description

The quickmap object.

This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

37.58 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.

37.58.1 Detailed Description

Coordinates for specifying a rectangular region.

See also

[t_pt](#)

[t_size](#)

37.59 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.

37.59.1 Detailed Description

The signal data structure.

37.60 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.

37.60.1 Detailed Description

Coordinates for specifying the size of a region.

See also

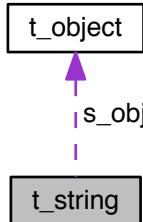
[t_rect](#)
[t_pt](#)

37.61 t_string Struct Reference

The string object.

```
#include <ext_objstring.h>
```

Collaboration diagram for t_string:



37.61.1 Detailed Description

The string object.

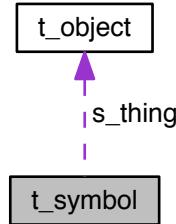
This struct is provided for debugging convenience, but should be considered opaque and is subject to change without notice.

37.62 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`

37.62.1 Detailed Description

The symbol.

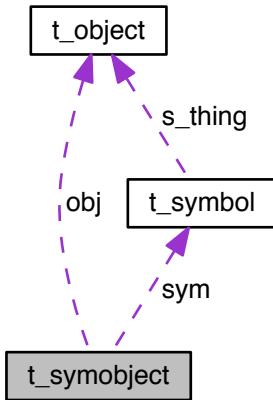
Note: You should *never* manipulate the `s_name` field of the `t_symbol` directly! Doing so will corrupt Max's symbol table. Instead, *always* use `gensym()` to get a symbol with the desired string contents for the `s_name` field.

37.63 `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.

37.63.1 Detailed Description

The symobject data structure.

37.64 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
- `long t_magic`
magic number

37.64.1 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()`).

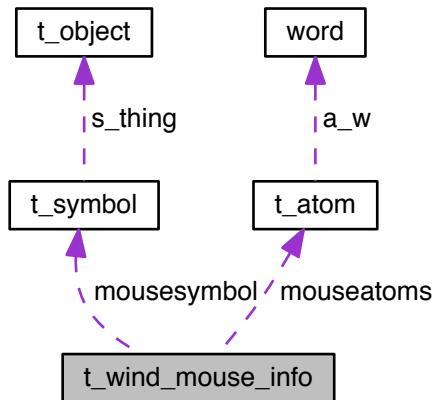
In general, you should use `t_object` instead.

37.65 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

- `t_atom mouseatoms [8]`
h, v, (up/down), cmdKey, shiftKey, alphaLock, option, control.
- `int argc`
argument count
- `t_symbol * mousesymbol`
mouse event type

37.65.1 Detailed Description

`t_wind_mouse_info_struct` provided by `jit.window` and `jit.pwindow` mouse events

37.66 t_zll Struct Reference

A simple doubly-linked list used by the `t_funbuff` object.

```
#include <ext_maxtypes.h>
```

37.66.1 Detailed Description

A simple doubly-linked list used by the [t_funbuff](#) object.

37.67 word Union Reference

Union for packing any of the datum defined in [e_max_atomtypes](#).

```
#include <ext_mess.h>
```

Data Fields

- [t_atom_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

37.67.1 Detailed Description

Union for packing any of the datum defined in [e_max_atomtypes](#).

Index

_buffer
 b_dirty_clock, [923](#)

A_CANT
 Atoms, [325](#)

A_COMMA
 Atoms, [325](#)

A_DEFER
 Atoms, [325](#)

A_DEFER_LOW
 Atoms, [325](#)

A_DEFFLOAT
 Atoms, [325](#)

A_DEFLONG
 Atoms, [325](#)

A_DEFSYM
 Atoms, [325](#)

A_DOLLAR
 Atoms, [325](#)

A_DOLLSYM
 Atoms, [325](#)

A_FLOAT
 Atoms, [325](#)

A_GIMME
 Atoms, [325](#)

A_GIMMEBACK
 Atoms, [325](#)

A_LONG
 Atoms, [325](#)

A_NOTHING
 Atoms, [325](#)

A_OBJ
 Atoms, [325](#)

A_SEMI
 Atoms, [325](#)

A_SYM
 Atoms, [325](#)

A_USURP
 Atoms, [325](#)

A_USURP_LOW
 Atoms, [325](#)

ATOMARRAY_FLAG_FREECHILDREN
 Atom Array, [232](#)

ATTR_FLAGS_NONE
 Attributes, [183](#)

ATTR_GET_OPAQUE
 Attributes, [183](#)

ATTR_GET_OPAQUE_USER
 Attributes, [184](#)

ATTR_SET_OPAQUE
 Attributes, [184](#)

ATTR_SET_OPAQUE_USER
 Attributes, [184](#)

aaCancel
 Miscellaneous, [399](#)

aaNo
 Miscellaneous, [399](#)

aaYes
 Miscellaneous, [399](#)

addbang
 Old-Style Classes, [216](#)

addfloat
 Old-Style Classes, [216](#)

addftx
 Old-Style Classes, [216](#)

addint
 Old-Style Classes, [216](#)

addinx
 Old-Style Classes, [216](#)

addmess
 Old-Style Classes, [217](#)

alias
 Old-Style Classes, [217](#)

Atom Array, [231](#)
 ATOMARRAY_FLAG_FREECHILDREN, [232](#)
 atomarray_appendatom, [232](#)
 atomarray_appendatoms, [232](#)
 atomarray_chuckindex, [233](#)
 atomarray_clear, [233](#)
 atomarray_clone, [233](#)
 atomarray_copyatoms, [233](#)
 atomarray_duplicate, [234](#)
 atomarray_flags, [234](#)
 atomarray_funall, [234](#)
 atomarray_getatoms, [235](#)
 atomarray_getflags, [235](#)
 atomarray_getindex, [235](#)
 atomarray_getsize, [237](#)
 atomarray_new, [237](#)
 atomarray_setatoms, [237](#)

Atom Module, [654](#)
 jit_atom_arg_getdouble, [654](#)
 jit_atom_arg_getfloat, [655](#)
 jit_atom_arg_getlong, [655](#)
 jit_atom_arg_getsym, [656](#)
 jit_atom_getcharfix, [657](#)
 jit_atom_getfloat, [657](#)
 jit_atom_getlong, [658](#)
 jit_atom_getobj, [658](#)

jit_atom_getsym, 659
jit_atom_setfloat, 659
jit_atom_setlong, 659
jit_atom_setobj, 661
jit_atom_setsym, 661
atom_alloc
 Atoms, 325
atom_alloc_array
 Atoms, 326
atom_arg_getdouble
 Atoms, 326
atom_arg_getfloat
 Atoms, 326
atom_arg_getlong
 Atoms, 327
atom_arg_getobjclass
 Atoms, 327
atom_arg_getsym
 Atoms, 328
atom_copy
 Atoms, 328
atom_getatom_array
 Atoms, 329
atom_getchar_array
 Atoms, 329
atom_getcharfix
 Atoms, 329
atom_getdouble_array
 Atoms, 329
atom_getfloat
 Atoms, 330
atom_getfloat_array
 Atoms, 330
atom_getformat
 Atoms, 330
atom_getlong
 Atoms, 331
atom_getlong_array
 Atoms, 331
atom_getobj
 Atoms, 331
atom_getobj_array
 Atoms, 332
atom_getobjclass
 Atoms, 332
atom_getsym
 Atoms, 332
atom_getsym_array
 Atoms, 332
atom_gettext
 Atoms, 334
atom_gettime
 Atoms, 334
atom_setatom_array
 Atoms, 334
atom_setchar_array
 Atoms, 335
atom_setdouble_array
 Atoms, 335
Atoms, 335
atom_setfloat
 Atoms, 335
atom_setfloat_array
 Atoms, 335
atom_setformat
 Atoms, 337
atom_setlong
 Atoms, 337
atom_setlong_array
 Atoms, 337
atom_setobj
 Atoms, 338
atom_setobj_array
 Atoms, 338
atom_setparse
 Atoms, 338
atom_setsym
 Atoms, 339
atom_setsym_array
 Atoms, 339
atomarray_appendatom
 Atom Array, 232
atomarray_appendatoms
 Atom Array, 232
atomarray_chuckindex
 Atom Array, 233
atomarray_clear
 Atom Array, 233
atomarray_clone
 Atom Array, 233
atomarray_copyatoms
 Atom Array, 233
atomarray_duplicate
 Atom Array, 234
atomarray_flags
 Atom Array, 234
atomarray_funall
 Atom Array, 234
atomarray_getatoms
 Atom Array, 235
atomarray_getflags
 Atom Array, 235
atomarray_getindex
 Atom Array, 235
atomarray_getsize
 Atom Array, 237
atomarray_new
 Atom Array, 237
atomarray_setatoms
 Atom Array, 237
atombuf_free
 Atombufs, 341
atombuf_new
 Atombufs, 341
atombuf_text
 Atombufs, 342
Atombufs, 341

atombuf_free, 341
atombuf_new, 341
atombuf_text, 342
atomisatomarray
 Atoms, 339
atomisdictionary
 Atoms, 340
atomisstring
 Atoms, 340
Atoms, 322
 A_CANT, 325
 A_COMMAS, 325
 A_DEFER, 325
 A_DEFER_LOW, 325
 A_DEFFLOAT, 325
 A_DEFLONG, 325
 A_DEFSYM, 325
 A_DOLLAR, 325
 A_DOLLSYM, 325
 A_FLOAT, 325
 A_GIMME, 325
 A_GIMMEBACK, 325
 A_LONG, 325
 A_NOTHING, 325
 A_OBJ, 325
 A_SEMI, 325
 A_SYM, 325
 A_USURP, 325
 A_USURP_LOW, 325
atom_alloc, 325
atom_alloc_array, 326
atom_arg_getdouble, 326
atom_arg_getfloat, 326
atom_arg_getlong, 327
atom_arg_getobjclass, 327
atom_arg_getsym, 328
atom_copy, 328
atom_getatom_array, 329
atom_getchar_array, 329
atom_getcharfix, 329
atom_getdouble_array, 329
atom_getfloat, 330
atom_getfloat_array, 330
atom_getformat, 330
atom_getlong, 331
atom_getlong_array, 331
atom_getobj, 331
atom_getobj_array, 332
atom_getobjclass, 332
atom_getsym, 332
atom_getsym_array, 332
atom_gettext, 334
atom_gettime, 334
atom_setatom_array, 334
atom_setchar_array, 335
atom_setdouble_array, 335
atom_setfloat, 335
atom_setfloat_array, 335
atom_setformat, 337
atom_setlong, 337
atom_setlong_array, 337
atom_setobj, 338
atom_setobj_array, 338
atom_setparse, 338
atom_setsym, 339
atom_setsym_array, 339
atomisatomarray, 339
atomisdictionary, 340
atomisstring, 340
e_max_atom_gettext_flags, 324
e_max_atomtypes, 324
OBEX_UTIL_ATOM_GETTEXT_COMMA_DELIM, 324
OBEX_UTIL_ATOM_GETTEXT_DEFAULT, 324
OBEX_UTIL_ATOM_GETTEXT_FORCE_ZEROS, 324
OBEX_UTIL_ATOM_GETTEXT_NUM_HI_RES, 324
OBEX_UTIL_ATOM_GETTEXT_NUM_LO_RES, 324
OBEX_UTIL_ATOM_GETTEXT_SYM_FORCE_QUOTE, 324
OBEX_UTIL_ATOM_GETTEXT_SYM_NO_QUOTE, 324
OBEX_UTIL_ATOM_GETTEXT_TRUNCATE_ZERO, 324
postargs, 340
atoms_to_jrgba
 Colors, 617
attr_addfilter_clip
 Attributes, 184
attr_addfilter_clip_scale
 Attributes, 184
attr_addfilterget_clip
 Attributes, 184
attr_addfilterget_clip_scale
 Attributes, 185
attr_addfilterget_proc
 Attributes, 185
attr_addfilterset_clip
 Attributes, 186
attr_addfilterset_clip_scale
 Attributes, 186
attr_addfilterset_proc
 Attributes, 186
attr_args_dictionary
 Attributes, 187
attr_args_offset
 Attributes, 187
attr_args_process
 Attributes, 188
attr_dictionary_check
 Attributes, 188
attr_dictionary_process
 Attributes, 188
attr_offset_array_new

Attributes, 189
attr_offset_new
 Attributes, 189
Attribute Module, 663
jit_attr_caget, 665
jit_attr_canset, 666
jit_attr_filter_clip_new, 666
jit_attr_filter_proc_new, 666
jit_attr_filterget, 667
jit_attr_filterset, 667
jit_attr_get, 668
jit_attr_getchar_array, 668
jit_attr_getdouble_array, 669
jit_attr_getfloat, 669
jit_attr_getfloat_array, 670
jit_attr_getlong, 670
jit_attr_getlong_array, 672
jit_attr_getmethod, 673
jit_attr_getname, 673
jit_attr_getsym, 673
jit_attr_getsym_array, 674
jit_attr_gettype, 675
jit_attr_offset_array_new, 675
jit_attr_offset_new, 676
jit_attr_set, 677
jit_attr_setchar_array, 677
jit_attr_setdouble_array, 678
jit_attr_setfloat, 679
jit_attr_setfloat_array, 679
jit_attr_setlong, 680
jit_attr_setlong_array, 680
jit_attr_setsym, 681
jit_attr_setsym_array, 682
jit_attr_symcompare, 682
jit_attr_usercanget, 684
jit_attr_usercanset, 684
jit_attribute_new, 685
attribute_new
 Attributes, 190
Attributes, 145
ATTR_FLAGS_NONE, 183
ATTR_GET_OPAQUE, 183
ATTR_GET_OPAQUE_USER, 184
ATTR_SET_OPAQUE, 184
ATTR_SET_OPAQUE_USER, 184
attr_addfilter_clip, 184
attr_addfilter_clip_scale, 184
attr_addfilterget_clip, 184
attr_addfilterget_clip_scale, 185
attr_addfilterget_proc, 185
attr_addfilterset_clip, 186
attr_addfilterset_clip_scale, 186
attr_addfilterset_proc, 186
attr_args_dictionary, 187
attr_args_offset, 187
attr_args_process, 188
attr_dictionary_check, 188
attr_dictionary_process, 188
attr_offset_array_new, 189
attr_offset_new, 189
attribute_new, 190
CLASS_ATTR_ACCESSORS, 153
CLASS_ATTR_ADD_FLAGS, 153
CLASS_ATTR_ALIAS, 154
CLASS_ATTR_ATOM, 154
CLASS_ATTR_ATOM_ARRAY, 154
CLASS_ATTR_ATOM_LONG, 154
CLASS_ATTR_ATOM_LONG_ARRAY, 155
CLASS_ATTR_ATOM_VARSIZE, 155
CLASS_ATTR_BASIC, 155
CLASS_ATTR_CATEGORY, 155
CLASS_ATTR_CHAR, 156
CLASS_ATTR_CHAR_ARRAY, 156
CLASS_ATTR_CHAR_VARSIZE, 156
CLASS_ATTR_DEFAULT, 156
CLASS_ATTR_DEFAULT_PAINT, 157
CLASS_ATTR_DEFAULT_SAVE, 157
CLASS_ATTR_DEFAULT_SAVE_PAINT, 157
CLASS_ATTR_DEFAULTNAME, 157
CLASS_ATTR_DEFAULTNAME_PAINT, 158
CLASS_ATTR_DEFAULTNAME_SAVE, 158
CLASS_ATTR_DEFAULTNAME_SAVE_PAINT, 158
CLASS_ATTR_DOUBLE, 159
CLASS_ATTR_DOUBLE_ARRAY, 159
CLASS_ATTR_DOUBLE_VARSIZE, 159
CLASS_ATTR_ENUM, 160
CLASS_ATTR_ENUMINDEX, 160
CLASS_ATTR_FILTER_CLIP, 160
CLASS_ATTR_FILTER_MAX, 161
CLASS_ATTR_FILTER_MIN, 161
CLASS_ATTR_FLOAT, 161
CLASS_ATTR_FLOAT_ARRAY, 162
CLASS_ATTR_FLOAT_VARSIZE, 162
CLASS_ATTR_INT32, 162
CLASS_ATTR_INTRODUCED, 162
CLASS_ATTR_INVISIBLE, 163
CLASS_ATTR_LABEL, 163
CLASS_ATTR_LEGACYDEFAULT, 163
CLASS_ATTR_LONG, 163
CLASS_ATTR_LONG_ARRAY, 164
CLASS_ATTR_LONG_VARSIZE, 164
CLASS_ATTR_MAX, 164
CLASS_ATTR_MIN, 164
CLASS_ATTR_OBJ, 165
CLASS_ATTR_OBJ_ARRAY, 165
CLASS_ATTR_OBJ_VARSIZE, 165
CLASS_ATTR_OBSOLETE, 166
CLASS_ATTR_ORDER, 166
CLASS_ATTR_PAINT, 166
CLASS_ATTR_REMOVE_FLAGS, 166
CLASS_ATTR_RENAMED, 167
CLASS_ATTR_RGBA, 167
CLASS_ATTR_SAVE, 167
CLASS_ATTR_SELFSAVE, 167
CLASS_ATTR_STYLE, 167

CLASS_ATTR_STYLE_LABEL, 168
CLASS_ATTR_SYM, 168
CLASS_ATTR_SYM_ARRAY, 169
CLASS_ATTR_SYM_VARSIZE, 169
CLASS_METHOD_ATTR_PARSE, 169
CLASS_METHOD_INTRODUCED, 169
CLASS_METHOD_OBSOLETE, 171
CLASS_METHOD_RENAMED, 171
CLASS_STICKY_ATTR, 171
CLASS_STICKY_ATTR_CLEAR, 171
CLASS_STICKY_METHOD, 172
CLASS_STICKY_METHOD_CLEAR, 172
e_max_attrflags, 183
OBJ_ATTR_ATOM, 172
OBJ_ATTR_ATOM_ARRAY, 173
OBJ_ATTR_CHAR, 173
OBJ_ATTR_CHAR_ARRAY, 173
OBJ_ATTR_DEFAULT, 173
OBJ_ATTR_DEFAULT_SAVE, 174
OBJ_ATTR_DOUBLE, 174
OBJ_ATTR_DOUBLE_ARRAY, 174
OBJ_ATTR_FLOAT, 174
OBJ_ATTR_FLOAT_ARRAY, 175
OBJ_ATTR_LONG, 175
OBJ_ATTR_LONG_ARRAY, 175
OBJ_ATTR_OBJ, 175
OBJ_ATTR_OBJ_ARRAY, 175
OBJ_ATTR_SAVE, 176
OBJ_ATTR_SYM, 176
OBJ_ATTR_SYM_ARRAY, 176
object_addattr, 191
object_attr_get, 191
object_attr_get_rect, 192
object_attr_getchar_array, 192
object_attr_getcolor, 192
object_attr_getdouble_array, 194
object_attr_getdump, 194
object_attr_getfloat, 194
object_attr_getfloat_array, 195
object_attr_getjrgba, 195
object_attr_getlong, 195
object_attr_getlong_array, 196
object_attr_getpt, 196
object_attr_getsize, 196
object_attr_getsym, 197
object_attr_getsym_array, 197
object_attr_method, 197
object_attr_set_rect, 198
object_attr_set_xywh, 198
object_attr_setchar_array, 198
object_attr_setcolor, 199
object_attr_setdouble_array, 199
object_attr_setfloat, 199
object_attr_setfloat_array, 200
object_attr_setjrgba, 200
object_attr_setlong, 200
object_attr_setlong_array, 201
object_attr_setparse, 201
object_attr_setpt, 201
object_attr_setsize, 202
object_attr_setsym, 202
object_attr_setsym_array, 202
object_attr_setvalueof, 203
object_attr_usercanget, 203
object_attr_usercanset, 203
object_chuckattr, 203
object_deleteattr, 204
object_new_parse, 204
STRUCT_ATTR_ATOM, 176
STRUCT_ATTR_ATOM_ARRAY, 177
STRUCT_ATTR_ATOM_LONG, 177
STRUCT_ATTR_ATOM_VARSIZE, 177
STRUCT_ATTR_CHAR, 177
STRUCT_ATTR_CHAR_ARRAY, 178
STRUCT_ATTR_CHAR_VARSIZE, 178
STRUCT_ATTR_DOUBLE, 178
STRUCT_ATTR_DOUBLE_ARRAY, 178
STRUCT_ATTR_DOUBLE_VARSIZE, 180
STRUCT_ATTR_FLOAT, 180
STRUCT_ATTR_FLOAT_ARRAY, 180
STRUCT_ATTR_FLOAT_VARSIZE, 180
STRUCT_ATTR_LONG, 181
STRUCT_ATTR_LONG_ARRAY, 181
STRUCT_ATTR_LONG_VARSIZE, 181
STRUCT_ATTR_OBJ, 182
STRUCT_ATTR_OBJ_ARRAY, 182
STRUCT_ATTR_OBJ_VARSIZE, 182
STRUCT_ATTR_SYM, 182
STRUCT_ATTR_SYM_ARRAY, 183
STRUCT_ATTR_SYM_VARSIZE, 183

b_dirty_clock
 _buffer, 923
BEGIN_USING_C_LINKAGE
 Miscellaneous, 397
BYTEORDER_SWAPF32
 Byte Ordering, 412
BYTEORDER_SWAPF64
 Byte Ordering, 412
BYTEORDER_SWAPW16
 Byte Ordering, 414
BYTEORDER_SWAPW32
 Byte Ordering, 414
BYTEORDER_SWAPW64
 Byte Ordering, 414
bangout
 Inlets and Outlets, 222
Binary Module, 686
 jit_bin_read_chunk_info, 686
 jit_bin_read_header, 687
 jit_bin_read_matrix, 687
 jit_bin_write_header, 688
 jit_bin_write_matrix, 688
binbuf_append
 Binbufs, 343
binbuf_eval
 Binbufs, 344

binbuf_getatom
 Binbufs, 344
binbuf_insert
 Binbufs, 344
binbuf_new
 Binbufs, 345
binbuf_set
 Binbufs, 345
binbuf_text
 Binbufs, 345
binbuf_totext
 Binbufs, 346
binbuf_vinsert
 Binbufs, 346
Binbufs, 343
 binbuf_append, 343
 binbuf_eval, 344
 binbuf_getatom, 344
 binbuf_insert, 344
 binbuf_new, 345
 binbuf_set, 345
 binbuf_text, 345
 binbuf_totext, 346
 binbuf_vinsert, 346
 readatom, 347
Box Layer, 646
 jbox_end_layer, 646
 jbox_invalidate_layer, 647
 jbox_paint_layer, 647
 jbox_start_layer, 647
buffer_getchannelcount
 Buffers, 442
buffer_getfilename
 Buffers, 443
buffer_getframecount
 Buffers, 443
buffer_getmillisamplerate
 Buffers, 443
buffer_getsamplerate
 Buffers, 443
buffer_locksamples
 Buffers, 444
buffer_ref_exists
 Buffers, 444
buffer_ref_getobject
 Buffers, 444
buffer_ref_new
 Buffers, 444
buffer_ref_notify
 Buffers, 445
buffer_ref_set
 Buffers, 445
buffer_setdirty
 Buffers, 445
buffer_setpadding
 Buffers, 445
buffer_unlocksamples
 Buffers, 446

buffer_view
 Buffers, 446
Buffers, 441
 buffer_getchannelcount, 442
 buffer_getfilename, 443
 buffer_getframecount, 443
 buffer_getmillisamplerate, 443
 buffer_getsamplerate, 443
 buffer_locksamples, 444
 buffer_ref_exists, 444
 buffer_ref_getobject, 444
 buffer_ref_new, 444
 buffer_ref_notify, 445
 buffer_ref_set, 445
 buffer_setdirty, 445
 buffer_setpadding, 445
 buffer_unlocksamples, 446
 buffer_view, 446
 t_buffer_obj, 442
 t_buffer_ref, 442

Byte Ordering, 412
 BYTEORDER_SWAPF32, 412
 BYTEORDER_SWAPF64, 412
 BYTEORDER_SWAPW16, 414
 BYTEORDER_SWAPW32, 414
 BYTEORDER_SWAPW64, 414

CLAMP
 Miscellaneous, 398

CLASS_ATTR_ACCESSORS
 Attributes, 153

CLASS_ATTR_ADD_FLAGS
 Attributes, 153

CLASS_ATTR_ALIAS
 Attributes, 154

CLASS_ATTR_ATOM
 Attributes, 154

CLASS_ATTR_ATOM_ARRAY
 Attributes, 154

CLASS_ATTR_ATOM_LONG
 Attributes, 154

CLASS_ATTR_ATOM_LONG_ARRAY
 Attributes, 155

CLASS_ATTR_ATOM_VARSIZE
 Attributes, 155

CLASS_ATTR_BASIC
 Attributes, 155

CLASS_ATTR_CATEGORY
 Attributes, 155

CLASS_ATTR_CHAR
 Attributes, 156

CLASS_ATTR_CHAR_ARRAY
 Attributes, 156

CLASS_ATTR_CHAR_VARSIZE
 Attributes, 156

CLASS_ATTR_DEFAULT
 Attributes, 156

CLASS_ATTR_DEFAULT_PAINT
 Attributes, 157

CLASS_ATTR_DEFAULT_SAVE
 Attributes, 157

CLASS_ATTR_DEFAULT_SAVE_PAINT
 Attributes, 157

CLASS_ATTR_DEFAULTNAME
 Attributes, 157

CLASS_ATTR_DEFAULTNAME_PAINT
 Attributes, 158

CLASS_ATTR_DEFAULTNAME_SAVE
 Attributes, 158

CLASS_ATTR_DEFAULTNAME_SAVE_PAINT
 Attributes, 158

CLASS_ATTR_DOUBLE
 Attributes, 159

CLASS_ATTR_DOUBLE_ARRAY
 Attributes, 159

CLASS_ATTR_DOUBLE_VARSIZE
 Attributes, 159

CLASS_ATTR_ENUM
 Attributes, 160

CLASS_ATTR_ENUMINDEX
 Attributes, 160

CLASS_ATTR_FILTER_CLIP
 Attributes, 160

CLASS_ATTR_FILTER_MAX
 Attributes, 161

CLASS_ATTR_FILTER_MIN
 Attributes, 161

CLASS_ATTR_FLOAT
 Attributes, 161

CLASS_ATTR_FLOAT_ARRAY
 Attributes, 162

CLASS_ATTR_FLOAT_VARSIZE
 Attributes, 162

CLASS_ATTR_INT32
 Attributes, 162

CLASS_ATTR_INTRODUCED
 Attributes, 162

CLASS_ATTR_INVISIBLE
 Attributes, 163

CLASS_ATTR_LABEL
 Attributes, 163

CLASS_ATTR_LEGACYDEFAULT
 Attributes, 163

CLASS_ATTR_LONG
 Attributes, 163

CLASS_ATTR_LONG_ARRAY
 Attributes, 164

CLASS_ATTR_LONG_VARSIZE
 Attributes, 164

CLASS_ATTR_MAX
 Attributes, 164

CLASS_ATTR_MIN
 Attributes, 164

CLASS_ATTR_OBJ
 Attributes, 165

CLASS_ATTR_OBJ_ARRAY
 Attributes, 165

CLASS_ATTR_OBJ_VARSIZE
 Attributes, 165

CLASS_ATTR_OBSOLETE
 Attributes, 166

CLASS_ATTR_ORDER
 Attributes, 166

CLASS_ATTR_PAINT
 Attributes, 166

CLASS_ATTR_REMOVE_FLAGS
 Attributes, 166

CLASS_ATTR_RENAMED
 Attributes, 167

CLASS_ATTR_RGBA
 Attributes, 167

CLASS_ATTR_SAVE
 Attributes, 167

CLASS_ATTR_SELFSAVE
 Attributes, 167

CLASS_ATTR_STYLE
 Attributes, 167

CLASS_ATTR_STYLE_ALIAS_COMPATIBILITY
 Styles, 621

CLASS_ATTR_STYLE_ALIAS_NOSAVE
 Styles, 621

CLASS_ATTR_STYLE_ALIAS_RGBA_LEGACY
 Styles, 621

CLASS_ATTR_STYLE_LABEL
 Attributes, 168

CLASS_ATTR_STYLE_RGBA
 Styles, 621

CLASS_ATTR_STYLE_RGBA_NOSAVE
 Styles, 622

CLASS_ATTR_STYLE_RGBA_PREVIEW
 Styles, 622

CLASS_ATTR_SYM
 Attributes, 168

CLASS_ATTR_SYM_ARRAY
 Attributes, 169

CLASS_ATTR_SYM_VARSIZE
 Attributes, 169

CLASS_BOX
 Classes, 207

CLASS_FLAG_ALIAS
 Classes, 207

CLASS_FLAG_BOX
 Classes, 207

CLASS_FLAG_DO_NOT_PARSE_ATTR_ARGS
 Classes, 207

CLASS_FLAG_DO_NOT_ZERO
 Classes, 207

CLASS_FLAG_NEWDICTIONARY
 Classes, 207

CLASS_FLAG_NOATTRIBUTES
 Classes, 207

CLASS_FLAG_OBJECT_METHOD
 Classes, 207

CLASS_FLAG_OWNATTRIBUTES
 Classes, 207

CLASS_FLAG_PARAMETER
 Classes, 207
CLASS_FLAG_POLYGLOT
 Classes, 207
CLASS_FLAG_REGISTERED
 Classes, 207
CLASS_FLAG RETYPEABLE
 Classes, 207
CLASS_FLAG_UIOBJECT
 Classes, 207
CLASS_METHOD_ATTR_PARSE
 Attributes, 169
CLASS_METHOD_INTRODUCED
 Attributes, 169
CLASS_METHOD_OBSOLETE
 Attributes, 171
CLASS_METHOD_RENAMED
 Attributes, 171
CLASS_STICKY_ATTR
 Attributes, 171
CLASS_STICKY_ATTR_CLEAR
 Attributes, 171
CLASS_STICKY_METHOD
 Attributes, 172
CLASS_STICKY_METHOD_CLEAR
 Attributes, 172
calcoffset
 Miscellaneous, 397
charset_convert
 Unicode, 652
charset_unicodeutf8
 Unicode, 652
charset_utf8_count
 Unicode, 652
charset_utf8_offset
 Unicode, 653
charset_utf8tounicode
 Unicode, 653
Class Module, 691
 class_copy, 692
 jit_class_addadornment, 692
 jit_class_addattr, 692
 jit_class_addinterface, 694
 jit_class_addmethod, 695
 jit_class_adddtypedwrapper, 696
 jit_class_adornment_get, 696
 jit_class_attr_get, 697
 jit_class_findbyname, 697
 jit_class_free, 697
 jit_class_mess, 698
 jit_class_method, 698
 jit_class_method_addargsafe, 698
 jit_class_method_argsafe_get, 699
 jit_class_nameget, 700
 jit_class_new, 700
 jit_class_register, 701
 jit_class_symcompare, 702
 jit_class_typedwrapper_get, 702
 class_addattr
 Classes, 207
 class_addmethod
 Classes, 207
 class_alias
 Classes, 208
 class_attr_setfill
 Styles, 623
 class_attr_setstyle
 Styles, 623
 class_attr_style_alias
 Styles, 623
 class_attr_stylemap
 Styles, 623
 class_copy
 Class Module, 692
 class_dspinit
 MSP, 438
 class_dspinitbbox
 MSP, 438
 class_dumpout_wrap
 Classes, 208
 class_findbyname
 Classes, 208
 class_findbyname_casemode
 Classes, 209
 class_free
 Classes, 209
 class_is_ui
 Classes, 209
 class_nameget
 Classes, 209
 class_new
 Classes, 211
 class_obexoffset_get
 Classes, 211
 class_obexoffset_set
 Classes, 211
 class_register
 Classes, 212
 class_setname
 Old-Style Classes, 217
 class_subclass
 Classes, 212
 class_super_construct
 Classes, 212
 class_time_addattr
 ITM Time Objects, 548
Classes, 205
 CLASS_BOX, 207
 CLASS_FLAG_ALIAS, 207
 CLASS_FLAG_BOX, 207
 CLASS_FLAG_DO_NOT_PARSE_ATTR_ARGS,
 207
 CLASS_FLAG_DO_NOT_ZERO, 207
 CLASS_FLAG_NEWDICTIONARY, 207
 CLASS_FLAG_NOATTRIBUTES, 207
 CLASS_FLAG_OBJECT_METHOD, 207

CLASS_FLAG_OWNATTRIBUTES, 207
CLASS_FLAG_PARAMETER, 207
CLASS_FLAG_POLYGLOT, 207
CLASS_FLAG_REGISTERED, 207
CLASS_FLAG RETYPEABLE, 207
CLASS_FLAG_UIOBJECT, 207
class_addattr, 207
class_addmethod, 207
class_alias, 208
class_dumpout_wrap, 208
class_findbyname, 208
class_findbyname_casemode, 209
class_free, 209
class_is_ui, 209
class_nameget, 209
class_new, 211
class_obexoffset_get, 211
class_obexoffset_set, 211
class_register, 212
class_subclass, 212
class_super_construct, 212
e_max_class_flags, 207
ext_main, 213
classname_openhelp
 Objects, 452
classname_openquery
 Objects, 452
classname_openrefpage
 Objects, 452
classname_openrefpage_ext
 Miscellaneous, 399
clock_delay
 Clocks, 532
clock_fdelay
 Clocks, 532
clock_gettime
 Clocks, 533
clock_new
 Clocks, 533
clock_unset
 Clocks, 533
Clocks, 529
 clock_delay, 532
 clock_fdelay, 532
 clock_gettime, 533
 clock_new, 533
 clock_unset, 533
 gettime, 533
 gettime_forobject, 534
 scheduler_fromobject, 534
 scheduler_get, 534
 scheduler_gettime, 534
 scheduler_new, 535
 scheduler_run, 535
 scheduler_set, 535
 scheduler_settime, 535
 scheduler_shift, 536
 setclock_delay, 536
setclock_fdelay, 536
setclock_gettime, 536
setclock_gettime, 538
setclock_unset, 538
systimer_gettime, 538
Colors, 617
 atoms_to_jrgba, 617
 jrgba_attr_get, 618
 jrgba_attr_set, 618
 jrgba_compare, 618
 jrgba_copy, 619
 jrgba_set, 619
 jrgba_to_atoms, 619
Console, 407
 cpost, 407
 error, 408
 object_error, 408
 object_error_obtrusive, 409
 object_post, 409
 object_warn, 409
 ouchstring, 410
 post, 410
 postatom, 411
cpost
 Console, 407
Critical Regions, 569
 critical_enter, 570
 critical_exit, 571
 critical_free, 571
 critical_new, 571
 critical_tryenter, 571
critical_enter
 Critical Regions, 570
critical_exit
 Critical Regions, 571
critical_free
 Critical Regions, 571
critical_new
 Critical Regions, 571
critical_tryenter
 Critical Regions, 571
Data Storage, 228
 e_max_datastore_flags, 229
 OBJ_FLAG_DATA, 230
 OBJ_FLAG_DEBUG, 230
 OBJ_FLAG_INHERITABLE, 230
 OBJ_FLAG_ITERATING, 230
 OBJ_FLAG_MEMORY, 230
 OBJ_FLAG_OBJ, 230
 OBJ_FLAG_REF, 230
 OBJ_FLAG_SILENT, 230
 t_cmpfn, 229
Data Types, 320
 DataView, 649
 jdataview_getclient, 649
 jdataview_new, 650
 jdataview_setclient, 650
Database, 239

db_close, 241
 db_open, 241
 db_open_ext, 241
 db_query, 242
 db_query_direct, 242
 db_query_getlastinsertid, 242
 db_query_silent, 243
 db_query_table_addcolumn, 243
 db_query_table_new, 243
 db_result_clear, 244
 db_result_datetimeminseconds, 244
 db_result_fieldname, 244
 db_result_float, 244
 db_result_long, 245
 db_result_nextrecord, 245
 db_result_numfields, 245
 db_result_numrecords, 245
 db_result_reset, 246
 db_result_string, 246
 db_transaction_end, 246
 db_transaction_flush, 246
 db_transaction_start, 246
 db_util_datetostring, 247
 db_util_stringtodata, 247
 db_view_create, 247
 db_view_getresult, 247
 db_view_remove, 249
 db_view_setquery, 249
 t_database, 240
 t_db_result, 240
 t_db_view, 241
 db_close
 Database, 241
 db_open
 Database, 241
 db_open_ext
 Database, 241
 db_query
 Database, 242
 db_query_direct
 Database, 242
 db_query_getlastinsertid
 Database, 242
 db_query_silent
 Database, 243
 db_query_table_addcolumn
 Database, 243
 db_query_table_new
 Database, 243
 db_result_clear
 Database, 244
 db_result_datetimeminseconds
 Database, 244
 db_result_fieldname
 Database, 244
 db_result_float
 Database, 244
 db_result_long
 Database, 245
 db_result_nextrecord
 Database, 245
 db_result_numfields
 Database, 245
 db_result_numrecords
 Database, 245
 db_result_reset
 Database, 246
 db_result_string
 Database, 246
 db_transaction_end
 Database, 246
 db_transaction_flush
 Database, 246
 db_transaction_start
 Database, 246
 db_util_datetostring
 Database, 247
 db_util_stringtodata
 Database, 247
 db_view_create
 Database, 247
 db_view_getresult
 Database, 247
 db_view_remove
 Database, 249
 db_view_setquery
 Database, 249
 defer
 Threads, 561
 defer_low
 Threads, 561
 Dictionary, 250
 dictionary_appendatom, 255
 dictionary_appendatomarray, 255
 dictionary_appendatoms, 255
 dictionary_appenddictionary, 256
 dictionary_appendfloat, 256
 dictionary_appendlong, 256
 dictionary_appendobject, 257
 dictionary_appendstring, 257
 dictionary_appendsym, 257
 dictionary_chuckentry, 258
 dictionary_clear, 258
 dictionary_copyatoms, 258
 dictionary_copydefatoms, 259
 dictionary_copyentries, 259
 dictionary_copyunique, 259
 dictionary_deleteentry, 260
 dictionary_dump, 260
 dictionary_entry_getkey, 260
 dictionary_entry_getvalue, 261
 dictionary_entry_getvalues, 261
 dictionary_entryisatomarray, 261
 dictionary_entryisdictionary, 261
 dictionary_entryisstring, 262
 dictionary_freekeys, 262

dictionary_funall, 262
dictionary_get_ex, 263
dictionary_getatom, 263
dictionary_getatomarray, 263
dictionary_getatoms, 264
dictionary_getatoms_ext, 264
dictionary_getdefatom, 264
dictionary_getdefatoms, 265
dictionary_getdeffloat, 265
dictionary_getdeflong, 265
dictionary_getdefstring, 266
dictionary_getdefsym, 266
dictionary_getdictionary, 267
dictionary_getentrycount, 267
dictionary_getfloat, 267
dictionary_getkeys, 267
dictionary_getlong, 268
dictionary_getobject, 268
dictionary_getstring, 268
dictionary_getsym, 269
dictionary_hasentry, 269
dictionary_new, 269
dictionary_read, 269
dictionary_read_yaml, 270
dictionary_sprintf, 270
dictionary_write, 271
dictionary_write_yaml, 271
postdictionary, 271
Dictionary Passing API, 310
dictobj_atom_safety, 312
dictobj_atom_safety_flags, 313
dictobj_dictionaryfromatoms, 313
dictobj_dictionaryfromatoms_extended, 313
dictobj_dictionaryfromstring, 314
dictobj_dictionarytoatoms, 314
dictobj_findregistered_clone, 314
dictobj_findregistered_retain, 316
dictobj_jsonfromstring, 316
dictobj_key_parse, 316
dictobj_namefromptr, 317
dictobj_outlet_atoms, 317
dictobj_register, 317
dictobj_release, 318
dictobj_unregister, 318
dictobj_validate, 318
dictionary_appendatom
 Dictionary, 255
dictionary_appendatomarray
 Dictionary, 255
dictionary_appendatoms
 Dictionary, 255
dictionary_appenddictionary
 Dictionary, 256
dictionary_appendfloat
 Dictionary, 256
dictionary_appendlong
 Dictionary, 256
dictionary_appendobject
 Dictionary, 257
dictionary_appendstring
 Dictionary, 257
dictionary_appendsym
 Dictionary, 257
dictionary_chuckentry
 Dictionary, 258
dictionary_clear
 Dictionary, 258
dictionary_copyatoms
 Dictionary, 258
dictionary_copydefatoms
 Dictionary, 259
dictionary_copyentries
 Dictionary, 259
dictionary_copyunique
 Dictionary, 259
dictionary_deleteentry
 Dictionary, 260
dictionary_dump
 Dictionary, 260
dictionary_entry_getkey
 Dictionary, 260
dictionary_entry_getvalue
 Dictionary, 261
dictionary_entry_getvalues
 Dictionary, 261
dictionary_entryisatomarray
 Dictionary, 261
dictionary_entryisdictionary
 Dictionary, 261
dictionary_entryisstring
 Dictionary, 262
dictionary_freekeys
 Dictionary, 262
dictionary_funall
 Dictionary, 262
dictionary_get_ex
 Dictionary, 263
dictionary_getatom
 Dictionary, 263
dictionary_getatomarray
 Dictionary, 263
dictionary_getatoms
 Dictionary, 264
dictionary_getatoms_ext
 Dictionary, 264
dictionary_getdefatom
 Dictionary, 264
dictionary_getdefatoms
 Dictionary, 265
dictionary_getdeffloat
 Dictionary, 265
dictionary_getdeflong
 Dictionary, 265
dictionary_getdefstring
 Dictionary, 266
dictionary_getdefsym
 Dictionary, 266

Dictionary, 266
dictionary_getdictionary
 Dictionary, 267
dictionary_getentrycount
 Dictionary, 267
dictionary_getfloat
 Dictionary, 267
dictionary_getkeys
 Dictionary, 267
dictionary_getlong
 Dictionary, 268
dictionary_getobject
 Dictionary, 268
dictionary_getstring
 Dictionary, 268
dictionary_getsym
 Dictionary, 269
dictionary_hasentry
 Dictionary, 269
dictionary_new
 Dictionary, 269
dictionary_read
 Dictionary, 269
dictionary_read_yaml
 Dictionary, 270
dictionary_sprintf
 Dictionary, 270
dictionary_write
 Dictionary, 271
dictionary_write_yaml
 Dictionary, 271
dictobj_atom_safety
 Dictionary Passing API, 312
dictobj_atom_safety_flags
 Dictionary Passing API, 313
dictobj_dictionaryfromatoms
 Dictionary Passing API, 313
dictobj_dictionaryfromatoms_extended
 Dictionary Passing API, 313
dictobj_dictionaryfromstring
 Dictionary Passing API, 314
dictobj_dictionarytoatoms
 Dictionary Passing API, 314
dictobj_findregistered_clone
 Dictionary Passing API, 314
dictobj_findregistered_retain
 Dictionary Passing API, 316
dictobj_jsonfromstring
 Dictionary Passing API, 316
dictobj_key_parse
 Dictionary Passing API, 316
dictobj_namefromptr
 Dictionary Passing API, 317
dictobj_outlet_atoms
 Dictionary Passing API, 317
dictobj_register
 Dictionary Passing API, 317
dictobj_release

Dictionary Passing API, 318
dictobj_unregister
 Dictionary Passing API, 318
dictobj_validate
 Dictionary Passing API, 318
disposhandle
 Memory Management, 384
dsp_add
 MSP, 438
dsp_advv
 MSP, 439
e_max_atom_gettext_flags
 Atoms, 324
e_max_atomtypes
 Atoms, 324
e_max_attrflags
 Attributes, 183
e_max_class_flags
 Classes, 207
e_max_datastore_flags
 Data Storage, 229
e_max_dateflags
 Systime API, 543
e_max_errorcodes
 Miscellaneous, 399
e_max_expr_types
 Extending expr, 415
e_max_fileinfo_flags
 Files and Folders, 356
e_max_openfile_permissions
 Files and Folders, 356
e_max_path_folder_flags
 Files and Folders, 356
e_max_path_styles
 Files and Folders, 356
e_max_path_types
 Files and Folders, 356
e_max_systhread_mutex_flags
 Threads, 561
e_max_wind_advise_result
 Miscellaneous, 399
eAltKey
 Mouse and Keyboard, 432
eAutoRepeat
 Mouse and Keyboard, 432
eCapsLock
 Mouse and Keyboard, 432
eCommandKey
 Mouse and Keyboard, 432
eControlKey
 Mouse and Keyboard, 432
eLeftButton
 Mouse and Keyboard, 432
eMiddleButton
 Mouse and Keyboard, 432
ePopupMenu
 Mouse and Keyboard, 432
eRightButton

Mouse and Keyboard, 432
eShiftKey
 Mouse and Keyboard, 432
ET_FI
 Extending expr, 416
ET_FLT
 Extending expr, 416
ET_FUNC
 Extending expr, 416
ET_II
 Extending expr, 416
ET_INT
 Extending expr, 416
ET_LB
 Extending expr, 416
ET_LP
 Extending expr, 416
ET_OP
 Extending expr, 416
ET_SI
 Extending expr, 416
ET_STR
 Extending expr, 416
ET_SYM
 Extending expr, 416
ET_TBL
 Extending expr, 416
ET_VSYM
 Extending expr, 416
egetfn
 Old-Style Classes, 217
error
 Console, 408
error_subscribe
 Miscellaneous, 400
error_sym
 Miscellaneous, 400
error_unsubscribe
 Miscellaneous, 400
expr_eval
 Extending expr, 416
expr_new
 Extending expr, 416
ext_main
 Classes, 213
Extending expr, 415
 e_max_expr_types, 415
ET_FI, 416
ET_FLT, 416
ET_FUNC, 416
ET_II, 416
ET_INT, 416
ET_LB, 416
ET_LP, 416
ET_OP, 416
ET_SI, 416
ET_STR, 416
ET_SYM, 416
ET_TBL, 416
ET_VSYM, 416
expr_eval, 416
expr_new, 416
FILL_ATTR_SAVE
 Styles, 622
fileload
 Loading Max Files, 425
Files and Folders, 351
 e_max_fileinfo_flags, 356
 e_max_openfile_permissions, 356
 e_max_path_folder_flags, 356
 e_max_path_styles, 356
 e_max_path_types, 356
 fileusage_addfile, 357
 fileusage_addpackage, 358
 filewatcher_new, 358
 locatefile, 358
 locatefile_extended, 359
 locatefiletype, 360
 MAX_FILENAME_CHARS, 355
 open_dialog, 360
 open_promptset, 360
 PATH_FILEINFO_ALIAS, 356
 PATH_FILEINFO_FOLDER, 356
 PATH_FILEINFO_PACKAGE, 356
 PATH_FOLDER_SNIF, 356
 PATH_NOALIASRESOLUTION, 356
 PATH_READ_PERM, 356
 PATH_REPORTPACKAGEASFOLDER, 356
 PATH_RW_PERM, 356
 PATH_STYLE_COLON, 356
 PATH_STYLE_MAX, 356
 PATH_STYLE_NATIVE, 356
 PATH_STYLE_NATIVE_WIN, 356
 PATH_STYLE_SLASH, 356
 PATH_TYPE_ABSOLUTE, 357
 PATH_TYPE_BOOT, 357
 PATH_TYPE_C74, 357
 PATH_TYPE_DESKTOP, 357
 PATH_TYPE_IGNORE, 357
 PATH_TYPE_MAXDB, 357
 PATH_TYPE_PATH, 357
 PATH_TYPE_RELATIVE, 357
 PATH_TYPE_TEMPFOLDER, 357
 PATH_TYPE_TILDE, 357
 PATH_WRITE_PERM, 356
path_absolutepath, 361
path_closefolder, 361
path_createsysfile, 361
path_fileinfo, 362
path_folernextfile, 362
path_frompathname, 362
path_getapppath, 363
path_getdefault, 363
path_getfilemoddate, 363
path_getmoddate, 363
path_nameconform, 363

path_openfolder, 364
 path_opensysfile, 364
 path_resolvefile, 364
 path_setdefault, 365
 path_toabsolutesystempath, 365
 path_topathname, 365
 path_topotentialname, 366
 SYSFILE_ATMARK, 357
 SYSFILE_FROMEOF, 357
 SYSFILE_FROMMARK, 357
 SYSFILE_FROMSTART, 357
 saveas_dialog, 366
 saveas_promptset, 366
 saveasdialog_extended, 367
 sysfile_close, 368
 sysfile_geteof, 368
 sysfile_getpos, 368
 sysfile_openhandle, 369
 sysfile_openptrsize, 369
 sysfile_read, 369
 sysfile_readtextfile, 370
 sysfile_readtohandle, 370
 sysfile_readtopr, 370
 sysfile_seteof, 371
 sysfile_setpos, 371
 sysfile_spoolcopy, 371
 sysfile_write, 372
 sysfile_writetextfile, 372
 t_filehandle, 355
 t_sysfile_pos_mode, 357
 t_sysfile_text_flags, 357
 TEXT_ENCODING_USE_FILE, 357
 TEXT_LB_MAC, 357
 TEXT_LB_NATIVE, 357
 TEXT_LB_PC, 357
 TEXT_LB_UNIX, 357
 TEXT_NULL_TERMINATE, 357
 fileusage_addfile
 Files and Folders, 357
 fileusage_addpackage
 Files and Folders, 358
 filewatcher_new
 Files and Folders, 358
 floatin
 Inlets and Outlets, 222
 floatout
 Inlets and Outlets, 222
 freebytes
 Memory Management, 384
 freebytes16
 Memory Management, 386
 freeobject
 Old-Style Classes, 218
 gensym
 Symbols, 349
 gensym_tr
 Symbols, 350
 getbytes
 Memory Management, 386
 getbytes16
 Memory Management, 386
 getfn
 Old-Style Classes, 218
 gettime
 Clocks, 533
 gettime_forobject
 Clocks, 534
 globalsymbol_bind
 Miscellaneous, 400
 globalsymbol_dereference
 Miscellaneous, 401
 globalsymbol_reference
 Miscellaneous, 401
 globalsymbol_unbind
 Miscellaneous, 401
 growhandle
 Memory Management, 386
 HASH_DEFSLOTS
 Hash Table, 274
 Hash Table, 272
 HASH_DEFSLOTS, 274
 hashtab_chuck, 274
 hashtab_chuckkey, 274
 hashtab_clear, 275
 hashtab_delete, 275
 hashtab_findfirst, 275
 hashtab_flags, 276
 hashtab_funall, 276
 hashtab_getflags, 276
 hashtab_getkeyflags, 277
 hashtab_getkeys, 277
 hashtab_getsize, 277
 hashtab_keyflags, 278
 hashtab_lookup, 278
 hashtab_lookupflags, 278
 hashtab_lookuplong, 279
 hashtab_lookupsym, 279
 hashtab_methodall, 279
 hashtab_new, 280
 hashtab_print, 280
 hashtab_READONLY, 280
 hashtab_store, 281
 hashtab_STORE_SAFE, 281
 hashtab_STOREFLAGS, 281
 hashtab_STORELONG, 282
 hashtab_STORESYM, 282
 hashtab_chuck
 Hash Table, 274
 hashtab_chuckkey
 Hash Table, 274
 hashtab_clear
 Hash Table, 275
 hashtab_delete
 Hash Table, 275
 hashtab_findfirst
 Hash Table, 275

hashtab_flags
 Hash Table, 276
hashtab_funall
 Hash Table, 276
hashtab_getflags
 Hash Table, 276
hashtab_getkeyflags
 Hash Table, 277
hashtab_getkeys
 Hash Table, 277
hashtab_getsize
 Hash Table, 277
hashtab_keyflags
 Hash Table, 278
hashtab_lookup
 Hash Table, 278
hashtab_lookupflags
 Hash Table, 278
hashtab_lookuplong
 Hash Table, 279
hashtab_lookupsym
 Hash Table, 279
hashtab_methodall
 Hash Table, 279
hashtab_new
 Hash Table, 280
hashtab_print
 Hash Table, 280
hashtab_READONLY
 Hash Table, 280
hashtab_store
 Hash Table, 281
hashtab_STORE_SAFE
 Hash Table, 281
hashtab_STOREFLAGS
 Hash Table, 281
hashtab_STORELONG
 Hash Table, 282
hashtab_STORESYM
 Hash Table, 282
HitBox
 jbox, 498
HitGrowBox
 jbox, 498
HitInlet
 jbox, 498
HitLine
 jbox, 498
HitLineLocked
 jbox, 498
HitNothing
 jbox, 498
HitOutlet
 jbox, 498
HitTestResult
 jbox, 498

INRANGE
 Miscellaneous, 398

ITM Time Objects, 546
 class_time_addattr, 548
 itm_barbeatunitstoticks, 549
 itm_dereference, 549
 itm_dump, 549
 itm_getglobal, 549
 itm_getname, 549
 itm_getnamed, 550
 itm_getresolution, 550
 itm_getstate, 550
 itm_getticks, 550
 itm_gettime, 551
 itm_gettimesignature, 551
 itm_isunitfixed, 551
 itm_mstosamps, 551
 itm_mstoticks, 552
 itm_pause, 552
 itm_reference, 552
 itm_resume, 552
 itm_sampstoms, 552
 itm_setresolution, 553
 itm_settimesignature, 553
 itm_tickstobarbeatunits, 553
 itm_ticksstoms, 553
 t_itm, 558
 t_timeobject, 558
TIME_FLAGS_BBUSOURCE, 548
TIME_FLAGS_CHECKSCHEDULE, 548
TIME_FLAGS_EVENTLIST, 548
TIME_FLAGS_FIXED, 548
TIME_FLAGS_FIXEDONLY, 548
TIME_FLAGS_LISTENTICKS, 548
TIME_FLAGS_LOCATION, 548
TIME_FLAGS_LOOKAHEAD, 548
TIME_FLAGS_NOUNITS, 548
TIME_FLAGS_PERMANENT, 548
TIME_FLAGS_POSITIVE, 548
TIME_FLAGS_TICKSONLY, 548
TIME_FLAGS_TRANSPORT, 548
TIME_FLAGS_USECLOCK, 548
TIME_FLAGS_USEQELEM, 548
time_calcquantize, 554
time_getitm, 554
time_getms, 554
time_getnamed, 554
time_getphase, 555
time_getticks, 555
time_isfixedunit, 555
time_listen, 555
time_new, 556
time_now, 556
time_schedule, 556
time_schedule_limit, 556
time_setclock, 556
time_setvalue, 558
time_stop, 558
time_tick, 558

Index Map, 283

indexmap_append, 284
 indexmap_clear, 284
 indexmap_datafromindex, 284
 indexmap_delete, 284
 indexmap_delete_index, 284
 indexmap_delete_index_multi, 286
 indexmap_delete_multi, 286
 indexmap_getsize, 286
 indexmap_indexfromdata, 286
 indexmap_move, 287
 indexmap_new, 287
 indexmap_sort, 287
 indexmap_append
 Index Map, 284
 indexmap_clear
 Index Map, 284
 indexmap_datafromindex
 Index Map, 284
 indexmap_delete
 Index Map, 284
 indexmap_delete_index
 Index Map, 284
 indexmap_delete_index_multi
 Index Map, 286
 indexmap_delete_multi
 Index Map, 286
 indexmap_getsize
 Index Map, 286
 indexmap_indexfromdata
 Index Map, 286
 indexmap_move
 Index Map, 287
 indexmap_new
 Index Map, 287
 indexmap_sort
 Index Map, 287
 inlet_new
 Inlets and Outlets, 222
 Inlets and Outlets, 221
 bangout, 222
 floatin, 222
 floatout, 222
 inlet_new, 222
 intin, 223
 intout, 223
 listout, 223
 outlet_anything, 224
 outlet_bang, 224
 outlet_float, 225
 outlet_int, 225
 outlet_list, 225
 outlet_new, 226
 proxy_getinlet, 226
 proxy_new, 226
 intin
 Inlets and Outlets, 223
 intload
 Loading Max Files, 425
 intout
 Inlets and Outlets, 223
 isr
 Threads, 563
 itm_barbeatunitstoticks
 ITM Time Objects, 549
 itm_dereference
 ITM Time Objects, 549
 itm_dump
 ITM Time Objects, 549
 itm_getglobal
 ITM Time Objects, 549
 itm_getname
 ITM Time Objects, 549
 itm_getnamed
 ITM Time Objects, 550
 itm_getresolution
 ITM Time Objects, 550
 itm_getstate
 ITM Time Objects, 550
 itm_getticks
 ITM Time Objects, 550
 itm_gettime
 ITM Time Objects, 551
 itm_gettimesignature
 ITM Time Objects, 551
 itm_isunitfixed
 ITM Time Objects, 551
 itm_mstosamps
 ITM Time Objects, 551
 itm_mstoticks
 ITM Time Objects, 552
 itm_pause
 ITM Time Objects, 552
 itm_reference
 ITM Time Objects, 552
 itm_resume
 ITM Time Objects, 552
 itm_sampstoms
 ITM Time Objects, 552
 itm_setresolution
 ITM Time Objects, 553
 itm_settimesignature
 ITM Time Objects, 553
 itm_ticksbarbeatunits
 ITM Time Objects, 553
 itm_tickstoms
 ITM Time Objects, 553
 JBOX_FONTFACE_BOLD
 jbox, 498
 JBOX_FONTFACE_BOLDITALIC
 jbox, 498
 JBOX_FONTFACE_ITALIC
 jbox, 498
 JBOX_FONTFACE_REGULAR
 jbox, 498
 JBOX_NOINSPECTFIRSTIN
 jbox, 498

JFont, 604
JGRAPHICS_FONT_SLANT_ITALIC, 605
JGRAPHICS_FONT_SLANT_NORMAL, 605
JGRAPHICS_FONT_WEIGHT_BOLD, 605
JGRAPHICS_FONT_WEIGHT_NORMAL, 605
jbox_get_font_slant, 606
jbox_get_font_weight, 606
jfont_create, 606
jfont_destroy, 606
jfont_extents, 606
jfont_get_em_dimensions, 607
jfont_get_family, 607
jfont_get_font_size, 607
jfont_get_slant, 607
jfont_get_underline, 607
jfont_get_weight, 608
jfont_getfontlist, 608
jfont_isequalto, 608
jfont_reference, 608
jfont_set_family, 609
jfont_set_font_size, 609
jfont_set_slant, 609
jfont_set_underline, 609
jfont_set_weight, 609
jfont_text_measure, 610
jfont_text_measuretext_wrapped, 610
systemfontname, 610
systemfontname_bold, 610
systemfontname_light, 610
systemfontsym, 610
t_jgraphics_font_slant, 605
t_jgraphics_font_weight, 605
JGRAPHICS_2PI
JGraphics, 580
JGRAPHICS_3PIOVER2
JGraphics, 580
JGRAPHICS_FILEFORMAT_JPEG
JGraphics, 581
JGRAPHICS_FILEFORMAT_PNG
JGraphics, 581
JGRAPHICS_FONT_SLANT_ITALIC
JFont, 605
JGRAPHICS_FONT_SLANT_NORMAL
JFont, 605
JGRAPHICS_FONT_WEIGHT_BOLD
JFont, 605
JGRAPHICS_FONT_WEIGHT_NORMAL
JFont, 605
JGRAPHICS_FORMAT_A8
JGraphics, 581
JGRAPHICS_FORMAT_ARGB32
JGraphics, 581
JGRAPHICS_FORMAT_RGB24
JGraphics, 581
JGRAPHICS_PI
JGraphics, 580
JGRAPHICS_TEXT_JUSTIFICATION_BOTTOM
JGraphics, 581
JGRAPHICS_TEXT_JUSTIFICATION_CENTERED
JGraphics, 581
JGRAPHICS_TEXT_JUSTIFICATION_HCENTERED
JGraphics, 581
JGRAPHICS_TEXT_JUSTIFICATION_HJUSTIFIED
JGraphics, 581
JGRAPHICS_TEXT_JUSTIFICATION_LEFT
JGraphics, 581
JGRAPHICS_TEXT_JUSTIFICATION_RIGHT
JGraphics, 581
JGRAPHICS_TEXT_JUSTIFICATION_TOP
JGraphics, 581
JGRAPHICS_TEXT_JUSTIFICATION_VCENTERED
JGraphics, 581
JGRAPHICS_TEXTLAYOUT_NOWRAP
TextLayout, 637
JGRAPHICS_TEXTLAYOUT_USEELLIPSIS
TextLayout, 637
JGraphics, 576
JGRAPHICS_2PI, 580
JGRAPHICS_3PIOVER2, 580
JGRAPHICS_FILEFORMAT_JPEG, 581
JGRAPHICS_FILEFORMAT_PNG, 581
JGRAPHICS_FORMAT_A8, 581
JGRAPHICS_FORMAT_ARGB32, 581
JGRAPHICS_FORMAT_RGB24, 581
JGRAPHICS_PI, 580
JGRAPHICS PIOVER2, 580
JGRAPHICS_TEXT_JUSTIFICATION_BOTTOM,
581
JGRAPHICS_TEXT_JUSTIFICATION_CENTER↔
ED, 581
JGRAPHICS_TEXT_JUSTIFICATION_HCENTE↔
RED, 581
JGRAPHICS_TEXT_JUSTIFICATION_HJUSTIF↔
IED, 581
JGRAPHICS_TEXT_JUSTIFICATION_LEFT, 581
JGRAPHICS_TEXT_JUSTIFICATION_RIGH↔
T, 581
JGRAPHICS_TEXT_JUSTIFICATION_TOP, 581
JGRAPHICS_TEXT_JUSTIFICATION_VCENTE↔
RED, 581
jgraphics_append_path, 581
jgraphics_arc, 581
jgraphics_arc_negative, 582
jgraphics_bubble, 582
jgraphics_clip, 582
jgraphics_close_path, 582
jgraphics_copy_path, 583
jgraphics_curve_to, 583
jgraphics_destroy, 583
jgraphics_device_to_user, 583
jgraphics_ellipse, 583
jgraphics_font_extents, 584
jgraphics_get_current_point, 584
jgraphics_getfiletypes, 584

jgraphics_line_to, 585
jgraphics_move_to, 585
jgraphics_new_path, 585
jgraphics_oval, 585
jgraphics_ovalarc, 585
jgraphics_path_contains, 586
jgraphics_path_createstroked, 586
jgraphics_path_destroy, 586
jgraphics_path_getlength, 586
jgraphics_path_getnearestpoint, 586
jgraphics_path_getpathelems, 588
jgraphics_path_getpointalongpath, 588
jgraphics_path_intersectsline, 588
jgraphics_path_roundcorners, 588
jgraphics_position_one_rect_near_another_rect←
 _but_keep_inside_a_third_rect, 589
jgraphics_rectangle, 589
jgraphics_rectangle_rounded, 589
jgraphics_rectcontainsrect, 589
jgraphics_rectintersectsrect, 590
jgraphics_reference, 590
jgraphics_rel_curve_to, 590
jgraphics_rel_line_to, 590
jgraphics_rel_move_to, 591
jgraphics_round, 591
jgraphics_select_font_face, 591
jgraphics_select_jfont, 591
jgraphics_set_font_size, 591
jgraphics_set_underline, 592
jgraphics_show_text, 592
jgraphics_system_canonicaltexttotransparentbg,
 592
jgraphics_text_measure, 592
jgraphics_text_measuretext_wrapped, 592
jgraphics_text_path, 593
jgraphics_triangle, 593
jgraphics_user_to_device, 593
t_jgraphics_fileformat, 581
t_jgraphics_format, 581
t_jgraphics_text_justification, 581
JGraphics Matrix Transformations, 612
jgraphics_matrix_init, 613
jgraphics_matrix_init_identity, 613
jgraphics_matrix_init_rotate, 613
jgraphics_matrix_init_scale, 613
jgraphics_matrix_init_translate, 613
jgraphics_matrix_invert, 614
jgraphics_matrix_multiply, 614
jgraphics_matrix_rotate, 614
jgraphics_matrix_scale, 614
jgraphics_matrix_transform_point, 614
jgraphics_matrix_translate, 615
JMOUSE_CURSOR_ARROW
 Mouse and Keyboard, 432
JMOUSE_CURSOR_COPYING
 Mouse and Keyboard, 432
JMOUSE_CURSOR_CROSSHAIR
 Mouse and Keyboard, 432
JMOUSE_CURSOR_DRAGGINGHAND
 Mouse and Keyboard, 432
JMOUSE_CURSOR_IBEAM
 Mouse and Keyboard, 432
JMOUSE_CURSOR_NONE
 Mouse and Keyboard, 432
JMOUSE_CURSOR_POINTINGHAND
 Mouse and Keyboard, 432
JMOUSE_CURSOR_RESIZE_BOTTOMEDGE
 Mouse and Keyboard, 432
JMOUSE_CURSOR_RESIZE_BOTTOMLEFTCORNER←
 ER
 Mouse and Keyboard, 432
JMOUSE_CURSOR_RESIZE_BOTTOMRIGHTCORNER←
 NR
 Mouse and Keyboard, 432
JMOUSE_CURSOR_RESIZE_FOURWAY
 Mouse and Keyboard, 432
JMOUSE_CURSOR_RESIZE_LEFTEDGE
 Mouse and Keyboard, 432
JMOUSE_CURSOR_RESIZE_LEFTRIGHT
 Mouse and Keyboard, 432
JMOUSE_CURSOR_RESIZE_RIGHTEDGE
 Mouse and Keyboard, 432
JMOUSE_CURSOR_RESIZE_TOPEDGE
 Mouse and Keyboard, 432
JMOUSE_CURSOR_RESIZE_TOPLEFTCORNER
 Mouse and Keyboard, 432
JMOUSE_CURSOR_RESIZE_TOPRIGHTCORNER
 Mouse and Keyboard, 432
JMOUSE_CURSOR_RESIZE_UPDOWN
 Mouse and Keyboard, 432
JMOUSE_CURSOR_WAIT
 Mouse and Keyboard, 432
JPattern, 616
JSurface, 594
 jgraphics_create, 595
 jgraphics_get_resource_data, 595
 jgraphics_image_surface_clear, 596
 jgraphics_image_surface_create, 596
 jgraphics_image_surface_create_for_data, 596
 jgraphics_image_surface_create_from_file, 597
 jgraphics_image_surface_create_from_filedata,
 597
 jgraphics_image_surface_create_from_resource,
 597
 jgraphics_image_surface_create_referenced, 598
 jgraphics_image_surface_draw, 598
 jgraphics_image_surface_draw_fast, 598
 jgraphics_image_surface_get_height, 599
 jgraphics_image_surface_get_pixel, 599
 jgraphics_image_surface_get_width, 599
 jgraphics_image_surface_scroll, 599
 jgraphics_image_surface_set_pixel, 600
 jgraphics_image_surface_writejpeg, 600
 jgraphics_image_surface_writepng, 600
 jgraphics_surface_destroy, 600
 jgraphics_surface_reference, 601

jgraphics_write_image_surface_to_filedata, 601
jbox, 494
 HitBox, 498
 HitGrowBox, 498
 HitInlet, 498
 HitLine, 498
 HitLineLocked, 498
 HitNothing, 498
 HitOutlet, 498
 HitTestResult, 498
 JBOX_FONTFACE_BOLD, 498
 JBOX_FONTFACE_BOLDITALIC, 498
 JBOX_FONTFACE_ITALIC, 498
 JBOX_FONTFACE_REGULAR, 498
 JBOX_NOINSPECTFIRSTIN, 498
 jbox_free, 498
 jbox_get_annotation, 499
 jbox_get_background, 499
 jbox_get_canhilite, 499
 jbox_get_color, 499
 jbox_get_drawfirstin, 499
 jbox_get_drawinlast, 500
 jbox_get_fontname, 500
 jbox_get fontsize, 500
 jbox_get_growboth, 500
 jbox_get_growy, 500
 jbox_get_hidden, 501
 jbox_get_hint, 501
 jbox_get_hintstring, 501
 jbox_get_id, 501
 jbox_get_ignoreclick, 501
 jbox_get_maxclass, 503
 jbox_get_nextobject, 503
 jbox_get_nogrow, 503
 jbox_get_object, 503
 jbox_get_outline, 503
 jbox_get_patcher, 505
 jbox_get_patching_position, 505
 jbox_get_patching_rect, 505
 jbox_get_patching_size, 505
 jbox_get_presentation, 505
 jbox_get_presentation_position, 507
 jbox_get_presentation_rect, 507
 jbox_get_presentation_size, 507
 jbox_get_prevobject, 507
 jbox_get_rect_for_sym, 508
 jbox_get_rect_for_view, 508
 jbox_get_textfield, 508
 jbox_get_varname, 508
 jbox_new, 509
 jbox_notify, 509
 jbox_ready, 509
 jbox_redraw, 509
 jbox_set_annotation, 510
 jbox_set_background, 510
 jbox_set_color, 510
 jbox_set_fontname, 510
 jbox_set fontsize, 510
 jbox_set_hidden, 512
 jbox_set_hint, 512
 jbox_set_hintstring, 512
 jbox_set_ignoreclick, 512
 jbox_set_outline, 513
 jbox_set_patching_position, 513
 jbox_set_patching_rect, 513
 jbox_set_patching_size, 513
 jbox_set_position, 514
 jbox_set_presentation, 514
 jbox_set_presentation_position, 514
 jbox_set_presentation_rect, 514
 jbox_set_presentation_size, 515
 jbox_set_rect, 515
 jbox_set_rect_for_sym, 515
 jbox_set_rect_for_view, 515
 jbox_set_size, 516
 jbox_set_varname, 516
jbox_end_layer
 Box Layer, 646
jbox_free
 jbox, 498
jbox_get_annotation
 jbox, 499
jbox_get_background
 jbox, 499
jbox_get_canhilite
 jbox, 499
jbox_get_color
 jbox, 499
jbox_get_drawfirstin
 jbox, 499
jbox_get_drawinlast
 jbox, 500
jbox_get_font_slant
 JFont, 606
jbox_get_font_weight
 JFont, 606
jbox_get_fontname
 jbox, 500
jbox_get fontsize
 jbox, 500
jbox_get_growboth
 jbox, 500
jbox_get_growy
 jbox, 500
jbox_get_hidden
 jbox, 501
jbox_get_hint
 jbox, 501
jbox_get_hintstring
 jbox, 501
jbox_get_id
 jbox, 501
jbox_get_ignoreclick
 jbox, 501
jbox_get_maxclass
 jbox, 503

jbox_get_nextobject
 jbox, 503
jbox_get_nogrow
 jbox, 503
jbox_get_object
 jbox, 503
jbox_get_outline
 jbox, 503
jbox_get_patcher
 jbox, 505
jbox_get_patching_position
 jbox, 505
jbox_get_patching_rect
 jbox, 505
jbox_get_patching_size
 jbox, 505
jbox_get_presentation
 jbox, 505
jbox_get_presentation_position
 jbox, 507
jbox_get_presentation_rect
 jbox, 507
jbox_get_presentation_size
 jbox, 507
jbox_get_prevobject
 jbox, 507
jbox_get_rect_for_sym
 jbox, 508
jbox_get_rect_for_view
 jbox, 508
jbox_get_textfield
 jbox, 508
jbox_get_varname
 jbox, 508
jbox_invalidate_layer
 Box Layer, 647
jbox_new
 jbox, 509
jbox_notify
 jbox, 509
jbox_paint_layer
 Box Layer, 647
jbox_ready
 jbox, 509
jbox_redraw
 jbox, 509
jbox_set_annotation
 jbox, 510
jbox_set_background
 jbox, 510
jbox_set_color
 jbox, 510
jbox_set_fontrname
 jbox, 510
jbox_set_fontsiz
 jbox, 510
jbox_set_hidden
 jbox, 512
jbox_set_hint
 jbox, 512
jbox_set_hintstring
 jbox, 512
jbox_set_ignoreclick
 jbox, 512
jbox_set_outline
 jbox, 513
jbox_set_patching_position
 jbox, 513
jbox_set_patching_rect
 jbox, 513
jbox_set_patching_size
 jbox, 513
jbox_set_position
 jbox, 514
jbox_set_presentation
 jbox, 514
jbox_set_presentation_position
 jbox, 514
jbox_set_presentation_rect
 jbox, 514
jbox_set_presentation_size
 jbox, 515
jbox_set_rect
 jbox, 515
jbox_set_rect_for_sym
 jbox, 515
jbox_set_rect_for_view
 jbox, 515
jbox_set_size
 jbox, 516
jbox_set_varname
 jbox, 516
jbox_start_layer
 Box Layer, 647
jdataview_getclient
 DataView, 649
jdataview_new
 DataView, 650
jdataview_setclient
 DataView, 650
jfont_create
 JFont, 606
jfont_destroy
 JFont, 606
jfont_extents
 JFont, 606
jfont_get_em_dimensions
 JFont, 607
jfont_get_family
 JFont, 607
jfont_get_font_size
 JFont, 607
jfont_get_slant
 JFont, 607
jfont_get_underline
 JFont, 607

jfont_get_weight
 JFont, 608
jfont_getfontlist
 JFont, 608
jfont_isequalto
 JFont, 608
jfont_reference
 JFont, 608
jfont_set_family
 JFont, 609
jfont_set_font_size
 JFont, 609
jfont_set_slant
 JFont, 609
jfont_set_underline
 JFont, 609
jfont_set_weight
 JFont, 609
jfont_text_measure
 JFont, 610
jfont_text_measuretext_wrapped
 JFont, 610
jgraphics_append_path
 JGraphics, 581
jgraphics_arc
 JGraphics, 581
jgraphics_arc_negative
 JGraphics, 582
jgraphics_attr_fillrect
 Styles, 625
jgraphics_attr_setfill
 Styles, 625
jgraphics_bubble
 JGraphics, 582
jgraphics_clip
 JGraphics, 582
jgraphics_close_path
 JGraphics, 582
jgraphics_copy_path
 JGraphics, 583
jgraphics_create
 JSurface, 595
jgraphics_curve_to
 JGraphics, 583
jgraphics_destroy
 JGraphics, 583
jgraphics_device_to_user
 JGraphics, 583
jgraphics_ellipse
 JGraphics, 583
jgraphics_font_extents
 JGraphics, 584
jgraphics_get_current_point
 JGraphics, 584
jgraphics_get_resource_data
 JSurface, 595
jgraphics_getfiletypes
 JGraphics, 584
jgraphics_image_surface_clear
 JSurface, 596
jgraphics_image_surface_create
 JSurface, 596
jgraphics_image_surface_create_for_data
 JSurface, 596
jgraphics_image_surface_create_from_file
 JSurface, 597
jgraphics_image_surface_create_from_filedata
 JSurface, 597
jgraphics_image_surface_create_from_resource
 JSurface, 597
jgraphics_image_surface_create_referenced
 JSurface, 598
jgraphics_image_surface_draw
 JSurface, 598
jgraphics_image_surface_draw_fast
 JSurface, 598
jgraphics_image_surface_get_height
 JSurface, 599
jgraphics_image_surface_get_pixel
 JSurface, 599
jgraphics_image_surface_get_width
 JSurface, 599
jgraphics_image_surface_scroll
 JSurface, 599
jgraphics_image_surface_set_pixel
 JSurface, 600
jgraphics_image_surface_writejpeg
 JSurface, 600
jgraphics_image_surface_writepng
 JSurface, 600
jgraphics_line_to
 JGraphics, 585
jgraphics_matrix_init
 JGraphics Matrix Transformations, 613
jgraphics_matrix_init_identity
 JGraphics Matrix Transformations, 613
jgraphics_matrix_init_rotate
 JGraphics Matrix Transformations, 613
jgraphics_matrix_init_scale
 JGraphics Matrix Transformations, 613
jgraphics_matrix_init_translate
 JGraphics Matrix Transformations, 613
jgraphics_matrix_invert
 JGraphics Matrix Transformations, 614
jgraphics_matrix_multiply
 JGraphics Matrix Transformations, 614
jgraphics_matrix_rotate
 JGraphics Matrix Transformations, 614
jgraphics_matrix_scale
 JGraphics Matrix Transformations, 614
jgraphics_matrix_transform_point
 JGraphics Matrix Transformations, 614
jgraphics_matrix_translate
 JGraphics Matrix Transformations, 615
jgraphics_move_to
 JGraphics, 585

jgraphics_new_path
 JGraphics, 585

jgraphics_oval
 JGraphics, 585

jgraphics_ovalarc
 JGraphics, 585

jgraphics_path_contains
 JGraphics, 586

jgraphics_path_createstroked
 JGraphics, 586

jgraphics_path_destroy
 JGraphics, 586

jgraphics_path_getlength
 JGraphics, 586

jgraphics_path_getnearestpoint
 JGraphics, 586

jgraphics_path_getpathelems
 JGraphics, 588

jgraphics_path_getpointalongpath
 JGraphics, 588

jgraphics_path_intersectsline
 JGraphics, 588

jgraphics_path_roundcorners
 JGraphics, 588

jgraphics_position_one_rect_near_another_rect_but←
 keep_inside_a_third_rect
 JGraphics, 589

jgraphics_rectangle
 JGraphics, 589

jgraphics_rectangle_rounded
 JGraphics, 589

jgraphics_rectcontainsrect
 JGraphics, 589

jgraphics_rectintersectsrect
 JGraphics, 590

jgraphics_reference
 JGraphics, 590

jgraphics_rel_curve_to
 JGraphics, 590

jgraphics_rel_line_to
 JGraphics, 590

jgraphics_rel_move_to
 JGraphics, 591

jgraphics_round
 JGraphics, 591

jgraphics_select_font_face
 JGraphics, 591

jgraphics_select_ifont
 JGraphics, 591

jgraphics_set_font_size
 JGraphics, 591

jgraphics_set_underline
 JGraphics, 592

jgraphics_show_text
 JGraphics, 592

jgraphics_surface_destroy
 JSurface, 600

jgraphics_surface_reference
 JSurface, 601

jgraphics_system_canantialiastexttotransparentbg
 JGraphics, 592

jgraphics_text_measure
 JGraphics, 592

jgraphics_text_measuretext_wrapped
 JGraphics, 592

jgraphics_text_path
 JGraphics, 593

jgraphics_triangle
 JGraphics, 593

jgraphics_user_to_device
 JGraphics, 593

jgraphics_write_image_surface_to_filedata
 JSurface, 601

jit_atom_arg_getdouble
 Atom Module, 654

jit_atom_arg_getfloat
 Atom Module, 655

jit_atom_arg_getlong
 Atom Module, 655

jit_atom_arg_getsym
 Atom Module, 656

jit_atom_getcharfix
 Atom Module, 657

jit_atom_getfloat
 Atom Module, 657

jit_atom_getlong
 Atom Module, 658

jit_atom_getobj
 Atom Module, 658

jit_atom_getsym
 Atom Module, 659

jit_atom_setfloat
 Atom Module, 659

jit_atom_setlong
 Atom Module, 659

jit_atom_setobj
 Atom Module, 661

jit_atom_setsym
 Atom Module, 661

jit_attr_canget
 Attribute Module, 665

jit_attr_canset
 Attribute Module, 666

jit_attr_filter_clip_new
 Attribute Module, 666

jit_attr_filter_proc_new
 Attribute Module, 666

jit_attr_filterget
 Attribute Module, 667

jit_attr_filterset
 Attribute Module, 667

jit_attr_get
 Attribute Module, 668

jit_attr_getchar_array
 Attribute Module, 668

jit_attr_getdouble_array
 Attribute Module, 668

Attribute Module, 669
jit_attr_getfloat
 Attribute Module, 669
jit_attr_getfloat_array
 Attribute Module, 670
jit_attr_getlong
 Attribute Module, 670
jit_attr_getlong_array
 Attribute Module, 672
jit_attr_getmethod
 Attribute Module, 673
jit_attr_getname
 Attribute Module, 673
jit_attr_getsym
 Attribute Module, 673
jit_attr_getsym_array
 Attribute Module, 674
jit_attr_gettime
 Attribute Module, 675
jit_attr_offset_array_new
 Attribute Module, 675
jit_attr_offset_new
 Attribute Module, 676
jit_attr_set
 Attribute Module, 677
jit_attr_setchar_array
 Attribute Module, 677
jit_attr_setdouble_array
 Attribute Module, 678
jit_attr_setfloat
 Attribute Module, 679
jit_attr_setfloat_array
 Attribute Module, 679
jit_attr_setlong
 Attribute Module, 680
jit_attr_setlong_array
 Attribute Module, 680
jit_attr_setsym
 Attribute Module, 681
jit_attr_setsym_array
 Attribute Module, 682
jit_attr_symcompare
 Attribute Module, 682
jit_attr_usercanget
 Attribute Module, 684
jit_attr_usercanset
 Attribute Module, 684
jit_attribute_new
 Attribute Module, 685
jit_bin_read_chunk_info
 Binary Module, 686
jit_bin_read_header
 Binary Module, 687
jit_bin_read_matrix
 Binary Module, 687
jit_bin_write_header
 Binary Module, 688
jit_bin_write_matrix
 Binary Module, 688
Binary Module, 688
jit_class_adaddornment
 Class Module, 692
jit_class_addattr
 Class Module, 692
jit_class_addinterface
 Class Module, 694
jit_class_addmethod
 Class Module, 695
jit_class_addtypedwrapper
 Class Module, 696
jit_class_adornment_get
 Class Module, 696
jit_class_attr_get
 Class Module, 697
jit_class_findbyname
 Class Module, 697
jit_class_free
 Class Module, 697
jit_class_mess
 Class Module, 698
jit_class_method
 Class Module, 698
jit_class_method_addargsafe
 Class Module, 698
jit_class_method_argsafe_get
 Class Module, 699
jit_class_nameget
 Class Module, 700
jit_class_new
 Class Module, 700
jit_class_register
 Class Module, 701
jit_class_symcompare
 Class Module, 702
jit_class_typedwrapper_get
 Class Module, 702
jit_copy_bytes
 Memory Module, 797
jit_disposeptr
 Memory Module, 798
jit_err_from_max_err
 Miscellaneous Utility Module, 718
jit_error_code
 Miscellaneous Utility Module, 719
jit_error_sym
 Miscellaneous Utility Module, 719
jit_freebytes
 Memory Module, 798
jit_freemem
 Memory Module, 799
jit_getbytes
 Memory Module, 799
jit_gl_begincapture
 OB3D Module, 840
jit_gl_bindtexture
 OB3D Module, 840
jit_gl_drawinfo_active_textures

OB3D Module, 842
`jit_gl_drawinfo_setup`
 OB3D Module, 842
`jit_gl_endcapture`
 OB3D Module, 842
`jit_gl_get_extensions`
 OB3D Module, 842
`jit_gl_get_glsl_version`
 OB3D Module, 842
`jit_gl_get_glu_version`
 OB3D Module, 843
`jit_gl_get_renderer`
 OB3D Module, 843
`jit_gl_get_vendor`
 OB3D Module, 843
`jit_gl_get_version`
 OB3D Module, 843
`jit_gl_is_extension_supported`
 OB3D Module, 843
`jit_gl_is_min_version`
 OB3D Module, 844
`jit_gl_report_error`
 OB3D Module, 844
`jit_gl_texcoord1f`
 OB3D Module, 845
`jit_gl_texcoord1fv`
 OB3D Module, 845
`jit_gl_texcoord2f`
 OB3D Module, 845
`jit_gl_texcoord2fv`
 OB3D Module, 845
`jit_gl_texcoord3f`
 OB3D Module, 846
`jit_gl_texcoord3fv`
 OB3D Module, 846
`jit_gl_unbindtexture`
 OB3D Module, 846
`jit_glchunk_copy`
 OB3D Module, 846
`jit_glchunk_delete`
 OB3D Module, 847
`jit_glchunk_grid_new`
 OB3D Module, 847
`jit_glchunk_new`
 OB3D Module, 848
`jit_global_critical_enter`
 Miscellaneous Utility Module, 719
`jit_global_critical_exit`
 Miscellaneous Utility Module, 720
`jit_handle_free`
 Memory Module, 800
`jit_handle_lock`
 Memory Module, 800
`jit_handle_new`
 Memory Module, 801
`jit_handle_size_get`
 Memory Module, 801
`jit_handle_size_set`
 Memory Module, 802
`jit_linklist_append`
 Linked List Module, 724
`jit_linklist_chuck`
 Linked List Module, 724
`jit_linklist_chuckindex`
 Linked List Module, 725
`jit_linklist_clear`
 Linked List Module, 726
`jit_linklist_deleteindex`
 Linked List Module, 726
`jit_linklist_findall`
 Linked List Module, 727
`jit_linklist_findcount`
 Linked List Module, 728
`jit_linklist_findfirst`
 Linked List Module, 729
`jit_linklist_free`
 Matrix Module, 759
`jit_linklist_getindex`
 Linked List Module, 730
`jit_linklist_getsize`
 Linked List Module, 730
`jit_linklist_insertindex`
 Linked List Module, 731
`jit_linklist_makearray`
 Linked List Module, 731
`jit_linklist_methodall`
 Linked List Module, 732
`jit_linklist_methodindex`
 Linked List Module, 733
`jit_linklist_new`
 Linked List Module, 733
`jit_linklist_objptr2index`
 Linked List Module, 734
`jit_linklist_reverse`
 Linked List Module, 735
`jit_linklist_rotate`
 Linked List Module, 735
`jit_linklist_shuffle`
 Linked List Module, 735
`jit_linklist_sort`
 Linked List Module, 737
`jit_linklist_swap`
 Linked List Module, 737
`jit_math_acos`
 Math Module, 741
`jit_math_acosh`
 Math Module, 741
`jit_math_asin`
 Math Module, 741
`jit_math_asinh`
 Math Module, 741
`jit_math_atan`
 Math Module, 742
`jit_math_atan2`
 Math Module, 742
`jit_math_atanh`

Math Module, 742
jit_math_ceil
 Math Module, 742
jit_math_cos
 Math Module, 742
jit_math_cosh
 Math Module, 744
jit_math_exp
 Math Module, 744
jit_math_exp2
 Math Module, 744
jit_math_expm1
 Math Module, 744
jit_math_fast_acos
 Math Module, 744
jit_math_fast_asin
 Math Module, 745
jit_math_fast_atan
 Math Module, 745
jit_math_fast_cos
 Math Module, 747
jit_math_fast_invsqrt
 Math Module, 747
jit_math_fast_sin
 Math Module, 747
jit_math_fast_sqrt
 Math Module, 747
jit_math_fast_tan
 Math Module, 748
jit_math_floor
 Math Module, 748
jit_math_fmod
 Math Module, 748
jit_math_fold
 Math Module, 748
jit_math_hypot
 Math Module, 749
jit_math_is_finite
 Math Module, 749
jit_math_is_nan
 Math Module, 749
jit_math_is_poweroftwo
 Math Module, 749
jit_math_is_valid
 Math Module, 749
jit_math_j1
 Math Module, 751
jit_math_j1_0
 Math Module, 752
jit_math_log
 Math Module, 752
jit_math_log10
 Math Module, 752
jit_math_log2
 Math Module, 753
jit_math_p1
 Math Module, 753
jit_math_pow
 Math Module, 753
jit_math_q1
 Math Module, 753
jit_math_round
 Math Module, 754
jit_math_roundup_poweroftwo
 Math Module, 754
jit_math_sin
 Math Module, 754
jit_math_sinh
 Math Module, 754
jit_math_sqrt
 Math Module, 755
jit_math_tan
 Math Module, 755
jit_math_tanh
 Math Module, 755
jit_math_trunc
 Math Module, 755
jit_math_wrap
 Math Module, 755
jit_matrix_clear
 Matrix Module, 759
jit_matrix_data
 Matrix Module, 761
jit_matrix_exprfill
 Matrix Module, 761
jit_matrix_fillplane
 Matrix Module, 762
jit_matrix_free
 Matrix Module, 762
jit_matrix_freedata
 Matrix Module, 764
jit_matrix_frommatrix
 Matrix Module, 764
jit_matrix_getcell
 Matrix Module, 766
jit_matrix_getdata
 Matrix Module, 767
jit_matrix_getinfo
 Matrix Module, 767
jit_matrix_info_default
 Matrix Module, 768
jit_matrix_jit_gl_texture
 Matrix Module, 768
jit_matrix_new
 Matrix Module, 769
jit_matrix_newcopy
 Matrix Module, 769
jit_matrix_op
 Matrix Module, 771
jit_matrix_setall
 Matrix Module, 772
jit_matrix_setcell
 Matrix Module, 773
jit_matrix_setcell1d
 Matrix Module, 774
jit_matrix_setcell2d

Matrix Module, [775](#)
 jit_matrix_setcell3d
 Matrix Module, [776](#)
 jit_matrix_setinfo
 Matrix Module, [777](#)
 jit_matrix_setinfo_ex
 Matrix Module, [777](#)
 jit_matrix_setplane1d
 Matrix Module, [778](#)
 jit_matrix_setplane2d
 Matrix Module, [779](#)
 jit_matrix_setplane3d
 Matrix Module, [780](#)
 jit_mop_free
 MOP Module, [805](#)
 jit_mop_getinput
 MOP Module, [806](#)
 jit_mop_getinputlist
 MOP Module, [806](#)
 jit_mop_getoutput
 MOP Module, [806](#)
 jit_mop_getoutputlist
 MOP Module, [808](#)
 jit_mop_input_nolink
 MOP Module, [808](#)
 jit_mop_io_free
 MOP Module, [809](#)
 jit_mop_io_getioproc
 MOP Module, [809](#)
 jit_mop_io_getmatrix
 MOP Module, [809](#)
 jit_mop_io_ioproc
 MOP Module, [810](#)
 jit_mop_io_matrix
 MOP Module, [810](#)
 jit_mop_io_new
 MOP Module, [810](#)
 jit_mop_io_newcopy
 MOP Module, [811](#)
 jit_mop_io_restrict_dim
 MOP Module, [811](#)
 jit_mop_io_restrict_planecount
 MOP Module, [812](#)
 jit_mop_io_restrict_type
 MOP Module, [812](#)
 jit_mop_ioproc_copy_adapt
 MOP Module, [813](#)
 jit_mop_ioproc_copy_trunc
 MOP Module, [813](#)
 jit_mop_ioproc_copy_trunc_zero
 MOP Module, [814](#)
 jit_mop_ioproc_tosym
 MOP Module, [814](#)
 jit_mop_methodall
 MOP Module, [815](#)
 jit_mop_new
 MOP Module, [815](#)
 jit_mop_newcopy

MOP Module, [816](#)
 jit_mop_output_nolink
 MOP Module, [817](#)
 jit_mop_single_planecount
 MOP Module, [817](#)
 jit_mop_single_type
 MOP Module, [818](#)
 jit_newptr
 Memory Module, [802](#)
 jit_ob3d_draw_chunk
 OB3D Module, [849](#)
 jit_ob3d_free
 OB3D Module, [849](#)
 jit_ob3d_new
 OB3D Module, [850](#)
 jit_ob3d_set_context
 OB3D Module, [851](#)
 jit_ob3d_setup
 OB3D Module, [851](#)
 jit_object_attach
 Object Module, [705](#)
 jit_object_attr_get
 Object Module, [705](#)
 jit_object_attr_usercanget
 Object Module, [706](#)
 jit_object_attr_usercanset
 Object Module, [706](#)
 jit_object_class
 Object Module, [707](#)
 jit_object_classname
 Object Module, [707](#)
 jit_object_classname_compare
 Object Module, [708](#)
 jit_object_detach
 Object Module, [708](#)
 jit_object_exportattrs
 Object Module, [709](#)
 jit_object_exportsummary
 Object Module, [710](#)
 jit_object_findregistered
 Object Module, [710](#)
 jit_object_findregisteredbyptr
 Object Module, [711](#)
 jit_object_free
 Object Module, [711](#)
 jit_object_getmethod
 Object Module, [712](#)
 jit_object_importattrs
 Object Module, [712](#)
 jit_object_method_argsafe_get
 Object Module, [713](#)
 jit_object_method_imp
 Object Module, [714](#)
 jit_object_method_typed
 Object Module, [714](#)
 jit_object_new_imp
 Object Module, [715](#)
 jit_object_notify

Object Module, 715
jit_object_register
 Object Module, 716
jit_object_unregister
 Object Module, 716
jit_op_vector_abs_float32
 Operator Vector Module, 865
jit_op_vector_abs_float64
 Operator Vector Module, 865
jit_op_vector_abs_long
 Operator Vector Module, 865
jit_op_vector_absdiff_char
 Operator Vector Module, 866
jit_op_vector_absdiff_float32
 Operator Vector Module, 866
jit_op_vector_absdiff_float64
 Operator Vector Module, 866
jit_op_vector_absdiff_long
 Operator Vector Module, 866
jit_op_vector_acos_float32
 Operator Vector Module, 867
jit_op_vector_acos_float64
 Operator Vector Module, 867
jit_op_vector_acosh_float32
 Operator Vector Module, 867
jit_op_vector_acosh_float64
 Operator Vector Module, 867
jit_op_vector_add_char
 Operator Vector Module, 868
jit_op_vector_add_float32
 Operator Vector Module, 868
jit_op_vector_add_float64
 Operator Vector Module, 868
jit_op_vector_add_long
 Operator Vector Module, 868
jit_op_vector_adds_char
 Operator Vector Module, 869
jit_op_vector_and_char
 Operator Vector Module, 869
jit_op_vector_and_float32
 Operator Vector Module, 869
jit_op_vector_and_float64
 Operator Vector Module, 869
jit_op_vector_and_long
 Operator Vector Module, 870
jit_op_vector_asin_float32
 Operator Vector Module, 870
jit_op_vector_asin_float64
 Operator Vector Module, 870
jit_op_vector_asinh_float32
 Operator Vector Module, 870
jit_op_vector_asinh_float64
 Operator Vector Module, 871
jit_op_vector_atan2_float32
 Operator Vector Module, 871
jit_op_vector_atan2_float64
 Operator Vector Module, 871
jit_op_vector_atan_float32
 Operator Vector Module, 871
jit_op_vector_atanh_float32
 Operator Vector Module, 872
jit_op_vector_atanh_float64
 Operator Vector Module, 872
jit_op_vector_avg_char
 Operator Vector Module, 872
jit_op_vector_avg_float32
 Operator Vector Module, 873
jit_op_vector_avg_float64
 Operator Vector Module, 873
jit_op_vector_avg_long
 Operator Vector Module, 873
jit_op_vector_bitand_char
 Operator Vector Module, 873
jit_op_vector_bitand_long
 Operator Vector Module, 874
jit_op_vector_bitnot_char
 Operator Vector Module, 874
jit_op_vector_bitnot_long
 Operator Vector Module, 874
jit_op_vector_bitor_char
 Operator Vector Module, 874
jit_op_vector_bitor_long
 Operator Vector Module, 875
jit_op_vector_bitxor_char
 Operator Vector Module, 875
jit_op_vector_bitxor_long
 Operator Vector Module, 875
jit_op_vector_ceil_float32
 Operator Vector Module, 875
jit_op_vector_ceil_float64
 Operator Vector Module, 876
jit_op_vector_cos_float32
 Operator Vector Module, 876
jit_op_vector_cos_float64
 Operator Vector Module, 876
jit_op_vector_cosh_float32
 Operator Vector Module, 876
jit_op_vector_cosh_float64
 Operator Vector Module, 877
jit_op_vector_div_char
 Operator Vector Module, 877
jit_op_vector_div_float32
 Operator Vector Module, 877
jit_op_vector_div_float64
 Operator Vector Module, 877
jit_op_vector_div_long
 Operator Vector Module, 878
jit_op_vector_eq_char
 Operator Vector Module, 878
jit_op_vector_eq_float32
 Operator Vector Module, 878
jit_op_vector_eq_float64
 Operator Vector Module, 878
jit_op_vector_eq_long
 Operator Vector Module, 878

Operator Vector Module, 879
`jit_op_vector_eqp_char`
 Operator Vector Module, 879
`jit_op_vector_eqp_float32`
 Operator Vector Module, 879
`jit_op_vector_eqp_float64`
 Operator Vector Module, 879
`jit_op_vector_eqp_long`
 Operator Vector Module, 880
`jit_op_vector_exp2_float32`
 Operator Vector Module, 880
`jit_op_vector_exp2_float64`
 Operator Vector Module, 880
`jit_op_vector_exp_float32`
 Operator Vector Module, 880
`jit_op_vector_exp_float64`
 Operator Vector Module, 881
`jit_op_vector_flipdiv_char`
 Operator Vector Module, 881
`jit_op_vector_flipdiv_float32`
 Operator Vector Module, 881
`jit_op_vector_flipdiv_float64`
 Operator Vector Module, 882
`jit_op_vector_flipdiv_long`
 Operator Vector Module, 882
`jit_op_vector_flipmod_char`
 Operator Vector Module, 883
`jit_op_vector_flipmod_float32`
 Operator Vector Module, 883
`jit_op_vector_flipmod_float64`
 Operator Vector Module, 884
`jit_op_vector_flipmod_long`
 Operator Vector Module, 884
`jit_op_vector_flippass_char`
 Operator Vector Module, 885
`jit_op_vector_flippass_float32`
 Operator Vector Module, 885
`jit_op_vector_flippass_float64`
 Operator Vector Module, 886
`jit_op_vector_flippass_long`
 Operator Vector Module, 886
`jit_op_vector_flipsub_char`
 Operator Vector Module, 887
`jit_op_vector_flipsub_float32`
 Operator Vector Module, 887
`jit_op_vector_flipsub_long`
 Operator Vector Module, 888
`jit_op_vector_floor_float32`
 Operator Vector Module, 888
`jit_op_vector_floor_float64`
 Operator Vector Module, 889
`jit_op_vector_fold_float32`
 Operator Vector Module, 889
`jit_op_vector_fold_float64`
 Operator Vector Module, 889
`jit_op_vector_gt_char`
 Operator Vector Module, 889
`jit_op_vector_gt_float32`
 Operator Vector Module, 890
`jit_op_vector_gt_float64`
 Operator Vector Module, 890
`jit_op_vector_gt_long`
 Operator Vector Module, 890
`jit_op_vector_gte_char`
 Operator Vector Module, 890
`jit_op_vector_gte_float32`
 Operator Vector Module, 891
`jit_op_vector_gte_float64`
 Operator Vector Module, 891
`jit_op_vector_gte_long`
 Operator Vector Module, 891
`jit_op_vector_gtep_char`
 Operator Vector Module, 891
`jit_op_vector_gtep_float32`
 Operator Vector Module, 892
`jit_op_vector_gtep_float64`
 Operator Vector Module, 892
`jit_op_vector_gtep_long`
 Operator Vector Module, 892
`jit_op_vector_gtp_char`
 Operator Vector Module, 892
`jit_op_vector_gtp_float32`
 Operator Vector Module, 893
`jit_op_vector_gtp_float64`
 Operator Vector Module, 893
`jit_op_vector_gtp_long`
 Operator Vector Module, 893
`jit_op_vector_hypot_float32`
 Operator Vector Module, 893
`jit_op_vector_hypot_float64`
 Operator Vector Module, 894
`jit_op_vector_log10_float32`
 Operator Vector Module, 894
`jit_op_vector_log10_float64`
 Operator Vector Module, 894
`jit_op_vector_log2_float32`
 Operator Vector Module, 894
`jit_op_vector_log2_float64`
 Operator Vector Module, 895
`jit_op_vector_log_float32`
 Operator Vector Module, 895
`jit_op_vector_log_float64`
 Operator Vector Module, 895
`jit_op_vector_lshift_char`
 Operator Vector Module, 895
`jit_op_vector_lshift_long`
 Operator Vector Module, 896
`jit_op_vector_lt_char`
 Operator Vector Module, 896
`jit_op_vector_lt_float32`
 Operator Vector Module, 896
`jit_op_vector_lt_float64`
 Operator Vector Module, 896
`jit_op_vector_lt_long`
 Operator Vector Module, 897
`jit_op_vector_lte_char`

Operator Vector Module, 897
jit_op_vector_lte_float32
 Operator Vector Module, 897
jit_op_vector_lte_float64
 Operator Vector Module, 897
jit_op_vector_lte_long
 Operator Vector Module, 898
jit_op_vector_ltep_char
 Operator Vector Module, 898
jit_op_vector_ltep_float32
 Operator Vector Module, 898
jit_op_vector_ltep_float64
 Operator Vector Module, 898
jit_op_vector_ltep_long
 Operator Vector Module, 899
jit_op_vector_ltp_char
 Operator Vector Module, 899
jit_op_vector_ltp_float32
 Operator Vector Module, 899
jit_op_vector_ltp_float64
 Operator Vector Module, 899
jit_op_vector_ltp_long
 Operator Vector Module, 900
jit_op_vector_max_char
 Operator Vector Module, 900
jit_op_vector_max_float32
 Operator Vector Module, 900
jit_op_vector_max_float64
 Operator Vector Module, 900
jit_op_vector_max_long
 Operator Vector Module, 901
jit_op_vector_min_char
 Operator Vector Module, 901
jit_op_vector_min_float32
 Operator Vector Module, 901
jit_op_vector_min_float64
 Operator Vector Module, 901
jit_op_vector_min_long
 Operator Vector Module, 902
jit_op_vector_mod_char
 Operator Vector Module, 902
jit_op_vector_mod_float32
 Operator Vector Module, 902
jit_op_vector_mod_float64
 Operator Vector Module, 902
jit_op_vector_mod_long
 Operator Vector Module, 903
jit_op_vector_mult_char
 Operator Vector Module, 903
jit_op_vector_mult_float32
 Operator Vector Module, 903
jit_op_vector_mult_float64
 Operator Vector Module, 903
jit_op_vector_mult_long
 Operator Vector Module, 904
jit_op_vector_neq_char
 Operator Vector Module, 904
jit_op_vector_neq_float32
 Operator Vector Module, 904
jit_op_vector_neq_float64
 Operator Vector Module, 904
jit_op_vector_neq_long
 Operator Vector Module, 905
jit_op_vector_neqp_char
 Operator Vector Module, 905
jit_op_vector_neqp_float32
 Operator Vector Module, 905
jit_op_vector_neqp_float64
 Operator Vector Module, 905
jit_op_vector_neqp_long
 Operator Vector Module, 906
jit_op_vector_not_char
 Operator Vector Module, 906
jit_op_vector_not_float32
 Operator Vector Module, 906
jit_op_vector_not_float64
 Operator Vector Module, 906
jit_op_vector_not_long
 Operator Vector Module, 907
jit_op_vector_or_char
 Operator Vector Module, 907
jit_op_vector_or_float32
 Operator Vector Module, 907
jit_op_vector_or_float64
 Operator Vector Module, 907
jit_op_vector_or_long
 Operator Vector Module, 908
jit_op_vector_pass_char
 Operator Vector Module, 908
jit_op_vector_pass_float32
 Operator Vector Module, 908
jit_op_vector_pass_float64
 Operator Vector Module, 908
jit_op_vector_pass_long
 Operator Vector Module, 909
jit_op_vector_pow_float32
 Operator Vector Module, 909
jit_op_vector_pow_float64
 Operator Vector Module, 909
jit_op_vector_round_float32
 Operator Vector Module, 909
jit_op_vector_round_float64
 Operator Vector Module, 910
jit_op_vector_rshift_char
 Operator Vector Module, 910
jit_op_vector_rshift_long
 Operator Vector Module, 910
jit_op_vector_sin_float32
 Operator Vector Module, 910
jit_op_vector_sin_float64
 Operator Vector Module, 911
jit_op_vector_sinh_float32
 Operator Vector Module, 911
jit_op_vector_sinh_float64
 Operator Vector Module, 911
jit_op_vector_sqrt_float32

Operator Vector Module, 911
`jit_op_vector_sqrt_float64`
 Operator Vector Module, 912
`jit_op_vector_sub_char`
 Operator Vector Module, 912
`jit_op_vector_sub_float32`
 Operator Vector Module, 912
`jit_op_vector_sub_float64`
 Operator Vector Module, 912
`jit_op_vector_sub_long`
 Operator Vector Module, 913
`jit_op_vector_subs_char`
 Operator Vector Module, 913
`jit_op_vector_tan_float32`
 Operator Vector Module, 913
`jit_op_vector_tan_float64`
 Operator Vector Module, 913
`jit_op_vector_tanh_float32`
 Operator Vector Module, 914
`jit_op_vector_tanh_float64`
 Operator Vector Module, 914
`jit_op_vector_trunc_float32`
 Operator Vector Module, 914
`jit_op_vector_trunc_float64`
 Operator Vector Module, 914
`jit_op_vector_wrap_float32`
 Operator Vector Module, 915
`jit_op_vector_wrap_float64`
 Operator Vector Module, 915
`jit_parallel_ndim_calc`
 Parallel Utility Module, 819
`jit_parallel_ndim_simplecalc1`
 Parallel Utility Module, 820
`jit_parallel_ndim_simplecalc2`
 Parallel Utility Module, 820
`jit_parallel_ndim_simplecalc3`
 Parallel Utility Module, 821
`jit_parallel_ndim_simplecalc4`
 Parallel Utility Module, 822
`jit_post_sym`
 Miscellaneous Utility Module, 720
`jit_rand`
 Miscellaneous Utility Module, 721
`jit_rand_setseed`
 Miscellaneous Utility Module, 721
`Jitter`, 373
`jkeyboard_getcurrentmodifiers`
 Mouse and Keyboard, 433
`jkeyboard_getcurrentmodifiers_realtime`
 Mouse and Keyboard, 433
`jmonitor_getdisplayrect`
 Monitors and Displays, 428
`jmonitor_getdisplayrect_foralldisplays`
 Monitors and Displays, 428
`jmonitor_getdisplayrect_forpoint`
 Monitors and Displays, 429
`jmonitor_getnumdisplays`
 Monitors and Displays, 429
`jmouse_getposition_global`
 Mouse and Keyboard, 433
`jmouse_setcursor`
 Mouse and Keyboard, 433
`jmouse_setposition_box`
 Mouse and Keyboard, 433
`jmouse_setposition_global`
 Mouse and Keyboard, 433
`jmouse_setposition_view`
 Mouse and Keyboard, 434
`jpatcher`, 480
 `jpatcher_deleteobj`, 482
 `jpatcher_get bgcolor`, 482
 `jpatcher_get bghidden`, 482
 `jpatcher_get bglocked`, 482
 `jpatcher_get box`, 483
 `jpatcher_get count`, 483
 `jpatcher_get currentfileversion`, 483
 `jpatcher_get default fontface`, 483
 `jpatcher_get default fontname`, 483
 `jpatcher_get default fontsize`, 484
 `jpatcher_get deflect`, 484
 `jpatcher_get dirty`, 484
 `jpatcher_get editing bgcolor`, 484
 `jpatcher_get fghidden`, 485
 `jpatcher_get filename`, 485
 `jpatcher_get filepath`, 485
 `jpatcher_get fileversion`, 485
 `jpatcher_get firstline`, 485
 `jpatcher_get firstobject`, 487
 `jpatcher_get firstview`, 487
 `jpatcher_get gridsize`, 487
 `jpatcher_get hubholder`, 487
 `jpatcher_get lastobject`, 488
 `jpatcher_get locked bgcolor`, 488
 `jpatcher_get name`, 488
 `jpatcher_get parentpatcher`, 488
 `jpatcher_get presentation`, 489
 `jpatcher_get rect`, 489
 `jpatcher_get title`, 489
 `jpatcher_get toppatcher`, 489
 `jpatcher_is patcher`, 490
 `jpatcher_set bgcolor`, 490
 `jpatcher_set bghidden`, 490
 `jpatcher_set bglocked`, 490
 `jpatcher_set deflect`, 491
 `jpatcher_set dirty`, 491
 `jpatcher_set editing bgcolor`, 491
 `jpatcher_set fghidden`, 491
 `jpatcher_set gridsize`, 492
 `jpatcher_set locked bgcolor`, 492
 `jpatcher_set presentation`, 492
 `jpatcher_set rect`, 493
 `jpatcher_set title`, 493
 `jpatcher_uniqueboxname`, 493
`jpatcher_deleteobj`
 `jpatcher`, 482

jpatcher_get_bgcolor
 jpatcher, 482
jpatcher_get_bghidden
 jpatcher, 482
jpatcher_get_bglocked
 jpatcher, 482
jpatcher_get_box
 jpatcher, 483
jpatcher_get_count
 jpatcher, 483
jpatcher_get_currentfileversion
 jpatcher, 483
jpatcher_get_default_fontface
 jpatcher, 483
jpatcher_get_default_fontname
 jpatcher, 483
jpatcher_get_default_fontsize
 jpatcher, 484
jpatcher_get_defrect
 jpatcher, 484
jpatcher_get_dirty
 jpatcher, 484
jpatcher_get_editing_bgcolor
 jpatcher, 484
jpatcher_get_fghidden
 jpatcher, 485
jpatcher_get_filename
 jpatcher, 485
jpatcher_get_filepath
 jpatcher, 485
jpatcher_get_fileversion
 jpatcher, 485
jpatcher_get_firstline
 jpatcher, 485
jpatcher_get_firstobject
 jpatcher, 487
jpatcher_get_firstview
 jpatcher, 487
jpatcher_get_gridsize
 jpatcher, 487
jpatcher_get_hubholder
 jpatcher, 487
jpatcher_get_lastobject
 jpatcher, 488
jpatcher_get_locked_bgcolor
 jpatcher, 488
jpatcher_get_name
 jpatcher, 488
jpatcher_get_parentpatcher
 jpatcher, 488
jpatcher_get_presentation
 jpatcher, 489
jpatcher_get_rect
 jpatcher, 489
jpatcher_get_title
 jpatcher, 489
jpatcher_get_toppatcher
 jpatcher, 489
jpatcher_is_patch
 jpatcher, 490
jpatcher_set_bgcolor
 jpatcher, 490
jpatcher_set_bghidden
 jpatcher, 490
jpatcher_set_bglocked
 jpatcher, 490
jpatcher_set_defrect
 jpatcher, 491
jpatcher_set_dirty
 jpatcher, 491
jpatcher_set_editing_bgcolor
 jpatcher, 491
jpatcher_set_fghidden
 jpatcher, 491
jpatcher_set_grayscale
 jpatcher, 492
jpatcher_set_locked
 jpatcher, 492
jpatcher_set_locked_bgcolor
 jpatcher, 492
jpatcher_set_presentation
 jpatcher, 492
jpatcher_set_rect
 jpatcher, 493
jpatcher_set_title
 jpatcher, 493
jpatcher_uniqueboxname
 jpatcher, 493
jpatcherview, 522
 patcherview_canvas_to_screen, 523
 patcherview_findpatcherview, 523
 patcherview_get_jgraphics, 523
 patcherview_get_locked, 523
 patcherview_get_nextview, 523
 patcherview_get_patch, 524
 patcherview_get_presentation, 524
 patcherview_get_rect, 524
 patcherview_get_topview, 524
 patcherview_get_visible, 525
 patcherview_get_zoomfactor, 525
 patcherview_screen_to_canvas, 525
 patcherview_set_locked, 525
 patcherview_set_presentation, 525
 patcherview_set_rect, 527
 patcherview_set_visible, 527
 patcherview_set_zoomfactor, 527
jpatchline, 517
 jpatchline_get_box1, 517
 jpatchline_get_box2, 518
 jpatchline_get_color, 518
 jpatchline_get_endpoint, 518
 jpatchline_get_hidden, 518
 jpatchline_get_inletnum, 519
 jpatchline_get_nextline, 519
 jpatchline_get_nummidpoints, 519
 jpatchline_get_outletnum, 519

jpatchline_get_startpoint, 519
 jpatchline_set_color, 521
 jpatchline_set_hidden, 521
 jpatchline_get_box1
 jpatchline, 517
 jpatchline_get_box2
 jpatchline, 518
 jpatchline_get_color
 jpatchline, 518
 jpatchline_get_endpoint
 jpatchline, 518
 jpatchline_get_hidden
 jpatchline, 518
 jpatchline_get_inletnum
 jpatchline, 519
 jpatchline_get_nextline
 jpatchline, 519
 jpatchline_get_nummidpoints
 jpatchline, 519
 jpatchline_get_outletnum
 jpatchline, 519
 jpatchline_get_startpoint
 jpatchline, 519
 jpatchline_set_color
 jpatchline, 521
 jpatchline_set_hidden
 jpatchline, 521
 jpopupmenu_additem
 Popup Menus, 642
 jpopupmenu_addseperator
 Popup Menus, 642
 jpopupmenu_addsubmenu
 Popup Menus, 642
 jpopupmenu_clear
 Popup Menus, 642
 jpopupmenu_create
 Popup Menus, 642
 jpopupmenu_destroy
 Popup Menus, 642
 jpopupmenu_popup
 Popup Menus, 644
 jpopupmenu_popup_abovebox
 Popup Menus, 644
 jpopupmenu_popup_aboverect
 Popup Menus, 644
 jpopupmenu_popup_belowrect
 Popup Menus, 644
 jpopupmenu_popup_nearbox
 Popup Menus, 645
 jpopupmenu_setcolors
 Popup Menus, 645
 jpopupmenu_setfont
 Popup Menus, 645
 jrgba_attr_get
 Colors, 618
 jrgba_attr_set
 Colors, 618
 jrgba_compare
 Colors, 618
 jrgba_copy
 Colors, 619
 jrgba_set
 Colors, 619
 jrgba_to_atoms
 Colors, 619
 jsvg_create_from_file
 Scalable Vector Graphics, 602
 jsvg_create_from_resource
 Scalable Vector Graphics, 602
 jsvg_create_from_xmlstring
 Scalable Vector Graphics, 603
 jsvg_destroy
 Scalable Vector Graphics, 603
 jsvg_get_size
 Scalable Vector Graphics, 603
 jsvg_render
 Scalable Vector Graphics, 603
 jtextlayout_create
 TextLayout, 637
 jtextlayout_createpath
 TextLayout, 637
 jtextlayout_destroy
 TextLayout, 637
 jtextlayout_draw
 TextLayout, 637
 jtextlayout_getchar
 TextLayout, 638
 jtextlayout_getcharbox
 TextLayout, 638
 jtextlayout_getnumchars
 TextLayout, 638
 jtextlayout_measuretext
 TextLayout, 638
 jtextlayout_set
 TextLayout, 639
 jtextlayout_settext
 TextLayout, 639
 jtextlayout_settextcolor
 TextLayout, 639
 jtextlayout_withbgcolor
 TextLayout, 639
 jwind_getactive
 Windows, 430
 jwind_getat
 Windows, 430
 jwind_getcount
 Windows, 430

 Linked List, 288
 linklist_append, 291
 linklist_chuck, 291
 linklist_chuckindex, 291
 linklist_chuckobject, 292
 linklist_clear, 292
 linklist_deleteindex, 292
 linklist_deleteobject, 292
 linklist_findall, 293

linklist_findfirst, 293
linklist_flags, 294
linklist_funall, 294
linklist_funall_break, 294
linklist_funindex, 295
linklist_getflags, 295
linklist_getindex, 296
linklist_getsize, 296
linklist_insert_sorted, 296
linklist_insertafterobjptr, 296
linklist_insertbeforeobjptr, 297
linklist_insertindex, 297
linklist_last, 297
linklist_makearray, 297
linklist_match, 298
linklist_methodall, 298
linklist_methodindex, 298
linklist_moveafterobjptr, 299
linklist_movebeforeobjptr, 299
linklist_new, 299
linklist_next, 299
linklist_objptr2index, 300
linklist_prev, 300
linklist_READONLY, 300
linklist_reverse, 300
linklist_rotate, 301
linklist_shuffle, 301
linklist_sort, 301
linklist_substitute, 301
linklist_swap, 302
Linked List Module, 723
 jit_linklist_append, 724
 jit_linklist_chuck, 724
 jit_linklist_chuckindex, 725
 jit_linklist_clear, 726
 jit_linklist_deleteindex, 726
 jit_linklist_findall, 727
 jit_linklist_findcount, 728
 jit_linklist_findfirst, 729
 jit_linklist_getindex, 730
 jit_linklist_getsize, 730
 jit_linklist_insertindex, 731
 jit_linklist_makearray, 731
 jit_linklist_methodall, 732
 jit_linklist_methodindex, 733
 jit_linklist_new, 733
 jit_linklist_objptr2index, 734
 jit_linklist_reverse, 735
 jit_linklist_rotate, 735
 jit_linklist_shuffle, 735
 jit_linklist_sort, 737
 jit_linklist_swap, 737
linklist_append
 Linked List, 291
linklist_chuck
 Linked List, 291
linklist_chuckindex
 Linked List, 291
linklist_chuckobject
 Linked List, 292
linklist_clear
 Linked List, 292
linklist_deleteindex
 Linked List, 292
linklist_deleteobject
 Linked List, 292
linklist_findall
 Linked List, 293
linklist_findfirst
 Linked List, 293
linklist_flags
 Linked List, 294
linklist_funall
 Linked List, 294
linklist_funall_break
 Linked List, 294
linklist_funindex
 Linked List, 295
linklist_getflags
 Linked List, 295
linklist_getindex
 Linked List, 296
linklist_getsize
 Linked List, 296
linklist_insert_sorted
 Linked List, 296
linklist_insertafterobjptr
 Linked List, 296
linklist_insertbeforeobjptr
 Linked List, 297
linklist_insertindex
 Linked List, 297
linklist_last
 Linked List, 297
linklist_makearray
 Linked List, 297
linklist_match
 Linked List, 298
linklist_methodall
 Linked List, 298
linklist_methodindex
 Linked List, 298
linklist_moveafterobjptr
 Linked List, 299
linklist_movebeforeobjptr
 Linked List, 299
linklist_new
 Linked List, 299
linklist_next
 Linked List, 299
linklist_objptr2index
 Linked List, 300
linklist_prev
 Linked List, 300
linklist_READONLY
 Linked List, 300

linklist_reverse
 Linked List, 300

linklist_rotate
 Linked List, 301

linklist_shuffle
 Linked List, 301

linklist_sort
 Linked List, 301

linklist_substitute
 Linked List, 301

linklist_swap
 Linked List, 302

listout
 Inlets and Outlets, 223

Loading Max Files, 425
 fileload, 425
 intload, 425
 readtohandle, 427
 stringload, 427

locatefile
 Files and Folders, 358

locatefile_extended
 Files and Folders, 359

locatefiletype
 Files and Folders, 360

MAX
 Miscellaneous, 398

MAX_ERR_DUPLICATE
 Miscellaneous, 399

MAX_ERR_GENERIC
 Miscellaneous, 399

MAX_ERR_INVALID_PTR
 Miscellaneous, 399

MAX_ERR_NONE
 Miscellaneous, 399

MAX_ERR_OUT_OF_MEM
 Miscellaneous, 399

MAX_FILENAME_CHARS
 Files and Folders, 355

MIN
 Miscellaneous, 398

MM_UNIFIED
 Memory Management, 384

MOP Max Wrapper Module, 823
 max_jit_classex_mop_mproc, 824
 max_jit_classex_mop_wrap, 824
 max_jit_mop_adapt_matrix_all, 825
 max_jit_mop_assist, 826
 max_jit_mop_bang, 826
 max_jit_mop_clear, 827
 max_jit_mop_free, 827
 max_jit_mop_get_io_by_name, 828
 max_jit_mop_getinput, 828
 max_jit_mop_getoutput, 829
 max_jit_mop_getoutputmode, 829
 max_jit_mop_inputs, 830
 max_jit_mop_jit_matrix, 830
 max_jit_mop_matrix_args, 831

max_jit_mop_matrixout_new, 832
 max_jit_mop_notify, 832
 max_jit_mop_outputmatrix, 833
 max_jit_mop_outputs, 834
 max_jit_mop_setup, 834
 max_jit_mop_setup_simple, 835
 max_jit_mop_variable_addinputs, 836
 max_jit_mop_variable_addoutputs, 836

MOP Module, 804
 jit_mop_free, 805
 jit_mop_getinput, 806
 jit_mop_getinputlist, 806
 jit_mop_getoutput, 806
 jit_mop_getoutputlist, 808
 jit_mop_input_nolink, 808
 jit_mop_io_free, 809
 jit_mop_io_getioproc, 809
 jit_mop_io_getmatrix, 809
 jit_mop_io_ioproc, 810
 jit_mop_io_matrix, 810
 jit_mop_io_new, 810
 jit_mop_io_newcopy, 811
 jit_mop_io_restrict_dim, 811
 jit_mop_io_restrict_planecount, 812
 jit_mop_io_restrict_type, 812
 jit_mop_ioproc_copy_adapt, 813
 jit_mop_ioproc_copy_trunc, 813
 jit_mop_ioproc_copy_trunc_zero, 814
 jit_mop_ioproc_tosym, 814
 jit_mop_methodall, 815
 jit_mop_new, 815
 jit_mop_newcopy, 816
 jit_mop_output_nolink, 817
 jit_mop_single_planecount, 817
 jit_mop_single_type, 818

MSG_MAXARG
 Objects, 452

MSP, 435
 class_dspinit, 438
 class_dspinitbbox, 438
 dsp_add, 438
 dsp_addv, 439
 PI, 437
 PIOVERTWO, 437
 SYS_MAXBLKSIZE, 438
 SYS_MAXSIGS, 438
 sys_getblksize, 439
 sys_getdspobjdspstate, 439
 sys_getdspstate, 439
 sys_getmaxblksize, 439
 sys_getsr, 440
 t_double, 437
 t_float, 437
 t_perfroutine, 440
 t_sample, 437
 t_vptr, 437
 TWOPI, 437
 vptr, 437

z_dsp_free, 440
z_dsp_setup, 440
Math Module, 739
 jit_math_acos, 741
 jit_math_acosh, 741
 jit_math_asin, 741
 jit_math_asinh, 741
 jit_math_atan, 742
 jit_math_atan2, 742
 jit_math_atanh, 742
 jit_math_ceil, 742
 jit_math_cos, 742
 jit_math_cosh, 744
 jit_math_exp, 744
 jit_math_exp2, 744
 jit_math_expm1, 744
 jit_math_fast_acos, 744
 jit_math_fast_asin, 745
 jit_math_fast_atan, 745
 jit_math_fast_cos, 747
 jit_math_fast_invsqrt, 747
 jit_math_fast_sin, 747
 jit_math_fast_sqrt, 747
 jit_math_fast_tan, 748
 jit_math_floor, 748
 jit_math_fmod, 748
 jit_math_fold, 748
 jit_math_hypot, 749
 jit_math_is_finite, 749
 jit_math_is_nan, 749
 jit_math_is_poweroftwo, 749
 jit_math_is_valid, 749
 jit_math_j1, 751
 jit_math_j1_0, 752
 jit_math_log, 752
 jit_math_log10, 752
 jit_math_log2, 753
 jit_math_p1, 753
 jit_math_pow, 753
 jit_math_q1, 753
 jit_math_round, 754
 jit_math_roundup_poweroftwo, 754
 jit_math_sin, 754
 jit_math_sinh, 754
 jit_math_sqrt, 755
 jit_math_tan, 755
 jit_math_tanh, 755
 jit_math_trunc, 755
 jit_math_wrap, 755
Matrix Module, 758
 jit_linklist_free, 759
 jit_matrix_clear, 759
 jit_matrix_data, 761
 jit_matrix_exprfill, 761
 jit_matrix_fillplane, 762
 jit_matrix_free, 762
 jit_matrix_freedata, 764
 jit_matrix_frommatrix, 764
 jit_matrix_getcell, 766
 jit_matrix_getdata, 767
 jit_matrix_getinfo, 767
 jit_matrix_info_default, 768
 jit_matrix_jit_gl_texture, 768
 jit_matrix_new, 769
 jit_matrix_newcopy, 769
 jit_matrix_op, 771
 jit_matrix_setall, 772
 jit_matrix_setcell, 773
 jit_matrix_setcell1d, 774
 jit_matrix_setcell2d, 775
 jit_matrix_setcell3d, 776
 jit_matrix_setinfo, 777
 jit_matrix_setinfo_ex, 777
 jit_matrix_setplane1d, 778
 jit_matrix_setplane2d, 779
 jit_matrix_setplane3d, 780
Max Wrapper Module, 782
 max_addmethod_defer, 783
 max_addmethod_defer_low, 783
 max_addmethod_usurp, 784
 max_addmethod_usurp_low, 784
 max_jit_attr_args, 785
 max_jit_attr_args_offset, 785
 max_jit_attr_get, 786
 max_jit_attr_getdump, 786
 max_jit_attr_set, 787
 max_jit_classex_addattr, 787
 max_jit_classex_setup, 788
 max_jit_classex_standard_wrap, 788
 max_jit_obex_adornment_get, 788
 max_jit_obex_attr_get, 789
 max_jit_obex_attr_set, 789
 max_jit_obex_dumpout, 790
 max_jit_obex_dumpout_get, 790
 max_jit_obex_dumpout_set, 791
 max_jit_obex_free, 791
 max_jit_obex_gimmeback, 791
 max_jit_obex_gimmeback_dumpout, 792
 max_jit_obex_inletnumber_get, 792
 max_jit_obex_inletnumber_set, 793
 max_jit_obex_jitob_get, 793
 max_jit_obex_jitob_set, 793
 max_jit_obex_new, 795
 max_jit_obex_proxy_new, 795
 max_addmethod_defer
 Max Wrapper Module, 783
 max_addmethod_defer_low
 Max Wrapper Module, 783
 max_addmethod_usurp
 Max Wrapper Module, 784
 max_addmethod_usurp_low
 Max Wrapper Module, 784
 max_jit_attr_args
 Max Wrapper Module, 785
 max_jit_attr_args_offset
 Max Wrapper Module, 785

max_jit_attr_get
 Max Wrapper Module, 786

max_jit_attr_getdump
 Max Wrapper Module, 786

max_jit_attr_set
 Max Wrapper Module, 787

max_jit_classex_addattr
 Max Wrapper Module, 787

max_jit_classex_mop_mproc
 MOP Max Wrapper Module, 824

max_jit_classex_mop_wrap
 MOP Max Wrapper Module, 824

max_jit_classex_setup
 Max Wrapper Module, 788

max_jit_classex_standard_wrap
 Max Wrapper Module, 788

max_jit_mop_adapt_matrix_all
 MOP Max Wrapper Module, 825

max_jit_mop_assist
 MOP Max Wrapper Module, 826

max_jit_mop_bang
 MOP Max Wrapper Module, 826

max_jit_mop_clear
 MOP Max Wrapper Module, 827

max_jit_mop_free
 MOP Max Wrapper Module, 827

max_jit_mop_get_io_by_name
 MOP Max Wrapper Module, 828

max_jit_mop_getinput
 MOP Max Wrapper Module, 828

max_jit_mop_getoutput
 MOP Max Wrapper Module, 829

max_jit_mop_getoutputmode
 MOP Max Wrapper Module, 829

max_jit_mop_inputs
 MOP Max Wrapper Module, 830

max_jit_mop_jit_matrix
 MOP Max Wrapper Module, 830

max_jit_mop_matrix_args
 MOP Max Wrapper Module, 831

max_jit_mop_matrixout_new
 MOP Max Wrapper Module, 832

max_jit_mop_notify
 MOP Max Wrapper Module, 832

max_jit_mop_outputmatrix
 MOP Max Wrapper Module, 833

max_jit_mop_outputs
 MOP Max Wrapper Module, 834

max_jit_mop_setup
 MOP Max Wrapper Module, 834

max_jit_mop_setup_simple
 MOP Max Wrapper Module, 835

max_jit_mop_variable_addinputs
 MOP Max Wrapper Module, 836

max_jit_mop_variable_addoutputs
 MOP Max Wrapper Module, 836

max_jit_ob3d_assist
 OB3D Module, 852

max_jit_ob3d_attach
 OB3D Module, 852

max_jit_ob3d_detach
 OB3D Module, 853

max_jit_obex_adornment_get
 Max Wrapper Module, 788

max_jit_obex_attr_get
 Max Wrapper Module, 789

max_jit_obex_attr_set
 Max Wrapper Module, 789

max_jit_obex_dumpout
 Max Wrapper Module, 790

max_jit_obex_dumpout_get
 Max Wrapper Module, 790

max_jit_obex_dumpout_set
 Max Wrapper Module, 791

max_jit_obex_free
 Max Wrapper Module, 791

max_jit_obex_gimmeback
 Max Wrapper Module, 791

max_jit_obex_gimmeback_dumpout
 Max Wrapper Module, 792

max_jit_obex_inletnumber_get
 Max Wrapper Module, 792

max_jit_obex_inletnumber_set
 Max Wrapper Module, 793

max_jit_obex_jitob_get
 Max Wrapper Module, 793

max_jit_obex_jitob_set
 Max Wrapper Module, 793

max_jit_obex_new
 Max Wrapper Module, 795

max_jit_obex_proxy_new
 Max Wrapper Module, 795

max_ob3d_bang
 OB3D Module, 853

max_ob3d_notify
 OB3D Module, 854

maxversion
 Miscellaneous, 402

Memory Management, 383

- disposhandle, 384
- freebytes, 384
- freebytes16, 386
- getbytes, 386
- getbytes16, 386
- growhandle, 386
- MM_UNIFIED, 384
- newhandle, 387
- sysmem_copyptr, 387
- sysmem_freehandle, 387
- sysmem_freetpr, 387
- sysmem_handlesize, 388
- sysmem_lockhandle, 388
- sysmem_newhandle, 388
- sysmem_newhandleclear, 388
- sysmem_newptr, 389
- sysmem_newptrclear, 389

sysmem_nullterminatehandle, 389
sysmem_ptrrandhand, 389
sysmem_ptrbeforehand, 391
sysmem_ptrsize, 391
sysmem_resizehandle, 391
sysmem_resizeptr, 391
sysmem_resizeptrclear, 393
Memory Module, 797
 jit_copy_bytes, 797
 jit_disposeptr, 798
 jit_freebytes, 798
 jit_freemem, 799
 jit_getbytes, 799
 jit_handle_free, 800
 jit_handle_lock, 800
 jit_handle_new, 801
 jit_handle_size_get, 801
 jit_handle_size_set, 802
 jit_newptr, 802
Miscellaneous, 394
 aaCancel, 399
 aaNo, 399
 aaYes, 399
 BEGIN_USING_C_LINKAGE, 397
 CLAMP, 398
 calcoffset, 397
 classname_openrefpage_ext, 399
 e_max_errorcodes, 399
 e_max_wind_advise_result, 399
 error_subscribe, 400
 error_sym, 400
 error_unsubscribe, 400
 globalsymbol_bind, 400
 globalsymbol_dereference, 401
 globalsymbol_reference, 401
 globalsymbol_unbind, 401
 INRANGE, 398
 MAX, 398
 MAX_ERR_DUPLICATE, 399
 MAX_ERR_GENERIC, 399
 MAX_ERR_INVALID_PTR, 399
 MAX_ERR_NONE, 399
 MAX_ERR_OUT_OF_MEM, 399
 MIN, 398
 maxversion, 402
 object_obex_quickref, 402
 post_sym, 402
 quittask_install, 402
 quittask_remove, 402
 snprintf_zero, 404
 strncat_zero, 404
 strncpy_zero, 404
 structmembersize, 399
 symbol_stripquotes, 404
 symbol_unique, 404
 symbolarray_sort, 405
 wind_advise, 405
 wind_setcursor, 405
Miscellaneous Utility Module, 718
 jit_err_from_max_err, 718
 jit_error_code, 719
 jit_error_sym, 719
 jit_global_critical_enter, 719
 jit_global_critical_exit, 720
 jit_post_sym, 720
 jit_rand, 721
 jit_rand_setseed, 721
 swapf32, 721
 swapf64, 722
Monitors and Displays, 428
 jmonitor_getdisplayrect, 428
 jmonitor_getdisplayrect_foralldisplays, 428
 jmonitor_getdisplayrect_forpoint, 429
 jmonitor_getnumdisplays, 429
Mouse and Keyboard, 431
 eAltKey, 432
 eAutoRepeat, 432
 eCapsLock, 432
 eCommandKey, 432
 eControlKey, 432
 eLeftButton, 432
 eMiddleButton, 432
 ePopupMenu, 432
 eRightButton, 432
 eShiftKey, 432
 JMOUSE_CURSOR_ARROW, 432
 JMOUSE_CURSOR_COPYING, 432
 JMOUSE_CURSOR_CROSSHAIR, 432
 JMOUSE_CURSOR_DRAGGINGHAND, 432
 JMOUSE_CURSOR_IBeam, 432
 JMOUSE_CURSOR_NONE, 432
 JMOUSE_CURSOR_POINTINGHAND, 432
 JMOUSE_CURSOR_RESIZE_BOTTOMEDGE, 432
 JMOUSE_CURSOR_RESIZE_BOTTOMLEFTCORNER, 432
 JMOUSE_CURSOR_RESIZE_BOTTOMRIGHTCORNER, 432
 JMOUSE_CURSOR_RESIZE_FOURWAY, 432
 JMOUSE_CURSOR_RESIZE_LEFTEDGE, 432
 JMOUSE_CURSOR_RESIZE_LEFTRIGHT, 432
 JMOUSE_CURSOR_RESIZE_RIGHTEDGE, 432
 JMOUSE_CURSOR_RESIZE_TOPEDGE, 432
 JMOUSE_CURSOR_RESIZE_TOPLEFTCORNER, 432
 JMOUSE_CURSOR_RESIZE_UPDOWN, 432
 JMOUSE_CURSOR_WAIT, 432
 jkeyboard_getcurrentmodifiers, 433
 jkeyboard_getcurrentmodifiers_realtime, 433
 jmouse_getposition_global, 433
 jmouse_setcursor, 433
 jmouse_setposition_box, 433
 jmouse_setposition_global, 433
 jmouse_setposition_view, 434

t_jmouse_cursortype, 432
 t_modifiers, 432
 Mutexes, 572
 systhread_mutex_free, 572
 systhread_mutex_lock, 572
 systhread_mutex_new, 573
 systhread_mutex_newlock, 573
 systhread_mutex_trylock, 573
 systhread_mutex_unlock, 574

newhandle
 Memory Management, 387

newinstance
 Old-Style Classes, 218

newobject
 Old-Style Classes, 219

newobject_fromboxtext
 Objects, 453

newobject_fromdictionary
 Objects, 453

newobject_sprintf
 Objects, 454

OB3D Module, 838
 jit_gl_begincapture, 840
 jit_gl_bindtexture, 840
 jit_gl_drawinfo_active_textures, 842
 jit_gl_drawinfo_setup, 842
 jit_gl_endcapture, 842
 jit_gl_get_extensions, 842
 jit_gl_get_glsl_version, 842
 jit_gl_get_glu_version, 843
 jit_gl_get_renderer, 843
 jit_gl_get_vendor, 843
 jit_gl_get_version, 843
 jit_gl_is_extension_supported, 843
 jit_gl_is_min_version, 844
 jit_gl_report_error, 844
 jit_gl_texcoord1f, 845
 jit_gl_texcoord1fv, 845
 jit_gl_texcoord2f, 845
 jit_gl_texcoord2fv, 845
 jit_gl_texcoord3f, 846
 jit_gl_texcoord3fv, 846
 jit_gl_unbindtexture, 846
 jit_glchunk_copy, 846
 jit_glchunk_delete, 847
 jit_glchunk_grid_new, 847
 jit_glchunk_new, 848
 jit_ob3d_draw_chunk, 849
 jit_ob3d_free, 849
 jit_ob3d_new, 850
 jit_ob3d_set_context, 851
 jit_ob3d_setup, 851
 max jit_ob3d_assist, 852
 max jit_ob3d_attach, 852
 max jit_ob3d_detach, 853
 max ob3d_bang, 853
 max ob3d_notify, 854

ob3d_auto_get, 854
 ob3d_dest_dim_get, 854
 ob3d_dest_dim_set, 854
 ob3d_dirty_get, 855
 ob3d_dirty_set, 855
 ob3d_enable_get, 855
 ob3d_jitob_get, 855
 ob3d_outlet_get, 855
 ob3d_patcher_get, 856
 ob3d_render_ptr_get, 856
 ob3d_render_ptr_set, 856
 ob3d_ui_get, 856

OBEX_UTIL_ATOM_GETTEXT_COMMAS_DELIM
 Atoms, 324

OBEX_UTIL_ATOM_GETTEXT_DEFAULT
 Atoms, 324

OBEX_UTIL_ATOM_GETTEXT_FORCE_ZEROS
 Atoms, 324

OBEX_UTIL_ATOM_GETTEXT_NUM_HI_RES
 Atoms, 324

OBEX_UTIL_ATOM_GETTEXT_NUM_LO_RES
 Atoms, 324

OBEX_UTIL_ATOM_GETTEXT_SYM_FORCE_QUOTES
 Atoms, 324

OBEX_UTIL_ATOM_GETTEXT_SYM_NO_QUOTE
 Atoms, 324

OBEX_UTIL_ATOM_GETTEXT_TRUNCATE_ZEROS
 Atoms, 324

OBJ_ATTR_ATOM
 Attributes, 172

OBJ_ATTR_ATOM_ARRAY
 Attributes, 173

OBJ_ATTR_CHAR
 Attributes, 173

OBJ_ATTR_CHAR_ARRAY
 Attributes, 173

OBJ_ATTR_DEFAULT
 Attributes, 173

OBJ_ATTR_DEFAULT_SAVE
 Attributes, 174

OBJ_ATTR_DOUBLE
 Attributes, 174

OBJ_ATTR_FLOAT
 Attributes, 174

OBJ_ATTR_FLOAT_ARRAY
 Attributes, 174

OBJ_ATTR_LONG
 Attributes, 175

OBJ_ATTR_LONG_ARRAY
 Attributes, 175

OBJ_ATTR_OBJ
 Attributes, 175

OBJ_ATTR_OBJ_ARRAY
 Attributes, 175

OBJ_ATTR_SAVE

Attributes, 176
OBJ_ATTR_SYM
 Attributes, 176
OBJ_ATTR_SYM_ARRAY
 Attributes, 176
OBJ_FLAG_DATA
 Data Storage, 230
OBJ_FLAG_DEBUG
 Data Storage, 230
OBJ_FLAG_INHERITABLE
 Data Storage, 230
OBJ_FLAG_ITERATING
 Data Storage, 230
OBJ_FLAG_MEMORY
 Data Storage, 230
OBJ_FLAG_OBJ
 Data Storage, 230
OBJ_FLAG_REF
 Data Storage, 230
OBJ_FLAG_SILENT
 Data Storage, 230
ob3d_auto_get
 OB3D Module, 854
ob3d_dest_dim_get
 OB3D Module, 854
ob3d_dest_dim_set
 OB3D Module, 854
ob3d_dirty_get
 OB3D Module, 855
ob3d_dirty_set
 OB3D Module, 855
ob3d_enable_get
 OB3D Module, 855
ob3d_jitob_get
 OB3D Module, 855
ob3d_outlet_get
 OB3D Module, 855
ob3d_patcher_get
 OB3D Module, 856
ob3d_render_ptr_get
 OB3D Module, 856
ob3d_render_ptr_set
 OB3D Module, 856
ob3d_ui_get
 OB3D Module, 856
Object Module, 704
 jit_object_attach, 705
 jit_object_attr_get, 705
 jit_object_attr_usercanget, 706
 jit_object_attr_usercanset, 706
 jit_object_class, 707
 jit_object_classname, 707
 jit_object_classname_compare, 708
 jit_object_detach, 708
 jit_object_exportattrs, 709
 jit_object_exportsummary, 710
 jit_object_findregistered, 710
 jit_object_findregisteredbyptr, 711
 jit_object_free, 711
 jit_object_getmethod, 712
 jit_object_importattrs, 712
 jit_object_method_argsafe_get, 713
 jit_object_method_imp, 714
 jit_object_method_typed, 714
 jit_object_new_imp, 715
 jit_object_notify, 715
 jit_object_register, 716
 jit_object_unregister, 716
object_addattr
 Attributes, 191
object_alloc
 Objects, 454
object_attach
 Objects, 454
object_attach_byptr
 Objects, 455
object_attach_byptr_register
 Objects, 456
object_attr_get
 Attributes, 191
object_attr_get_rect
 Attributes, 192
object_attr_getchar_array
 Attributes, 192
object_attr_getcolor
 Attributes, 192
object_attr_getdouble_array
 Attributes, 194
object_attr_getdump
 Attributes, 194
object_attr_getfill
 Styles, 626
object_attr_getfillcolor_atposition
 Styles, 626
object_attr_getfloat
 Attributes, 194
object_attr_getfloat_array
 Attributes, 195
object_attr_getjrgba
 Attributes, 195
object_attr_getlong
 Attributes, 195
object_attr_getlong_array
 Attributes, 196
object_attr_getpt
 Attributes, 196
object_attr_getsize
 Attributes, 196
object_attr_getsym
 Attributes, 197
object_attr_getsym_array
 Attributes, 197
object_attr_method
 Attributes, 197
object_attr_set_rect
 Attributes, 198

object_attr_set_xywh
 Attributes, 198
object_attr_setchar_array
 Attributes, 198
object_attr_setcolor
 Attributes, 199
object_attr_setdouble_array
 Attributes, 199
object_attr_setfloat
 Attributes, 199
object_attr_setfloat_array
 Attributes, 200
object_attr_setjrgba
 Attributes, 200
object_attr_setlong
 Attributes, 200
object_attr_setlong_array
 Attributes, 201
object_attr_setparse
 Attributes, 201
object_attr_setpt
 Attributes, 201
object_attr_setsize
 Attributes, 202
object_attr_setsym
 Attributes, 202
object_attr_setsym_array
 Attributes, 202
object_attr_setvalueof
 Attributes, 203
object_attr_touch
 Objects, 456
object_attr_touch_parse
 Objects, 456
object_attr_usercanget
 Attributes, 203
object_attr_usercanset
 Attributes, 203
object_chuckattr
 Attributes, 203
object_class
 Objects, 456
object_classname
 Objects, 458
object_classname_compare
 Objects, 458
object_deleteattr
 Attributes, 204
object_detach
 Objects, 458
object_detach_byptr
 Objects, 459
object_dictionaryarg
 Objects, 459
object_error
 Console, 408
object_error_obtrusive
 Console, 409
object_findregistered
 Objects, 459
object_findregisteredbyptr
 Objects, 459
object_free
 Objects, 461
object_getmethod
 Objects, 461
object_getvalueof
 Objects, 461
object_method
 Objects, 462
object_method_char
 Objects, 462
object_method_char_array
 Objects, 463
object_method_direct
 Objects, 452
object_method_double
 Objects, 463
object_method_double_array
 Objects, 463
object_method_float
 Objects, 464
object_method_float_array
 Objects, 464
object_method_format
 Objects, 464
object_method_long
 Objects, 465
object_method_long_array
 Objects, 465
object_method_obj
 Objects, 465
object_method_obj_array
 Objects, 467
object_method_parse
 Objects, 467
object_method_sym
 Objects, 467
object_method_sym_array
 Objects, 468
object_method_typed
 Objects, 468
object_method_typedfun
 Objects, 468
object_new
 Objects, 470
object_new_parse
 Attributes, 204
object_new_typed
 Objects, 470
object_notify
 Objects, 470
object_obex_dumpout
 Objects, 472
object_obex_lookup
 Objects, 472

object_obex_quickref
 Miscellaneous, 402

object_obex_store
 Objects, 473

object_openhelp
 Objects, 473

object_openquery
 Objects, 474

object_openrefpage
 Objects, 474

object_post
 Console, 409

object_register
 Objects, 474

object_register_getnames
 Objects, 474

object_setvalueof
 Objects, 475

object_subscribe
 Objects, 475

object_super_method
 Objects, 475

object_this_method
 Objects, 476

object_unregister
 Objects, 476

object_unsubscribe
 Objects, 476

object_warn
 Console, 409

Objects, 449

 classname_openhelp, 452

 classname_openquery, 452

 classname_openrefpage, 452

 MSG_MAXARG, 452

 newobject_fromboxtext, 453

 newobject_fromdictionary, 453

 newobject_sprintf, 454

 object_alloc, 454

 object_attach, 454

 object_attach_byptr, 455

 object_attach_byptr_register, 456

 object_attr_touch, 456

 object_attr_touch_parse, 456

 object_class, 456

 object_classname, 458

 object_classname_compare, 458

 object_detach, 458

 object_detach_byptr, 459

 object_dictionaryarg, 459

 object_findregistered, 459

 object_findregisteredbyptr, 459

 object_free, 461

 object_getmethod, 461

 object_getvalueof, 461

 object_method, 462

 object_method_char, 462

 object_method_char_array, 463

 object_method_direct, 452

 object_method_double, 463

 object_method_double_array, 463

 object_method_float, 464

 object_method_float_array, 464

 object_method_format, 464

 object_method_long, 465

 object_method_long_array, 465

 object_method_obj, 465

 object_method_obj_array, 467

 object_method_parse, 467

 object_method_sym, 467

 object_method_sym_array, 468

 object_method_typed, 468

 object_method_typedfun, 468

 object_new, 470

 object_new_typed, 470

 object_notify, 470

 object_obex_dumpout, 472

 object_obex_lookup, 472

 object_obex_store, 473

 object_openhelp, 473

 object_openquery, 474

 object_openrefpage, 474

 object_register, 474

 object_register_getnames, 474

 object_setvalueof, 475

 object_subscribe, 475

 object_super_method, 475

 object_this_method, 476

 object_unregister, 476

 object_unsubscribe, 476

Old-Style Classes, 215

 addbang, 216

 addfloat, 216

 addftx, 216

 addint, 216

 addinx, 216

 addmess, 217

 alias, 217

 class_setname, 217

 egetfn, 217

 freeobject, 218

 getfn, 218

 newinstance, 218

 newobject, 219

 setup, 219

 typedmess, 220

 zgetfn, 220

open_dialog
 Files and Folders, 360

open_promptset
 Files and Folders, 360

Operator Vector Module, 857

 jit_op_vector_abs_float32, 865

 jit_op_vector_abs_float64, 865

 jit_op_vector_abs_long, 865

 jit_op_vector_absdiff_char, 866

jit_op_vector_absdiff_float32, 866
jit_op_vector_absdiff_float64, 866
jit_op_vector_absdiff_long, 866
jit_op_vector_acos_float32, 867
jit_op_vector_acos_float64, 867
jit_op_vector_acosh_float32, 867
jit_op_vector_acosh_float64, 867
jit_op_vector_add_char, 868
jit_op_vector_add_float32, 868
jit_op_vector_add_float64, 868
jit_op_vector_add_long, 868
jit_op_vector_adds_char, 869
jit_op_vector_and_char, 869
jit_op_vector_and_float32, 869
jit_op_vector_and_float64, 869
jit_op_vector_and_long, 870
jit_op_vector_asin_float32, 870
jit_op_vector_asin_float64, 870
jit_op_vector_asinh_float32, 870
jit_op_vector_asinh_float64, 871
jit_op_vector_atan2_float32, 871
jit_op_vector_atan2_float64, 871
jit_op_vector_atan_float32, 871
jit_op_vector_atan_float64, 872
jit_op_vector_atanh_float32, 872
jit_op_vector_atanh_float64, 872
jit_op_vector_avg_char, 872
jit_op_vector_avg_float32, 873
jit_op_vector_avg_float64, 873
jit_op_vector_avg_long, 873
jit_op_vector_bitand_char, 873
jit_op_vector_bitand_long, 874
jit_op_vector_bitnot_char, 874
jit_op_vector_bitnot_long, 874
jit_op_vector_bitor_char, 874
jit_op_vector_bitor_long, 875
jit_op_vector_bitxor_char, 875
jit_op_vector_bitxor_long, 875
jit_op_vector_ceil_float32, 875
jit_op_vector_ceil_float64, 876
jit_op_vector_cos_float32, 876
jit_op_vector_cos_float64, 876
jit_op_vector_cosh_float32, 876
jit_op_vector_cosh_float64, 877
jit_op_vector_div_char, 877
jit_op_vector_div_float32, 877
jit_op_vector_div_float64, 877
jit_op_vector_div_long, 878
jit_op_vector_eq_char, 878
jit_op_vector_eq_float32, 878
jit_op_vector_eq_float64, 878
jit_op_vector_eq_long, 879
jit_op_vector_eqp_char, 879
jit_op_vector_eqp_float32, 879
jit_op_vector_eqp_float64, 879
jit_op_vector_eqp_long, 880
jit_op_vector_exp2_float32, 880
jit_op_vector_exp2_float64, 880
jit_op_vector_exp_float32, 880
jit_op_vector_exp_float64, 881
jit_op_vector_flipdiv_char, 881
jit_op_vector_flipdiv_float32, 881
jit_op_vector_flipdiv_float64, 882
jit_op_vector_flipdiv_long, 882
jit_op_vector_flipmod_char, 883
jit_op_vector_flipmod_float32, 883
jit_op_vector_flipmod_float64, 884
jit_op_vector_flipmod_long, 884
jit_op_vector_flippass_char, 885
jit_op_vector_flippass_float32, 885
jit_op_vector_flippass_float64, 886
jit_op_vector_flippass_long, 886
jit_op_vector_flipsub_char, 887
jit_op_vector_flipsub_float32, 887
jit_op_vector_flipsub_long, 888
jit_op_vector_floor_float32, 888
jit_op_vector_floor_float64, 889
jit_op_vector_fold_float32, 889
jit_op_vector_fold_float64, 889
jit_op_vector_gt_char, 889
jit_op_vector_gt_float32, 890
jit_op_vector_gt_float64, 890
jit_op_vector_gt_long, 890
jit_op_vector_gte_char, 890
jit_op_vector_gte_float32, 891
jit_op_vector_gte_float64, 891
jit_op_vector_gte_long, 891
jit_op_vector_gtep_char, 891
jit_op_vector_gtep_float32, 892
jit_op_vector_gtep_float64, 892
jit_op_vector_gtep_long, 892
jit_op_vector_gtp_char, 892
jit_op_vector_gtp_float32, 893
jit_op_vector_gtp_float64, 893
jit_op_vector_gtp_long, 893
jit_op_vector_hypot_float32, 893
jit_op_vector_hypot_float64, 894
jit_op_vector_log10_float32, 894
jit_op_vector_log10_float64, 894
jit_op_vector_log2_float32, 894
jit_op_vector_log2_float64, 895
jit_op_vector_log_float32, 895
jit_op_vector_log_float64, 895
jit_op_vector_lshift_char, 895
jit_op_vector_lshift_long, 896
jit_op_vector_lt_char, 896
jit_op_vector_lt_float32, 896
jit_op_vector_lt_float64, 896
jit_op_vector_lt_long, 897
jit_op_vector_lte_char, 897
jit_op_vector_lte_float32, 897
jit_op_vector_lte_float64, 897
jit_op_vector_lte_long, 898
jit_op_vector_ltep_char, 898
jit_op_vector_ltep_float32, 898
jit_op_vector_ltep_float64, 898

jit_op_vector_ltep_long, 899
jit_op_vector_ltp_char, 899
jit_op_vector_ltp_float32, 899
jit_op_vector_ltp_float64, 899
jit_op_vector_ltp_long, 900
jit_op_vector_max_char, 900
jit_op_vector_max_float32, 900
jit_op_vector_max_float64, 900
jit_op_vector_max_long, 901
jit_op_vector_min_char, 901
jit_op_vector_min_float32, 901
jit_op_vector_min_float64, 901
jit_op_vector_min_long, 902
jit_op_vector_mod_char, 902
jit_op_vector_mod_float32, 902
jit_op_vector_mod_float64, 902
jit_op_vector_mod_long, 903
jit_op_vector_mult_char, 903
jit_op_vector_mult_float32, 903
jit_op_vector_mult_float64, 903
jit_op_vector_mult_long, 904
jit_op_vector_neq_char, 904
jit_op_vector_neq_float32, 904
jit_op_vector_neq_float64, 904
jit_op_vector_neq_long, 905
jit_op_vector_neqp_char, 905
jit_op_vector_neqp_float32, 905
jit_op_vector_neqp_float64, 905
jit_op_vector_neqp_long, 906
jit_op_vector_not_char, 906
jit_op_vector_not_float32, 906
jit_op_vector_not_float64, 906
jit_op_vector_not_long, 907
jit_op_vector_or_char, 907
jit_op_vector_or_float32, 907
jit_op_vector_or_float64, 907
jit_op_vector_or_long, 908
jit_op_vector_pass_char, 908
jit_op_vector_pass_float32, 908
jit_op_vector_pass_float64, 908
jit_op_vector_pass_long, 909
jit_op_vector_pow_float32, 909
jit_op_vector_pow_float64, 909
jit_op_vector_round_float32, 909
jit_op_vector_round_float64, 910
jit_op_vector_rshift_char, 910
jit_op_vector_rshift_long, 910
jit_op_vector_sin_float32, 910
jit_op_vector_sin_float64, 911
jit_op_vector_sinh_float32, 911
jit_op_vector_sinh_float64, 911
jit_op_vector_sqrt_float32, 911
jit_op_vector_sqrt_float64, 912
jit_op_vector_sub_char, 912
jit_op_vector_sub_float32, 912
jit_op_vector_sub_float64, 912
jit_op_vector_sub_long, 913
jit_op_vector_subs_char, 913
jit_op_vector_tan_float32, 913
jit_op_vector_tan_float64, 913
jit_op_vector_tanh_float32, 914
jit_op_vector_tanh_float64, 914
jit_op_vector_trunc_float32, 914
jit_op_vector_trunc_float64, 914
jit_op_vector_wrap_float32, 915
jit_op_vector_wrap_float64, 915
ouchstring
 Console, 410
outlet_anything
 Inlets and Outlets, 224
outlet_bang
 Inlets and Outlets, 224
outlet_float
 Inlets and Outlets, 225
outlet_int
 Inlets and Outlets, 225
outlet_list
 Inlets and Outlets, 225
outlet_new
 Inlets and Outlets, 226

PATH_FILEINFO_ALIAS
 Files and Folders, 356
PATH_FILEINFO_FOLDER
 Files and Folders, 356
PATH_FILEINFO_PACKAGE
 Files and Folders, 356
PATH_FOLDER_SNIFF
 Files and Folders, 356
PATH_NOALIASRESOLUTION
 Files and Folders, 356
PATH_READ_PERM
 Files and Folders, 356
PATH_REPORTPACKAGEASFOLDER
 Files and Folders, 356
PATH_RW_PERM
 Files and Folders, 356
PATH_STYLE_COLON
 Files and Folders, 356
PATH_STYLE_MAX
 Files and Folders, 356
PATH_STYLE_NATIVE
 Files and Folders, 356
PATH_STYLE_NATIVE_WIN
 Files and Folders, 356
PATH_STYLE_SLASH
 Files and Folders, 356
PATH_TYPE_ABSOLUTE
 Files and Folders, 357
PATH_TYPE_BOOT
 Files and Folders, 357
PATH_TYPE_C74
 Files and Folders, 357
PATH_TYPE_DESKTOP
 Files and Folders, 357
PATH_TYPE_IGNORE
 Files and Folders, 357

PATH_TYPE_MAXDB
 Files and Folders, 357

PATH_TYPE_PATH
 Files and Folders, 357

PATH_TYPE_RELATIVE
 Files and Folders, 357

PATH_TYPE_TEMPFOLDER
 Files and Folders, 357

PATH_TYPE_TILDE
 Files and Folders, 357

PATH_WRITE_PERM
 Files and Folders, 356

PFFT, 447

PI
 MSP, 437

PI_DEEP
 Patcher, 479

PI_REQUIREFIRSTIN
 Patcher, 479

PI_WANTBOX
 Patcher, 479

PIOVERTWO
 MSP, 437

Parallel Utility Module, 819
 jit_parallel_ndim_calc, 819
 jit_parallel_ndim_simplecalc1, 820
 jit_parallel_ndim_simplecalc2, 820
 jit_parallel_ndim_simplecalc3, 821
 jit_parallel_ndim_simplecalc4, 822

Patcher, 478
 PI_DEEP, 479
 PI_REQUIREFIRSTIN, 479
 PI_WANTBOX, 479
 t_box, 479
 t_patcher, 479

patcherview_canvas_to_screen
 jpatcherview, 523

patcherview_findpatcherview
 jpatcherview, 523

patcherview_get_jgraphics
 jpatcherview, 523

patcherview_get_locked
 jpatcherview, 523

patcherview_get_nextview
 jpatcherview, 523

patcherview_get_patcher
 jpatcherview, 524

patcherview_get_presentation
 jpatcherview, 524

patcherview_get_rect
 jpatcherview, 524

patcherview_get_topview
 jpatcherview, 524

patcherview_get_visible
 jpatcherview, 525

patcherview_get_zoomfactor
 jpatcherview, 525

patcherview_screen_to_canvas
 jpatcherview, 525

patcherview_set_locked
 jpatcherview, 525

patcherview_set_presentation
 jpatcherview, 525

patcherview_set_rect
 jpatcherview, 527

patcherview_set_visible
 jpatcherview, 527

patcherview_set_zoomfactor
 jpatcherview, 527

path_absolutepath
 Files and Folders, 361

path_closefolder
 Files and Folders, 361

path_createsysfile
 Files and Folders, 361

path_fileinfo
 Files and Folders, 362

path_foldernextfile
 Files and Folders, 362

path_frompathname
 Files and Folders, 362

path_getapppath
 Files and Folders, 363

path_getdefault
 Files and Folders, 363

path_getfilemoddate
 Files and Folders, 363

path_getmoddate
 Files and Folders, 363

path_nameconform
 Files and Folders, 363

path_openfolder
 Files and Folders, 364

path_opensysfile
 Files and Folders, 364

path_resolvefile
 Files and Folders, 364

path_setdefault
 Files and Folders, 365

path_toabsolutestystempath
 Files and Folders, 365

path_top pathname
 Files and Folders, 365

path_topotentialname
 Files and Folders, 366

Poly, 448

Popup Menus, 641
 jpopupmenu_additem, 642
 jpopupmenu_addseperator, 642
 jpopupmenu_addsubmenu, 642
 jpopupmenu_clear, 642
 jpopupmenu_create, 642
 jpopupmenu_destroy, 642
 jpopupmenu_popup, 644
 jpopupmenu_popup_abovebox, 644
 jpopupmenu_popup_aboverect, 644

jpopupmenu_popup_belowrect, 644
jpopupmenu_popup_nearbox, 645
jpopupmenu_setcolors, 645
jpopupmenu_setfont, 645
post
 Console, 410
post_sym
 Miscellaneous, 402
postargs
 Atoms, 340
postatom
 Console, 411
postdictionary
 Dictionary, 271
preset_int
 Presets, 422
preset_set
 Presets, 422
preset_store
 Presets, 422
Presets, 421
 preset_int, 422
 preset_set, 422
 preset_store, 422
proxy_getinlet
 Inlets and Outlets, 226
proxy_new
 Inlets and Outlets, 226

qelem_free
 Qelems, 540
qelem_front
 Qelems, 540
qelem_new
 Qelems, 540
qelem_set
 Qelems, 540
qelem_unset
 Qelems, 541
Qelems, 539
 qelem_free, 540
 qelem_front, 540
 qelem_new, 540
 qelem_set, 540
 qelem_unset, 541
Quick Map, 303
 quickmap_add, 303
 quickmap_drop, 304
 quickmap_lookup_key1, 304
 quickmap_lookup_key2, 304
 quickmap_new, 304
 quickmap_READONLY, 305
quickmap_add
 Quick Map, 303
quickmap_drop
 Quick Map, 304
quickmap_lookup_key1
 Quick Map, 304
quickmap_lookup_key2

 Quick Map, 304
 Quick Map, 305
 Miscellaneous, 402
 Miscellaneous, 402
readatom
 Binbufs, 347
readtohandle
 Loading Max Files, 427

STRUCT_ATTR_ATOM
 Attributes, 176
STRUCT_ATTR_ATOM_ARRAY
 Attributes, 177
STRUCT_ATTR_ATOM_LONG
 Attributes, 177
STRUCT_ATTR_ATOM_VARSIZE
 Attributes, 177
STRUCT_ATTR_CHAR
 Attributes, 177
STRUCT_ATTR_CHAR_ARRAY
 Attributes, 178
STRUCT_ATTR_CHAR_VARSIZE
 Attributes, 178
STRUCT_ATTR_DOUBLE
 Attributes, 178
STRUCT_ATTR_DOUBLE_ARRAY
 Attributes, 178
STRUCT_ATTR_DOUBLE_VARSIZE
 Attributes, 180
STRUCT_ATTR_FLOAT
 Attributes, 180
STRUCT_ATTR_FLOAT_ARRAY
 Attributes, 180
STRUCT_ATTR_FLOAT_VARSIZE
 Attributes, 180
STRUCT_ATTR_LONG
 Attributes, 181
STRUCT_ATTR_LONG_ARRAY
 Attributes, 181
STRUCT_ATTR_LONG_VARSIZE
 Attributes, 181
STRUCT_ATTR_OBJ
 Attributes, 182
STRUCT_ATTR_OBJ_ARRAY
 Attributes, 182
STRUCT_ATTR_OBJ_VARSIZE
 Attributes, 182
STRUCT_ATTR_SYM
 Attributes, 182
STRUCT_ATTR_SYM_ARRAY
 Attributes, 183
STRUCT_ATTR_SYM_VARSIZE
 Attributes, 183

SYS_MAXBLKSIZE
 MSP, 438

SYS_MAXSIGS
 MSP, 438

SYSDATEFORMAT_FLAGS_LONG
 Systime API, 543

SYSDATEFORMAT_FLAGS_MEDIUM
 Systime API, 543

SYSDATEFORMAT_FLAGS_SHORT
 Systime API, 543

SYSFILE_ATMARK
 Files and Folders, 357

SYSFILE_FROMLEOF
 Files and Folders, 357

SYSFILE_FROMMARK
 Files and Folders, 357

SYSFILE_FROMSTART
 Files and Folders, 357

SYSTHREAD_MUTEX_ERRORCHECK
 Threads, 561

SYSTHREAD_MUTEX_NORMAL
 Threads, 561

SYSTHREAD_MUTEX_RECURSIVE
 Threads, 561

saveas_dialog
 Files and Folders, 366

saveas_promptset
 Files and Folders, 366

saveasdialog_extended
 Files and Folders, 367

Scalable Vector Graphics, 602
 jsvg_create_from_file, 602
 jsvg_create_from_resource, 602
 jsvg_create_from_xmlstring, 603
 jsvg_destroy, 603
 jsvg_get_size, 603
 jsvg_render, 603

schedule
 Threads, 563

schedule_delay
 Threads, 564

scheduler_fromobject
 Clocks, 534

scheduler_get
 Clocks, 534

scheduler_gettime
 Clocks, 534

scheduler_new
 Clocks, 535

scheduler_run
 Clocks, 535

scheduler_set
 Clocks, 535

scheduler_settime
 Clocks, 535

scheduler_shift
 Clocks, 536

setclock_delay
 Clocks, 536

setclock_fdelay
 Clocks, 536

setclock_gettime
 Clocks, 536

setclock_gettime
 Clocks, 538

setclock_unset
 Clocks, 538

setup
 Old-Style Classes, 219

snprintf_zero
 Miscellaneous, 404

String Object, 306
 string_append, 306
 string_chop, 307
 string_getptr, 307
 string_new, 307
 string_reserve, 307

string_append
 String Object, 306

string_chop
 String Object, 307

string_getptr
 String Object, 307

string_new
 String Object, 307

string_reserve
 String Object, 307

stringload
 Loading Max Files, 427

strncat_zero
 Miscellaneous, 404

strncpy_zero
 Miscellaneous, 404

structmembersize
 Miscellaneous, 399

Styles, 620
 CLASS_ATTR_STYLE_ALIAS_COMPATIBILITY, 621
 CLASS_ATTR_STYLE_ALIAS_NOSAVE, 621
 CLASS_ATTR_STYLE_ALIAS_RGBA_LEGACY, 621
 CLASS_ATTR_STYLE_RGBA, 621
 CLASS_ATTR_STYLE_RGBA_NOSAVE, 622
 CLASS_ATTR_STYLE_RGBA_PREVIEW, 622
 class_attr_setfill, 623
 class_attr_setstyle, 623
 class_attr_style_alias, 623
 class_attr_stylemap, 623
 FILL_ATTR_SAVE, 622
 jgraphics_attr_fillrect, 625
 jgraphics_attr_setfill, 625
 object_attr_getfill, 626
 object_attr_getfillcolor_atposition, 626

swapf32
 Miscellaneous Utility Module, 721

swapf64

Miscellaneous Utility Module, 722
Symbol Object, 308
 symobject_linklist_match, 308
 symobject_new, 309
symbol_stripquotes
 Miscellaneous, 404
symbol_unique
 Miscellaneous, 404
symbolarray_sort
 Miscellaneous, 405
Symbols, 348
 gensym, 349
 gensym_tr, 350
symobject_linklist_match
 Symbol Object, 308
symobject_new
 Symbol Object, 309
sys_getblksize
 MSP, 439
sys_getdspobjdspstate
 MSP, 439
sys_getdspstate
 MSP, 439
sys_getmaxblksize
 MSP, 439
sys_getstr
 MSP, 440
sysdateformat_formatdatetime
 Systime API, 543
sysdateformat_strftimetodatetime
 Systime API, 543
sysfile_close
 Files and Folders, 368
sysfile_geteof
 Files and Folders, 368
sysfile_getpos
 Files and Folders, 368
sysfile_openhandle
 Files and Folders, 369
sysfile_openptrsize
 Files and Folders, 369
sysfile_read
 Files and Folders, 369
sysfile_readtextfile
 Files and Folders, 370
sysfile_readtohandle
 Files and Folders, 370
sysfile_readtoptr
 Files and Folders, 370
sysfile_seteof
 Files and Folders, 371
sysfile_setpos
 Files and Folders, 371
sysfile_spoolcopy
 Files and Folders, 371
sysfile_write
 Files and Folders, 372
sysfile_writetextfile
 Files and Folders, 372
Files and Folders, 372
sysmem_copyptr
 Memory Management, 387
sysmem_freehandle
 Memory Management, 387
sysmem_freeptr
 Memory Management, 387
sysmem_handlesize
 Memory Management, 388
sysmem_lockhandle
 Memory Management, 388
sysmem_newhandle
 Memory Management, 388
sysmem_newhandleclear
 Memory Management, 388
sysmem_newptr
 Memory Management, 389
sysmem_newptrclear
 Memory Management, 389
sysmem_nullterminatehandle
 Memory Management, 389
sysmem_ptrandhand
 Memory Management, 389
sysmem_ptrbeforehand
 Memory Management, 391
sysmem_ptrsize
 Memory Management, 391
sysmem_resizehandle
 Memory Management, 391
sysmem_resizeptr
 Memory Management, 391
sysmem_resizeptrclear
 Memory Management, 393
systemfontname
 JFont, 610
systemfontname_bold
 JFont, 610
systemfontname_light
 JFont, 610
systemfontsym
 JFont, 610
systhread_create
 Threads, 564
systhread_detach
 Threads, 566
systhread_exit
 Threads, 566
systhread_getpriority
 Threads, 566
systhread_isaudiothread
 Threads, 566
systhread_ismainthread
 Threads, 567
systhread_istimerthread
 Threads, 567
systhread_join
 Threads, 567
systhread_mutex_free

Mutexes, 572
`systhread_mutex_lock`
 Mutexes, 572
`systhread_mutex_new`
 Mutexes, 573
`systhread_mutex_newlock`
 Mutexes, 573
`systhread_mutex_trylock`
 Mutexes, 573
`systhread_mutex_unlock`
 Mutexes, 574
`systhread_self`
 Threads, 567
`systhread_setpriority`
 Threads, 567
`systhread_sleep`
 Threads, 568
`systhread_terminate`
 Threads, 568
 Systime API, 542
 `e_max_dateflags`, 543
 `SYSDATEFORMAT_FLAGS_LONG`, 543
 `SYSDATEFORMAT_FLAGS_MEDIUM`, 543
 `SYSDATEFORMAT_FLAGS_SHORT`, 543
 `sysdateformat_formatdatetime`, 543
 `sysdateformat_strftimetodatetime`, 543
 `systime_datetime`, 543
 `systime_datetime_milliseconds`, 543
 `systime_datetoseconds`, 543
 `systime_ms`, 545
 `systime_seconds`, 545
 `systime_secondstodate`, 545
 `systime_ticks`, 545
 `systime_datetime`
 Systime API, 543
 `systime_datetime_milliseconds`
 Systime API, 543
 `systime_datetoseconds`
 Systime API, 543
 `systime_ms`
 Systime API, 545
 `systime_seconds`
 Systime API, 545
 `systime_secondstodate`
 Systime API, 545
 `systime_ticks`
 Systime API, 545
 `systimer_gettime`
 Clocks, 538

`t_atom`, 917
`t_atomarray`, 917
`t_atombuf`, 918
`t_attr`, 919
`t_box`
 Patcher, 479
`t_buffer`, 920
`t_buffer_info`, 923
`t_buffer_obj`

 Buffers, 442
`t_buffer_ref`
 Buffers, 442
`t_celldesc`, 924
`t_charset_converter`, 924
`t_class`, 924
`t_cmpfn`
 Data Storage, 229
`t_database`
 Database, 240
`t_datetime`, 925
`t_db_result`
 Database, 240
`t_db_view`
 Database, 241
`t_dictionary`, 926
`t_dictionary_entry`, 927
`t_double`
 MSP, 437
`t_ex_ex`, 927
`t_expr`, 928
`t_filehandle`
 Files and Folders, 355
`t_fileinfo`, 928
`t_float`
 MSP, 437
`t_funbuff`, 929
`t_hashtab`, 930
`t_hashtab_entry`, 931
`t_indexmap`, 932
`t_indexmap_entry`, 933
`t_itm`
 ITM Time Objects, 558
`t_jbox`, 933
`t_jboxdrawparams`, 934
`t_jcolumn`, 934
`t_jdataview`, 937
`t_jgraphics_fileformat`
 JGraphics, 581
`t_jgraphics_font_extents`, 940
`t_jgraphics_font_slant`
 JFont, 605
`t_jgraphics_font_weight`
 JFont, 605
`t_jgraphics_format`
 JGraphics, 581
`t_jgraphics_text_justification`
 JGraphics, 581
`t_jgraphics_textlayout_flags`
 TextLayout, 637
`t_jit_attr`, 941
`t_jit_attr_filter_clip`, 942
`t_jit_attr_filter_proc`, 943
`t_jit_attr_offset`, 944
`t_jit_attr_offset_array`, 945
`t_jit_attribute`, 946
`t_jit_gl_context_view`, 948
`t_jit_gl_drawinfo`, 950

t_jit_glchunk, 950
t_jit_matrix_info, 951
t_jit_mop, 952
t_jit_mop_io, 954
t_jit_op_info, 955
t_jmatrix, 955
t_jmouse_cursortype
 Mouse and Keyboard, 432
t_jrgb, 956
t_jrgba, 956
t_line_3d, 957
t_linklist, 957
t_llelem, 958
t_matrix_conv_info, 959
t_messlist, 960
t_modifiers
 Mouse and Keyboard, 432
t_object, 960
t_package_file, 960
t_patcher
 Patcher, 479
t_path, 961
t_pathlink, 961
t_perfroutine
 MSP, 440
t_pfftpub, 962
t_privatesortrec, 962
t_pt, 964
t_pxdata, 964
t_pxjbox, 964
t_pxobject, 965
t_quickmap, 966
t_rect, 967
t_sample
 MSP, 437
t_signal, 968
t_size, 968
t_string, 969
t_symbol, 969
t_symobject, 970
t_sysfile_pos_mode
 Files and Folders, 357
t_sysfile_text_flags
 Files and Folders, 357
t_timeobject
 ITM Time Objects, 558
t_tinyobject, 971
t_vptr
 MSP, 437
t_wind_mouse_info, 972
t_zll, 972
TEXT_ENCODING_USE_FILE
 Files and Folders, 357
TEXT_LB_MAC
 Files and Folders, 357
TEXT_LB_NATIVE
 Files and Folders, 357
TEXT_LB_PC
 Files and Folders, 357
 Files and Folders, 357
TEXT_LB_UNIX
 Files and Folders, 357
TEXT_NULL_TERMINATE
 Files and Folders, 357
TIME_FLAGS_BBUSOURCE
 ITM Time Objects, 548
TIME_FLAGS_CHECKSCHEDULE
 ITM Time Objects, 548
TIME_FLAGS_EVENTLIST
 ITM Time Objects, 548
TIME_FLAGS_FIXED
 ITM Time Objects, 548
TIME_FLAGS_FIXEDONLY
 ITM Time Objects, 548
TIME_FLAGS_LISTENTICKS
 ITM Time Objects, 548
TIME_FLAGS_LOCATION
 ITM Time Objects, 548
TIME_FLAGS_LOOKAHEAD
 ITM Time Objects, 548
TIME_FLAGS_NOUNITS
 ITM Time Objects, 548
TIME_FLAGS_PERMANENT
 ITM Time Objects, 548
TIME_FLAGS_POSITIVE
 ITM Time Objects, 548
TIME_FLAGS_TICKSONLY
 ITM Time Objects, 548
TIME_FLAGS_TRANSPORT
 ITM Time Objects, 548
TIME_FLAGS_USECLOCK
 ITM Time Objects, 548
TIME_FLAGS_USEQELEM
 ITM Time Objects, 548
TWOPI
 MSP, 437
Table Access, 418
 table_dirty, 418
 table_get, 418
table_dirty
 Table Access, 418
table_get
 Table Access, 418
Text Editor Windows, 420
TextField, 627
 textfield_get_autoscroll, 628
 textfield_get_bgcolor, 628
 textfield_get_editonclick, 629
 textfield_get_emptytext, 629
 textfield_get_noactivate, 629
 textfield_get_owner, 629
 textfield_get_readonly, 630
 textfield_get_selectallonedit, 630
 textfield_get_textcolor, 630
 textfield_get_textmargins, 630
 textfield_get_underline, 631
 textfield_get_useellipsis, 631

textfield_get_wantsreturn, 631
textfield_get_wantstab, 631
textfield_get_wordwrap, 632
textfield_set_autoscroll, 632
textfield_set_bgcolor, 632
textfield_set_editonclick, 632
textfield_set_emptytext, 633
textfield_set_noactivate, 633
textfield_set_READONLY, 633
textfield_set_selectallonedit, 633
textfield_set_textcolor, 634
textfield_set_textmargins, 634
textfield_set_underline, 634
textfield_set_useellipsis, 634
textfield_set_wantsreturn, 635
textfield_set_wantstab, 635
textfield_set_wordwrap, 635

TextLayout, 636

JGRAPHICS_TEXTLAYOUT_NOWRAP, 637
JGRAPHICS_TEXTLAYOUT_USEELLIPSIS, 637

jtextlayout_create, 637
jtextlayout_createpath, 637
jtextlayout_destroy, 637
jtextlayout_draw, 637
jtextlayout_getchar, 638
jtextlayout_getcharbox, 638
jtextlayout_getnumchars, 638
jtextlayout_measuretext, 638
jtextlayout_set, 639
jtextlayout_settext, 639
jtextlayout_settextcolor, 639
jtextlayout_withbgcolor, 639
t_jgraphics_textlayout_flags, 637

textfield_get_autoscroll
 TextField, 628

textfield_get_bgcolor
 TextField, 628

textfield_get_editonclick
 TextField, 629

textfield_get_emptytext
 TextField, 629

textfield_get_noactivate
 TextField, 629

textfield_get_owner
 TextField, 629

textfield_get_READONLY
 TextField, 630

textfield_get_selectallonedit
 TextField, 630

textfield_get_textcolor
 TextField, 630

textfield_get_textmargins
 TextField, 630

textfield_get_underline
 TextField, 631

textfield_get_useellipsis
 TextField, 631

textfield_get_wantsreturn
 TextField, 631

textfield_get_wantstab
 TextField, 631

textfield_get_wordwrap
 TextField, 632

textfield_set_autoscroll
 TextField, 632

textfield_set_bgcolor
 TextField, 632

textfield_set_editonclick
 TextField, 632

textfield_set_emptytext
 TextField, 633

textfield_set_noactivate
 TextField, 633

textfield_set_READONLY
 TextField, 633

textfield_set_selectallonedit
 TextField, 633

textfield_set_textcolor
 TextField, 634

textfield_set_textmargins
 TextField, 634

textfield_set_underline
 TextField, 634

textfield_set_useellipsis
 TextField, 634

textfield_set_wantsreturn
 TextField, 635

textfield_set_wantstab
 TextField, 635

textfield_set_wordwrap
 TextField, 635

Threads, 559

defer, 561

defer_low, 561

e_max_systhread_mutex_flags, 561

isr, 563

SYSTHREAD_MUTEX_ERRORCHECK, 561

SYSTHREAD_MUTEX_NORMAL, 561

SYSTHREAD_MUTEX_RECURSIVE, 561

schedule, 563

schedule_delay, 564

systhread_create, 564

systhread_detach, 566

systhread_exit, 566

systhread_getpriority, 566

systhread_isaudiothread, 566

systhread_ismainthread, 567

systhread_istimerthread, 567

systhread_join, 567

systhread_self, 567

systhread_setpriority, 567

systhread_sleep, 568

systhread_terminate, 568

time_calcquantize
 ITM Time Objects, 554

time_getitm

ITM Time Objects, 554
time_getms
 ITM Time Objects, 554
time_getnamed
 ITM Time Objects, 554
time_getphase
 ITM Time Objects, 555
time_gettimecs
 ITM Time Objects, 555
time_isfixedunit
 ITM Time Objects, 555
time_listen
 ITM Time Objects, 555
time_new
 ITM Time Objects, 556
time_now
 ITM Time Objects, 556
time_schedule
 ITM Time Objects, 556
time_schedule_limit
 ITM Time Objects, 556
time_setclock
 ITM Time Objects, 556
time_setvalue
 ITM Time Objects, 558
time_stop
 ITM Time Objects, 558
time_tick
 ITM Time Objects, 558
Timing, 528
typedmess
 Old-Style Classes, 220
Unicode, 651
 charset_convert, 652
 charset_unicodetouf8, 652
 charset_utf8_count, 652
 charset_utf8_offset, 653
 charset_utf8tounicode, 653
User Interface, 575

vptr
 MSP, 437

wind_advise
 Miscellaneous, 405
wind_setcursor
 Miscellaneous, 405
Windows, 430
 jwind_getactive, 430
 jwind_getat, 430
 jwind_getcount, 430
word, 973

z_dsp_free
 MSP, 440
z_dsp_setup
 MSP, 440
zgetfn
 Old-Style Classes, 220