

JavaScriptCore Framework Reference

Contents

[JavaScriptCore Framework Reference](#) 3

[Other References](#) 4

[JSBase.h Reference](#) 5

[Overview](#) 5

[Functions](#) 5

[Data Types](#) 8

[JSContextRef.h Reference](#) 12

[Overview](#) 12

[Functions](#) 12

[JSObjectRef.h Reference](#) 18

[Overview](#) 18

[Functions](#) 18

[Callbacks](#) 40

[Data Types](#) 50

[Constants](#) 53

[JSStringRef.h Reference](#) 55

[Overview](#) 55

[Functions](#) 55

[Data Types](#) 61

[JSStringRefCF.h Reference](#) 62

[Overview](#) 62

[Functions](#) 62

[JSValueRef.h Reference](#) 64

[Overview](#) 64

[Functions](#) 64

[Data Types](#) 80

JavaScriptCore Framework Reference

The JavaScriptCore Framework allows you to evaluate JavaScript programs from within a C-based program. It also lets you insert custom objects to the JavaScript environment.

Other References

JSBase.h Reference

Declared in	JSBase.h
	JSContextRef.h
	JSObjectRef.h
	JSStringRef.h
	JSStringRefCF.h
	JSValueRef.h

Overview

Included Headers

- `<stdbool.h>`
- `<Foundation/Foundation.h>`

Functions

See the Overview section above for header-level documentation.

JSCheckScriptSyntax

Checks for syntax errors in a string of JavaScript.

```
JS_EXPORT bool JSCheckScriptSyntax(  
    JSContextRef ctx,  
    JSStringRef script,  
    JSStringRef sourceURL,  
    int startingLineNumber,  
    JSValueRef *exception);
```

Parameters

`ctx`

The execution context to use.

`script`

A `JSString` containing the script to check for syntax errors.

`sourceURL`

A `JSString` containing a URL for the script's source file. This is only used when reporting exceptions. Pass `NULL` if you do not care to include source file information in exceptions.

`startingLineNumber`

An integer value specifying the script's starting line number in the file located at `sourceURL`. This is only used when reporting exceptions.

`exception`

A pointer to a `JSValueRef` in which to store a syntax error exception, if any. Pass `NULL` if you do not care to store a syntax error exception.

Return Value

true if the script is syntactically correct, otherwise false.

Availability

Available in OS X v10.5 and later.

Declared in

`JSBase.h`

JSEvaluateScript

Evaluates a string of JavaScript.

```
JS_EXPORT JSValueRef JSEvaluateScript(  
    JSContextRef ctx,  
    JSStringRef script,  
    JSObjectRef thisObject,  
    JSStringRef sourceURL,  
    int startingLineNumber,  
    JSValueRef *exception);
```

Parameters

`ctx`

The execution context to use.

`script`

A JSString containing the script to evaluate.

`thisObject`

The object to use as "this," or NULL to use the global object as "this."

`sourceURL`

A JSString containing a URL for the script's source file. This is only used when reporting exceptions. Pass NULL if you do not care to include source file information in exceptions.

`startingLineNumber`

An integer value specifying the script's starting line number in the file located at `sourceURL`. This is only used when reporting exceptions.

`exception`

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

The JSValue that results from evaluating `script`, or NULL if an exception is thrown.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

JSInterpreter

Declared in

JSBase.h

JSGarbageCollect

Performs a JavaScript garbage collection.

```
JS_EXPORT void JSGarbageCollect(  
    JSContextRef ctx);
```

Parameters

`ctx`

The execution context to use.

Discussion

JavaScript values that are on the machine stack, in a register, protected by `JSValueProtect`, set as the global object of an execution context, or reachable from any such value will not be collected.

During JavaScript execution, you are not required to call this function; the JavaScript engine will garbage collect as needed. JavaScript values created within a context group are automatically destroyed when the last reference to the context group is released.

Availability

Available in OS X v10.5 and later.

Declared in

`JSBase.h`

Data Types

See the Overview section above for header-level documentation.

JSClassRef

```
typedef struct OpaqueJSClass* JSClassRef;
```

Discussion

A JavaScript class. Used with `JSObjectMake` to construct objects with custom behavior.

Availability

Available in OS X v10.5 and later.

Declared in

`JSBase.h`

JSContextGroupRef

```
typedef const struct OpaqueJSContextGroup* JSContextGroupRef;
```

Discussion

A group that associates JavaScript contexts with one another. Contexts in the same group may share and exchange JavaScript objects.

Availability

Available in OS X v10.6 and later.

Declared in

JSBase.h

JSContextRef

```
typedef const struct OpaqueJSContext* JSContextRef;
```

Discussion

A JavaScript execution context. Holds the global object and other execution state.

Availability

Available in OS X v10.5 and later.

Declared in

JSBase.h

JSGlobalContextRef

```
typedef struct OpaqueJSContext* JSGlobalContextRef;
```

Discussion

A global JavaScript execution context. A JSGlobalContext is a JSContext.

Availability

Available in OS X v10.5 and later.

Declared in

JSBase.h

JSObjectRef

```
typedef struct OpaqueJSValue* JSObjectRef;
```

Discussion

A JavaScript object. A JSObject is a JSValue.

Availability

Available in OS X v10.5 and later.

Declared in

JSBase.h

JSPropertyNameAccumulatorRef

```
typedef struct OpaqueJSPropertyNameAccumulator* JSPropertyNameAccumulatorRef;
```

Discussion

An ordered set used to collect the names of a JavaScript object's properties.

Availability

Available in OS X v10.5 and later.

Declared in

JSBase.h

JSPropertyNameArrayRef

```
typedef struct OpaqueJSPropertyNameArray* JSPropertyNameArrayRef;
```

Discussion

An array of JavaScript property names.

Availability

Available in OS X v10.5 and later.

Declared in

JSBase.h

JSStringRef

```
typedef struct OpaqueJSString* JSStringRef;
```

Discussion

A UTF16 character buffer. The fundamental string representation in JavaScript.

Availability

Available in OS X v10.5 and later.

Declared in

JSBase.h

JSValueRef

```
typedef const struct OpaqueJSValue* JSValueRef;
```

Discussion

A JavaScript value. The base type for all JavaScript values, and polymorphic functions on them.

Availability

Available in OS X v10.5 and later.

Declared in

JSBase.h

JSContextRef.h Reference

Declared in	JSContextRef.h
-------------	----------------

Overview

Included Headers

- <JavaScriptCore/JSObjectRef.h>
- <JavaScriptCore/JSValueRef.h>
- <JavaScriptCore/WebKitAvailability.h>
- <stdbool.h>

Functions

See the Overview section above for header-level documentation.

JSContextGetGlobalObject

Gets the global object of a JavaScript execution context.

```
JS_EXPORT JSObjectRef JSContextGetGlobalObject(  
    JSContextRef ctx);
```

Parameters

ctx

The JSContext whose global object you want to get.

Return Value

ctx's global object.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

Declared in

JSContextRef.h

JSContextGetGroup

Gets the context group to which a JavaScript execution context belongs.

```
JS_EXPORT JSContextGroupRef JSContextGetGroup(  
    JSContextRef ctx) AVAILABLE_IN_WEBKIT_VERSION_4_0;
```

Parameters

ctx

The JSContext whose group you want to get.

Return Value

ctx's group.

Availability

Available in OS X v10.6 and later.

Declared in

JSContextRef.h

JSContextGroupCreate

Creates a JavaScript context group.

```
JS_EXPORT JSContextGroupRef JSContextGroupCreate() AVAILABLE_IN_WEBKIT_VERSION_4_0;
```

Return Value

The created JSContextGroup.

Discussion

A JSContextGroup associates JavaScript contexts with one another. Contexts in the same group may share and exchange JavaScript objects. Sharing and/or exchanging JavaScript objects between contexts in different groups will produce undefined behavior. When objects from the same context group are used in multiple threads, explicit synchronization is required.

Availability

Available in OS X v10.6 and later.

Declared in

JSContextRef.h

JSContextGroupRelease

Releases a JavaScript context group.

```
JS_EXPORT void JSContextGroupRelease(  
    JSContextGroupRef group) AVAILABLE_IN_WEBKIT_VERSION_4_0;
```

Parameters

group

The JSContextGroup to release.

Availability

Available in OS X v10.6 and later.

Declared in

JSContextRef.h

JSContextGroupRetain

Retains a JavaScript context group.

```
JS_EXPORT JSContextGroupRef JSContextGroupRetain(  
    JSContextGroupRef group) AVAILABLE_IN_WEBKIT_VERSION_4_0;
```

Parameters

group

The JSContextGroup to retain.

Return Value

A JSContextGroup that is the same as group.

Availability

Available in OS X v10.6 and later.

Declared in

JSContextRef.h

JSGlobalContextCreate

Creates a global JavaScript execution context.

```
JS_EXPORT JSGlobalContextRef JSGlobalContextCreate(  
    JSCClassRef globalObjectClass) AVAILABLE_WEBKIT_VERSION_3_0_AND_LATER;
```

Parameters

globalObjectClass

The class to use when creating the global object. Pass NULL to use the default object class.

Return Value

A JSGlobalContext with a global object of class globalObjectClass.

Discussion

JSGlobalContextCreate allocates a global object and populates it with all the built-in JavaScript objects, such as Object, Function, String, and Array.

In WebKit version 4.0 and later, the context is created in a unique context group. Therefore, scripts may execute in it concurrently with scripts executing in other contexts. However, you may not use values created in the context in other contexts.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

JSInterpreter

Declared in

JSContextRef.h

JSGlobalContextCreateInGroup

Creates a global JavaScript execution context in the context group provided.

```
JS_EXPORT JSGlobalContextRef JSGlobalContextCreateInGroup(  
    JSContextGroupRef group,  
    JSCClassRef globalObjectClass) AVAILABLE_IN_WEBKIT_VERSION_4_0;
```

Parameters

`globalObjectClass`

The class to use when creating the global object. Pass NULL to use the default object class.

`group`

The context group to use. The created global context retains the group. Pass NULL to create a unique group for the context.

Return Value

A JSGlobalContext with a global object of class `globalObjectClass` and a context group equal to `group`.

Discussion

`JSGlobalContextCreateInGroup` allocates a global object and populates it with all the built-in JavaScript objects, such as `Object`, `Function`, `String`, and `Array`.

Availability

Available in OS X v10.6 and later.

Declared in

`JSContextRef.h`

JSGlobalContextRelease

Releases a global JavaScript execution context.

```
JS_EXPORT void JSGlobalContextRelease(  
    JSGlobalContextRef ctx);
```

Parameters

`ctx`

The JSGlobalContext to release.

Availability

Available in OS X v10.5 and later.

Related Sample Code
JSInterpreter

Declared in
JSContextRef.h

JSGlobalContextRetain

Retains a global JavaScript execution context.

```
JS_EXPORT JSGlobalContextRef JSGlobalContextRetain(  
    JSGlobalContextRef ctx);
```

Parameters

ctx

The JSGlobalContext to retain.

Return Value

A JSGlobalContext that is the same as ctx.

Availability

Available in OS X v10.5 and later.

Declared in
JSContextRef.h

JSObjectRef.h Reference

Declared in	JSObjectRef.h
-------------	---------------

Overview

Included Headers

- <JavaScriptCore/JSBase.h>
- <JavaScriptCore/JSValueRef.h>
- <JavaScriptCore/WebKitAvailability.h>
- <stdbool.h>
- <stddef.h>

Functions

See the Overview section above for header-level documentation.

JSClassCreate

Creates a JavaScript class suitable for use with JSObjectMake.

```
JS_EXPORT JSClassRef JSClassCreate(  
    const JSClassDefinition *definition);
```

Parameters

definition

A JSClassDefinition that defines the class.

Return Value

A JSClass with the given definition. Ownership follows the Create Rule.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

Declared in

JSObjectRef.h

JSClassRelease

Releases a JavaScript class.

```
JS_EXPORT void JSClassRelease(  
    JSClassRef jsClass);
```

Parameters

jsClass

The JSClass to release.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSClassRetain

Retains a JavaScript class.

```
JS_EXPORT JSClassRef JSClassRetain(  
    JSClassRef jsClass);
```

Parameters

jsClass

The JSClass to retain.

Return Value

A JSClass that is the same as jsClass.

Availability

Available in OS X v10.5 and later.

Declared in
JSObjectRef.h

JSObjectCallAsConstructor

Calls an object as a constructor.

```
JS_EXPORT JSObjectRef JSObjectCallAsConstructor(  
    JSContextRef ctx,  
    JSObjectRef object,  
    size_t argumentCount,  
    const JSValueRef arguments[],  
    JSValueRef *exception);
```

Parameters

ctx

The execution context to use.

object

The JSObject to call as a constructor.

argumentCount

An integer count of the number of arguments in arguments.

arguments

A JSValue array of arguments to pass to the constructor. Pass NULL if argumentCount is 0.

exception

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

The JSObject that results from calling object as a constructor, or NULL if an exception is thrown or object is not a constructor.

Availability

Available in OS X v10.5 and later.

Declared in
JSObjectRef.h

JSObjectCallAsFunction

Calls an object as a function.

```
JS_EXPORT JSValueRef JSObjectCallAsFunction(  
    JSContextRef ctx,  
    JSObjectRef object,  
    JSObjectRef thisObject,  
    size_t argumentCount,  
    const JSValueRef arguments[],  
    JSValueRef *exception);
```

Parameters

`ctx`

The execution context to use.

`object`

The JSObject to call as a function.

`thisObject`

The object to use as "this," or NULL to use the global object as "this."

`argumentCount`

An integer count of the number of arguments in `arguments`.

`arguments`

A JSValue array of arguments to pass to the function. Pass NULL if `argumentCount` is 0.

`exception`

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

The JSValue that results from calling `object` as a function, or NULL if an exception is thrown or `object` is not a function.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

Declared in

JSObjectRef.h

JSObjectCopyPropertyNames

Gets the names of an object's enumerable properties.

```
JS_EXPORT JSPropertyNameArrayRef JSObjectCopyPropertyNames(  
    JSContextRef ctx,  
    JSObjectRef object);
```

Parameters

ctx

The execution context to use.

object

The object whose property names you want to get.

Return Value

A JSPropertyNameArray containing the names object's enumerable properties. Ownership follows the Create Rule.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSObjectDeleteProperty

Deletes a property from an object.

```
JS_EXPORT bool JSObjectDeleteProperty(  
    JSContextRef ctx,  
    JSObjectRef object,  
    JSStringRef propertyName,  
    JSValueRef *exception);
```

Parameters

ctx

The execution context to use.

object

The JSObject whose property you want to delete.

`propertyName`

A JSString containing the property's name.

`exception`

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

true if the delete operation succeeds, otherwise false (for example, if the property has the `kJSPropertyAttributeDontDelete` attribute set).

Availability

Available in OS X v10.5 and later.

Declared in

`JSObjectRef.h`

JSObjectGetPrivate

Gets an object's private data.

```
JS_EXPORT void* JSObjectGetPrivate(  
    JSObjectRef object);
```

Parameters

`object`

A JSObject whose private data you want to get.

Return Value

A void* that is the object's private data, if the object has private data, otherwise NULL.

Availability

Available in OS X v10.5 and later.

Related Sample Code

`JavaScriptCoreHeadstart`

Declared in

`JSObjectRef.h`

JSObjectGetProperty

Gets a property from an object.

```
JS_EXPORT JSValueRef JSObjectGetProperty(  
    JSContextRef ctx,  
    JSObjectRef object,  
    JSStringRef propertyName,  
    JSValueRef *exception);
```

Parameters

ctx

The execution context to use.

object

The JSObject whose property you want to get.

propertyName

A JSString containing the property's name.

exception

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

The property's value if object has the property, otherwise the undefined value.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

Declared in

JSObjectRef.h

JSObjectGetPropertyAtIndex

Gets a property from an object by numeric index.

```
JS_EXPORT JSValueRef JSObjectGetPropertyAtIndex(  
    JSContextRef ctx,  
    JSObjectRef object,  
    unsigned propertyIndex,  
    JSValueRef *exception);
```


Parameters

ctx

The execution context to use.

object

The JSObject whose property you want to get.

propertyIndex

An integer value that is the property's name.

exception

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

The property's value if object has the property, otherwise the undefined value.

Discussion

Calling JSObjectGetPropertyAtIndex is equivalent to calling JSObjectGetProperty with a string containing propertyIndex, but JSObjectGetPropertyAtIndex provides optimized access to numeric properties.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSObjectGetPrototype

Gets an object's prototype.

```
JS_EXPORT JSValueRef JSObjectGetPrototype(  
    JSContextRef ctx,  
    JSObjectRef object);
```

Parameters

ctx

The execution context to use.

object

A JSObject whose prototype you want to get.

Return Value

A JSValue that is the object's prototype.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSObjectHasProperty

Tests whether an object has a given property.

```
JS_EXPORT bool JSObjectHasProperty(  
    JSContextRef ctx,  
    JSObjectRef object,  
    JSStringRef propertyName);
```

Parameters

object

The JSObject to test.

propertyName

A JSString containing the property's name.

Return Value

true if the object has a property whose name matches propertyName, otherwise false.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSObjectIsConstructor

Tests whether an object can be called as a constructor.

```
JS_EXPORT bool JSObjectIsConstructor(  
    JSContextRef ctx,  
    JSObjectRef object);
```

Parameters

ctx

The execution context to use.

object

The JSObject to test.

Return Value

true if the object can be called as a constructor, otherwise false.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSObjectIsFunction

Tests whether an object can be called as a function.

```
JS_EXPORT bool JSObjectIsFunction(  
    JSContextRef ctx,  
    JSObjectRef object);
```

Parameters

ctx

The execution context to use.

object

The JSObject to test.

Return Value

true if the object can be called as a function, otherwise false.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSObjectMake

Creates a JavaScript object.

```
JS_EXPORT JSObjectRef JSObjectMake(  
    JSContextRef ctx,
```

```
JSClassRef jsClass,  
void *data);
```

Parameters

ctx

The execution context to use.

jsClass

The JSClass to assign to the object. Pass NULL to use the default object class.

data

A void* to set as the object's private data. Pass NULL to specify no private data.

Return Value

A JSObject with the given class and private data.

Discussion

The default object class does not allocate storage for private data, so you must provide a non-NULL jsClass to JSObjectMake if you want your object to be able to store private data.

data is set on the created object before the initialize methods in its class chain are called. This enables the initialize methods to retrieve and manipulate data through JSObjectGetPrivate.

Availability

Available in OS X v10.5 and later.

Related Sample Code
JavaScriptCoreHeadstart

Declared in

JSObjectRef.h

JSObjectMakeArray

Creates a JavaScript Array object.

```
JS_EXPORT JSObjectRef JSObjectMakeArray(  
    JSContextRef ctx,  
    size_t argumentCount,  
    const JSValueRef arguments[],  
    JSValueRef *exception) AVAILABLE_IN_WEBKIT_VERSION_4_0;
```

Parameters

`ctx`

The execution context to use.

`argumentCount`

An integer count of the number of arguments in arguments.

`arguments`

A JSValue array of data to populate the Array with. Pass NULL if argumentCount is 0.

`exception`

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

A JSObject that is an Array.

Discussion

The behavior of this function does not exactly match the behavior of the built-in Array constructor. Specifically, if one argument is supplied, this function returns an array with one element.

Availability

Available in OS X v10.6 and later.

Declared in

JSObjectRef.h

JSObjectMakeConstructor

Convenience method for creating a JavaScript constructor.

```
JS_EXPORT JSObjectRef JSObjectMakeConstructor(  
    JSContextRef ctx,  
    JSClassRef jsClass,  
    JSObjectCallAsConstructorCallback callAsConstructor);
```

Parameters

`ctx`

The execution context to use.

`jsClass`

A JSCClass that is the class your constructor will assign to the objects its constructs. jsClass will be used to set the constructor's .prototype property, and to evaluate 'instanceof' expressions. Pass NULL to use the default object class.

callAsConstructor

A JSObjectCallAsConstructorCallback to invoke when your constructor is used in a 'new' expression. Pass NULL to use the default object constructor.

Return Value

A JSObject that is a constructor. The object's prototype will be the default object prototype.

Discussion

The default object constructor takes no arguments and constructs an object of class jsClass with no private data.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSObjectMakeDate

Creates a JavaScript Date object, as if by invoking the built-in Date constructor.

```
JS_EXPORT JSObjectRef JSObjectMakeDate(  
    JSContextRef ctx,  
    size_t argumentCount,  
    const JSValueRef arguments[],  
    JSValueRef *exception) AVAILABLE_IN_WEBKIT_VERSION_4_0;
```

Parameters

ctx

The execution context to use.

argumentCount

An integer count of the number of arguments in arguments.

arguments

A JSValue array of arguments to pass to the Date Constructor. Pass NULL if argumentCount is 0.

exception

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

A JSObject that is a Date.

Availability

Available in OS X v10.6 and later.

Declared in

JSObjectRef.h

JSObjectMakeError

Creates a JavaScript Error object, as if by invoking the built-in Error constructor.

```
JS_EXPORT JSObjectRef JSObjectMakeError(  
    JSContextRef ctx,  
    size_t argumentCount,  
    const JSValueRef arguments[],  
    JSValueRef *exception) AVAILABLE_IN_WEBKIT_VERSION_4_0;
```

Parameters

ctx

The execution context to use.

argumentCount

An integer count of the number of arguments in arguments.

arguments

A JSValue array of arguments to pass to the Error Constructor. Pass NULL if argumentCount is 0.

exception

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

A JSObject that is a Error.

Availability

Available in OS X v10.6 and later.

Declared in

JSObjectRef.h

JSObjectMakeFunction

Creates a function with a given script as its body.

```
JS_EXPORT JSObjectRef JSObjectMakeFunction(  
    JSContextRef ctx,  
    JSStringRef name,  
    unsigned parameterCount,  
    const JSStringRef parameterNames[],  
    JSStringRef body,  
    JSStringRef sourceURL,  
    int startingLineNumber,  
    JSValueRef *exception);
```

Parameters

ctx

The execution context to use.

name

A JSString containing the function's name. This will be used when converting the function to string. Pass NULL to create an anonymous function.

parameterCount

An integer count of the number of parameter names in parameterNames.

parameterNames

A JSString array containing the names of the function's parameters. Pass NULL if parameterCount is 0.

body

A JSString containing the script to use as the function's body.

sourceURL

A JSString containing a URL for the script's source file. This is only used when reporting exceptions. Pass NULL if you do not care to include source file information in exceptions.

startingLineNumber

An integer value specifying the script's starting line number in the file located at sourceURL. This is only used when reporting exceptions.

exception

A pointer to a JSValueRef in which to store a syntax error exception, if any. Pass NULL if you do not care to store a syntax error exception.

Return Value

A JSObject that is a function, or NULL if either body or parameterNames contains a syntax error. The object's prototype will be the default function prototype.

Discussion

Use this method when you want to execute a script repeatedly, to avoid the cost of re-parsing the script before each execution.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSObjectMakeFunctionWithCallback

Convenience method for creating a JavaScript function with a given callback as its implementation.

```
JS_EXPORT JSObjectRef JSObjectMakeFunctionWithCallback(  
    JSContextRef ctx,  
    JSStringRef name,  
    JSObjectCallAsFunctionCallback callAsFunction);
```

Parameters

`ctx`

The execution context to use.

`name`

A JSString containing the function's name. This will be used when converting the function to string. Pass NULL to create an anonymous function.

`callAsFunction`

The JSObjectCallAsFunctionCallback to invoke when the function is called.

Return Value

A JSObject that is a function. The object's prototype will be the default function prototype.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

Declared in

JSObjectRef.h

JSObjectMakeRegExp

Creates a JavaScript RegExp object, as if by invoking the built-in RegExp constructor.

```
JS_EXPORT JSObjectRef JSObjectMakeRegExp(  

```

```
JSContextRef ctx,  
size_t argumentCount,  
const JSValueRef arguments[],  
JSValueRef *exception) AVAILABLE_IN_WEBKIT_VERSION_4_0;
```

Parameters

ctx

The execution context to use.

argumentCount

An integer count of the number of arguments in arguments.

arguments

A JSValue array of arguments to pass to the RegExp Constructor. Pass NULL if argumentCount is 0.

exception

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

A JSObject that is a RegExp.

Availability

Available in OS X v10.6 and later.

Declared in

JSObjectRef.h

JSObjectSetPrivate

Sets a pointer to private data on an object.

```
JS_EXPORT bool JSObjectSetPrivate(  
    JSObjectRef object,  
    void *data);
```

Parameters

object

The JSObject whose private data you want to set.

data

A void* to set as the object's private data.

Return Value

true if object can store private data, otherwise false.

Discussion

The default object class does not allocate storage for private data. Only objects created with a non-NULL JSClass can store private data.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSObjectSetProperty

Sets a property on an object.

```
JS_EXPORT void JSObjectSetProperty(  
    JSContextRef ctx,  
    JSObjectRef object,  
    JSStringRef propertyName,  
    JSValueRef value,  
    JSPROPERTY_ATTRIBUTES attributes,  
    JSValueRef *exception);
```

Parameters

ctx

The execution context to use.

object

The JSObject whose property you want to set.

propertyName

A JSStringRef containing the property's name.

value

A JSValue to use as the property's value.

exception

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

attributes

A logically ORed set of JSPROPERTY_ATTRIBUTES to give to the property.

Availability

Available in OS X v10.5 and later.

Related Sample Code
JavaScriptCoreHeadstart

Declared in

JSObjectRef.h

JSObjectSetPropertyAtIndex

Sets a property on an object by numeric index.

```
JS_EXPORT void JSObjectSetPropertyAtIndex(  
    JSContextRef ctx,  
    JSObjectRef object,  
    unsigned propertyIndex,  
    JSValueRef value,  
    JSValueRef *exception);
```

Parameters

ctx

The execution context to use.

object

The JSObject whose property you want to set.

propertyIndex

The property's name as a number.

value

A JSValue to use as the property's value.

exception

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Discussion

Calling JSObjectSetPropertyAtIndex is equivalent to calling JSObjectSetProperty with a string containing propertyIndex, but JSObjectSetPropertyAtIndex provides optimized access to numeric properties.

Availability

Available in OS X v10.5 and later.

Declared in
JSObjectRef.h

JSObjectSetPrototype

Sets an object's prototype.

```
JS_EXPORT void JSObjectSetPrototype(  
    JSContextRef ctx,  
    JSObjectRef object,  
    JSValueRef value);
```

Parameters

ctx

The execution context to use.

object

The JSObject whose prototype you want to set.

value

A JSValue to set as the object's prototype.

Availability

Available in OS X v10.5 and later.

Declared in
JSObjectRef.h

JSPropertyNameAccumulatorAddName

Adds a property name to a JavaScript property name accumulator.

```
JS_EXPORT void JSPropertyNameAccumulatorAddName(  
    JSPropertyNameAccumulatorRef accumulator,  
    JSStringRef propertyName);
```

Parameters

accumulator

The accumulator object to which to add the property name.

propertyName

The property name to add.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSPropertyNameArrayGetCount

Gets a count of the number of items in a JavaScript property name array.

```
JS_EXPORT size_t JSPropertyNameArrayGetCount(  
    JSPropertyNameArrayRef array);
```

Parameters

array

The array from which to retrieve the count.

Return Value

An integer count of the number of names in array.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSPropertyNameArrayGetNameAtIndex

Gets a property name at a given index in a JavaScript property name array.

```
JS_EXPORT JSStringRef JSPropertyNameArrayGetNameAtIndex(  
    JSPropertyNameArrayRef array,  
    size_t index);
```

Parameters

array

The array from which to retrieve the property name.

index

The index of the property name to retrieve.

Return Value

A JSStringRef containing the property name.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSPropertyNameArrayRelease

Releases a JavaScript property name array.

```
JS_EXPORT void JSPropertyNameArrayRelease(  
    JSPropertyNameArrayRef array);
```

Parameters

array

The JSPropertyNameArray to release.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSPropertyNameArrayRetain

Retains a JavaScript property name array.

```
JS_EXPORT JSPropertyNameArrayRef JSPropertyNameArrayRetain(  
    JSPropertyNameArrayRef array);
```

Parameters

array

The JSPropertyNameArray to retain.

Return Value

A JSPropertyNameArray that is the same as array.

Availability

Available in OS X v10.5 and later.

Declared in
JSObjectRef.h

Callbacks

See the Overview section above for header-level documentation.

JSObjectCallAsConstructorCallback

The callback invoked when an object is used as a constructor in a 'new' expression.

```
typedef JSObjectRef ( *JSObjectCallAsConstructorCallback) (  
    JSContextRef ctx,  
    JSObjectRef constructor,  
    size_t argumentCount,  
    const JSValueRef arguments[],  
    JSValueRef *exception);
```

Parameters

`ctx`

The execution context to use.

`constructor`

A JSObject that is the constructor being called.

`argumentCount`

An integer count of the number of arguments in `arguments`.

`arguments`

A JSValue array of the arguments passed to the function.

`exception`

A pointer to a JSValueRef in which to return an exception, if any.

Return Value

A JSObject that is the constructor's return value.

Discussion

If you named your function `CallAsConstructor`, you would declare it like this:

```
JSObjectRef CallAsConstructor(JSContextRef ctx, JSObjectRef constructor, size_t argumentCount, const JSValueRef  
arguments[], JSValueRef* exception);
```


If your callback were invoked by the JavaScript expression 'new myConstructor()', constructor would be set to myConstructor.

If this callback is NULL, using your object as a constructor in a 'new' expression will throw an exception.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSObjectCallAsFunctionCallback

The callback invoked when an object is called as a function.

```
typedef JSValueRef ( *JSObjectCallAsFunctionCallback) (  
    JSContextRef ctx,  
    JSObjectRef function,  
    JSObjectRef thisObject,  
    size_t argumentCount,  
    const JSValueRef arguments[],  
    JSValueRef *exception);
```

Parameters

ctx

The execution context to use.

function

A JSObject that is the function being called.

thisObject

A JSObject that is the 'this' variable in the function's scope.

argumentCount

An integer count of the number of arguments in arguments.

arguments

A JSValue array of the arguments passed to the function.

exception

A pointer to a JSValueRef in which to return an exception, if any.

Return Value

A JSValue that is the function's return value.

Discussion

If you named your function `CallAsFunction`, you would declare it like this:

```
JSValueRef CallAsFunction(JSContextRef ctx, JSObjectRef function, JSObjectRef thisObject, size_t argumentCount,
const JSValueRef arguments[], JSValueRef* exception);
```

If your callback were invoked by the JavaScript expression `'myObject.myFunction()'`, `function` would be set to `myFunction`, and `thisObject` would be set to `myObject`.

If this callback is `NULL`, calling your object as a function will throw an exception.

Availability

Available in OS X v10.5 and later.

Declared in

`JSObjectRef.h`

JSObjectConvertToTypeCallback

The callback invoked when converting an object to a particular JavaScript type.

```
typedef JSValueRef ( *JSObjectConvertToTypeCallback) (
    JSContextRef ctx,
    JSObjectRef object,
    JSType type,
    JSValueRef *exception);
```

Parameters

`ctx`

The execution context to use.

`object`

The JSObject to convert.

`type`

A `JSType` specifying the JavaScript type to convert to.

`exception`

A pointer to a `JSValueRef` in which to return an exception, if any.

Return Value

The object's converted value, or `NULL` if the object was not converted.

Discussion

If you named your function `ConvertToType`, you would declare it like this:

```
JSValueRef ConvertToType(JSContextRef ctx, JSObjectRef object, JSType type, JSValueRef* exception);
```

If this function returns false, the conversion request forwards to object's parent class chain (which includes the default object class).

This function is only invoked when converting an object to number or string. An object converted to boolean is 'true.' An object converted to object is itself.

Availability

Available in OS X v10.5 and later.

Declared in

`JSObjectRef.h`

JSObjectDeletePropertyCallback

The callback invoked when deleting a property.

```
typedef bool ( *JSObjectDeletePropertyCallback) (  
    JSContextRef ctx,  
    JSObjectRef object,  
    JSStringRef propertyName,  
    JSValueRef *exception);
```

Parameters

`ctx`

The execution context to use.

`object`

The JSObject in which to delete the property.

`propertyName`

A JSString containing the name of the property to delete.

`exception`

A pointer to a JSValueRef in which to return an exception, if any.

Return Value

true if `propertyName` was successfully deleted, otherwise false.

Discussion

If you named your function `DeleteProperty`, you would declare it like this:

`bool DeleteProperty(JSContextRef ctx, JSObjectRef object, JSStringRef propertyName, JSValueRef* exception);`

If this function returns false, the delete request forwards to object's statically declared properties, then its parent class chain (which includes the default object class).

Availability

Available in OS X v10.5 and later.

Declared in

`JSObjectRef.h`

JSObjectFinalizeCallback

The callback invoked when an object is finalized (prepared for garbage collection). An object may be finalized on any thread.

```
typedef void ( *JSObjectFinalizeCallback) (  
    JSObjectRef object);
```

Parameters

`object`

The JSObject being finalized.

Discussion

If you named your function `Finalize`, you would declare it like this:

```
void Finalize(JSObjectRef object);
```

The finalize callback is called on the most derived class first, and the least derived class (the parent class) last.

You must not call any function that may cause a garbage collection or an allocation of a garbage collected object from within a `JSObjectFinalizeCallback`. This includes all functions that have a `JSContextRef` parameter.

Availability

Available in OS X v10.5 and later.

Declared in

`JSObjectRef.h`

JSObjectGetPropertyCallback

The callback invoked when getting a property's value.

```
typedef JSValueRef ( *JSObjectGetPropertyCallback) (  
    JSContextRef ctx,  
    JSObjectRef object,  
    JSStringRef propertyName,  
    JSValueRef *exception);
```

Parameters

ctx

The execution context to use.

object

The JSObject to search for the property.

propertyName

A JSString containing the name of the property to get.

exception

A pointer to a JSValueRef in which to return an exception, if any.

Return Value

The property's value if object has the property, otherwise NULL.

Discussion

If you named your function `GetProperty`, you would declare it like this:

```
JSValueRef GetProperty(JSContextRef ctx, JSObjectRef object, JSStringRef propertyName, JSValueRef* exception);
```

If this function returns NULL, the get request forwards to object's statically declared properties, then its parent class chain (which includes the default object class), then its prototype chain.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSObjectGetPropertyNamesCallback

The callback invoked when collecting the names of an object's properties.

```
typedef void ( *JSObjectGetPropertyNamesCallback) (  
    JSContextRef ctx,
```

```
JSObjectRef object,  
JSPropertyNameAccumulatorRef propertyNames);
```

Parameters

ctx

The execution context to use.

object

The JSObject whose property names are being collected.

accumulator

A JavaScript property name accumulator in which to accumulate the names of object's properties.

Discussion

If you named your function `GetPropertyNames`, you would declare it like this:

```
void GetPropertyNames(JSContextRef ctx, JSObjectRef object, JSPropertyNameAccumulatorRef propertyNames);
```

Property name accumulators are used by `JSObjectCopyPropertyNames` and JavaScript `for...in` loops.

Use `JSPropertyNameAccumulatorAddName` to add property names to accumulator. A class's `getPropertyNames` callback only needs to provide the names of properties that the class vends through a custom `getProperty` or `setProperty` callback. Other properties, including statically declared properties, properties vended by other classes, and properties belonging to object's prototype, are added independently.

Availability

Available in OS X v10.5 and later.

Declared in

`JSObjectRef.h`

JSObjectHasInstanceCallback

hasInstance The callback invoked when an object is used as the target of an 'instanceof' expression.

```
typedef bool ( *JSObjectHasInstanceCallback) (  
    JSContextRef ctx,  
    JSObjectRef constructor,  
    JSValueRef possibleInstance,  
    JSValueRef *exception);
```

Parameters

ctx

The execution context to use.

constructor

The JSObject that is the target of the 'instanceof' expression.

possibleInstance

The JSValue being tested to determine if it is an instance of constructor.

exception

A pointer to a JSValueRef in which to return an exception, if any.

Return Value

true if possibleInstance is an instance of constructor, otherwise false.

Discussion

If you named your function HasInstance, you would declare it like this:

```
bool HasInstance(JSContextRef ctx, JSObjectRef constructor, JSValueRef possibleInstance, JSValueRef* exception);
```

If your callback were invoked by the JavaScript expression 'someValue instanceof myObject', constructor would be set to myObject and possibleInstance would be set to someValue.

If this callback is NULL, 'instanceof' expressions that target your object will return false.

Standard JavaScript practice calls for objects that implement the callAsConstructor callback to implement the hasInstance callback as well.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSObjectHasPropertyCallback

The callback invoked when determining whether an object has a property.

```
typedef bool ( *JSObjectHasPropertyCallback) (  
    JSContextRef ctx,  
    JSObjectRef object,  
    JSStringRef propertyName);
```

Parameters

ctx

The execution context to use.

object

The JSObject to search for the property.

propertyName

A JSString containing the name of the property look up.

Return Value

true if object has the property, otherwise false.

Discussion

If you named your function HasProperty, you would declare it like this:

```
bool HasProperty(JSContextRef ctx, JSObjectRef object, JSStringRef propertyName);
```

If this function returns false, the hasProperty request forwards to object's statically declared properties, then its parent class chain (which includes the default object class), then its prototype chain.

This callback enables optimization in cases where only a property's existence needs to be known, not its value, and computing its value would be expensive.

If this callback is NULL, the getProperty callback will be used to service hasProperty requests.

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSObjectInitializeCallback

The callback invoked when an object is first created.

```
typedef void ( *JSObjectInitializeCallback) (  
    JSContextRef ctx,  
    JSObjectRef object);
```

Parameters

ctx

The execution context to use.

object

The JSObject being created.

Discussion

If you named your function `Initialize`, you would declare it like this:

```
void Initialize(JSContextRef ctx, JSObjectRef object);
```

Unlike the other object callbacks, the `initialize` callback is called on the least derived class (the parent class) first, and the most derived class last.

Availability

Available in OS X v10.5 and later.

Declared in

`JSObjectRef.h`

JSObjectSetPropertyCallback

The callback invoked when setting a property's value.

```
typedef bool ( *JSObjectSetPropertyCallback) (  
    JSContextRef ctx,  
    JSObjectRef object,  
    JSStringRef propertyName,  
    JSValueRef value,  
    JSValueRef *exception);
```

Parameters

`ctx`

The execution context to use.

`object`

The JSObject on which to set the property's value.

`propertyName`

A JSStringRef containing the name of the property to set.

`value`

A JSValue to use as the property's value.

`exception`

A pointer to a JSValueRef in which to return an exception, if any.

Return Value

true if the property was set, otherwise false.

Discussion

If you named your function SetProperty, you would declare it like this:

```
bool SetProperty(JSContextRef ctx, JSObjectRef object, JSStringRef propertyName, JSValueRef value, JSValueRef* exception);
```

If this function returns false, the set request forwards to object's statically declared properties, then its parent class chain (which includes the default object class).

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

Data Types

See the Overview section above for header-level documentation.

JSClassAttributes

A set of JSClassAttributes. Combine multiple attributes by logically ORing them together.

```
typedef unsigned JSClassAttributes;
```

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSClassDefinition

This structure contains properties and callbacks that define a type of object. All fields other than the version field are optional. Any pointer may be NULL.

```
typedef struct {  
    int version; /* current (and only) version is 0 */  
    JSClassAttributes attributes;  
    const char *className;  
    JSClassRef parentClass;
```

```
const JSStaticValue *staticValues;
const JSStaticFunction *staticFunctions;
JSObjectInitializeCallback initialize;
JSObjectFinalizeCallback finalize;
JSObjectHasPropertyCallback hasProperty;
JSObjectGetPropertyCallback getProperty;
JSObjectSetPropertyCallback setProperty;
JSObjectDeletePropertyCallback deleteProperty;
JSObjectGetPropertyNamesCallback getPropertyNames;
JSObjectCallAsFunctionCallback callAsFunction;
JSObjectCallAsConstructorCallback callAsConstructor;
JSObjectHasInstanceCallback hasInstance;
JSObjectConvertToTypeCallback convertToType;
} JSClassDefinition;
```

Discussion

The `staticValues` and `staticFunctions` arrays are the simplest and most efficient means for vending custom properties. Statically declared properties automatically service requests like `getProperty`, `setProperty`, and `getPropertyNames`. Property access callbacks are required only to implement unusual properties, like array indexes, whose names are not known at compile-time.

If you named your getter function "GetX" and your setter function "SetX", you would declare a `JSStaticValue` array containing "X" like this:

```
JSStaticValue StaticValueArray[] = { { "X", GetX, SetX, kJSPropertyAttributeNone }, { 0, 0, 0, 0 } };
```

Standard JavaScript practice calls for storing function objects in prototypes, so they can be shared. The default `JSClass` created by `JSClassCreate` follows this idiom, instantiating objects with a shared, automatically generating prototype containing the class's function objects. The `kJSClassAttributeNoAutomaticPrototype` attribute specifies that a `JSClass` should not automatically generate such a prototype. The resulting `JSClass` instantiates objects with the default object prototype, and gives each instance object its own copy of the class's function objects.

A `NULL` callback specifies that the default object callback should substitute, except in the case of `hasProperty`, where it specifies that `getProperty` should substitute.

Availability

Available in OS X v10.5 and later.

Declared in

`JSObjectRef.h`

JSPropertyAttributes

A set of `JSPropertyAttributes`. Combine multiple attributes by logically ORing them together.

```
typedef unsigned JSPropertyAttributes;
```

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSStaticFunction

This structure describes a statically declared function property.

```
typedef struct {  
    const char *name;  
    JSObjectCallAsFunctionCallback callAsFunction;  
    JSPropertyAttributes attributes;  
} JSStaticFunction;
```

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

JSStaticValue

This structure describes a statically declared value property.

```
typedef struct {  
    const char *name;  
    JSObjectGetPropertyCallback getProperty;  
    JSObjectSetPropertyCallback setProperty;  
    JSPropertyAttributes attributes;  
} JSStaticValue;
```

Availability

Available in OS X v10.5 and later.

Declared in

JSObjectRef.h

Constants

See the Overview section above for header-level documentation.

Global Constants

```
JS_EXPORT extern const JSClassDefinition kJSClassDefinitionEmpty;
```

Constants

`kJSClassDefinitionEmpty`

A `JSClassDefinition` structure of the current version, filled with NULL pointers and having no attributes. Use this constant as a convenience when creating class definitions. For example, to create a class definition with only a finalize method:

```
JSClassDefinition definition = kJSClassDefinitionEmpty; definition.finalize = Finalize;
```

Available in OS X v10.5 and later.

Declared in `JSObjectRef.h`.

JSClassAttribute

```
enum {  
    kJSClassAttributeNone = 0,  
    kJSClassAttributeNoAutomaticPrototype = 1 << 1  
};
```

Constants

`kJSClassAttributeNone`

Specifies that a class has no special attributes.

Available in OS X v10.5 and later.

Declared in `JSObjectRef.h`.

`kJSClassAttributeNoAutomaticPrototype`

Specifies that a class should not automatically generate a shared prototype for its instance objects. Use `kJSClassAttributeNoAutomaticPrototype` in combination with `JSObjectSetPrototype` to manage prototypes manually.

Available in OS X v10.5 and later.

Declared in `JSObjectRef.h`.

JSPropertyAttribute

```
enum {  
    kJSPropertyAttributeNone = 0,  
    kJSPropertyAttributeReadOnly = 1 << 1,  
    kJSPropertyAttributeDontEnum = 1 << 2,  
    kJSPropertyAttributeDontDelete = 1 << 3  
};
```

Constants

kJSPropertyAttributeNone

Specifies that a property has no special attributes.

Available in OS X v10.5 and later.

Declared in JSObjectRef.h.

kJSPropertyAttributeReadOnly

Specifies that a property is read-only.

Available in OS X v10.5 and later.

Declared in JSObjectRef.h.

kJSPropertyAttributeDontEnum

Specifies that a property should not be enumerated by JSPropertyEnumerators and JavaScript for...in loops.

Available in OS X v10.5 and later.

Declared in JSObjectRef.h.

kJSPropertyAttributeDontDelete

Specifies that the delete operation should fail on a property.

Available in OS X v10.5 and later.

Declared in JSObjectRef.h.

JSStringRef.h Reference

Declared in	JSStringRef.h
-------------	---------------

Overview

Included Headers

- `<JavaScriptCore/JSValueRef.h>`
- `<stdbool.h>`
- `<stddef.h>`

Functions

See the Overview section above for header-level documentation.

JSStringCreateWithCharacters

Creates a JavaScript string from a buffer of Unicode characters.

```
JS_EXPORT JSStringRef JSStringCreateWithCharacters(  
    const JSChar *chars,  
    size_t numChars);
```

Parameters

`chars`

The buffer of Unicode characters to copy into the new JSString.

`numChars`

The number of characters to copy from the buffer pointed to by `chars`.

Return Value

A JSString containing `chars`. Ownership follows the Create Rule.

Availability

Available in OS X v10.5 and later.

Declared in

JSStringRef.h

JSStringCreateWithUTF8CString

Creates a JavaScript string from a null-terminated UTF8 string.

```
JS_EXPORT JSStringRef JSStringCreateWithUTF8CString(  
    const char *string);
```

Parameters

string

The null-terminated UTF8 string to copy into the new JSString.

Return Value

A JSString containing string. Ownership follows the Create Rule.

Availability

Available in OS X v10.5 and later.

Declared in

JSStringRef.h

JSStringGetCharactersPtr

Returns a pointer to the Unicode character buffer that serves as the backing store for a JavaScript string.

```
JS_EXPORT const JSChar* JSStringGetCharactersPtr(  
    JSStringRef string);
```

Parameters

string

The JSString whose backing store you want to access.

Return Value

A pointer to the Unicode character buffer that serves as string's backing store, which will be deallocated when string is deallocated.

Availability

Available in OS X v10.5 and later.

Declared in

JSStringRef.h

JSStringGetLength

Returns the number of Unicode characters in a JavaScript string.

```
JS_EXPORT size_t JSStringGetLength(  
    JSStringRef string);
```

Parameters

string

The JSString whose length (in Unicode characters) you want to know.

Return Value

The number of Unicode characters stored in string.

Availability

Available in OS X v10.5 and later.

Declared in

JSStringRef.h

JSStringGetMaximumUTF8CStringSize

Returns the maximum number of bytes a JavaScript string will take up if converted into a null-terminated UTF8 string.

```
JS_EXPORT size_t JSStringGetMaximumUTF8CStringSize(  
    JSStringRef string);
```

Parameters

string

The JSString whose maximum converted size (in bytes) you want to know.

Return Value

The maximum number of bytes that could be required to convert string into a null-terminated UTF8 string. The number of bytes that the conversion actually ends up requiring could be less than this, but never more.

Availability

Available in OS X v10.5 and later.

Declared in

JSStringRef.h

JSStringGetUTF8CString

Converts a JavaScript string into a null-terminated UTF8 string, and copies the result into an external byte buffer.

```
JS_EXPORT size_t JSStringGetUTF8CString(  
    JSStringRef string,  
    char *buffer,  
    size_t bufferSize);
```

Parameters

string

The source JSString.

buffer

The destination byte buffer into which to copy a null-terminated UTF8 representation of string. On return, buffer contains a UTF8 string representation of string. If bufferSize is too small, buffer will contain only partial results. If buffer is not at least bufferSize bytes in size, behavior is undefined.

bufferSize

The size of the external buffer in bytes.

Return Value

The number of bytes written into buffer (including the null-terminator byte).

Availability

Available in OS X v10.5 and later.

Declared in

JSStringRef.h

JSStringIsEqual

Tests whether two JavaScript strings match.

```
JS_EXPORT bool JSStringIsEqual(  
    JSStringRef string1,  
    JSStringRef string2);
```

```
JSStringRef a,  
JSStringRef b);
```

Parameters

a

The first JSString to test.

b

The second JSString to test.

Return Value

true if the two strings match, otherwise false.

Availability

Available in OS X v10.5 and later.

Declared in

JSStringRef.h

JSStringIsEqualToUTF8CString

Tests whether a JavaScript string matches a null-terminated UTF8 string.

```
JS_EXPORT bool JSStringIsEqualToUTF8CString(  
    JSStringRef a,  
    const char *b);
```

Parameters

a

The JSString to test.

b

The null-terminated UTF8 string to test.

Return Value

true if the two strings match, otherwise false.

Availability

Available in OS X v10.5 and later.

Declared in

JSStringRef.h

JSStringRelease

Releases a JavaScript string.

```
JS_EXPORT void JSStringRelease(  
    JSStringRef string);
```

Parameters

string

The JSString to release.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

JSInterpreter

Declared in

JSStringRef.h

JSStringRetain

Retains a JavaScript string.

```
JS_EXPORT JSStringRef JSStringRetain(  
    JSStringRef string);
```

Parameters

string

The JSString to retain.

Return Value

A JSString that is the same as string.

Availability

Available in OS X v10.5 and later.

Declared in

JSStringRef.h

Data Types

See the Overview section above for header-level documentation.

JSChar

A Unicode character.

```
typedef unsigned short JSChar;
```

Availability

Available in OS X v10.5 and later.

Declared in

JSStringRef.h

JSStringRefCF.h Reference

Declared in	JSStringRefCF.h
-------------	-----------------

Overview

Included Headers

- "JSBase.h"
- <CoreFoundation/CoreFoundation.h>

Functions

See the Overview section above for header-level documentation.

JSStringCopyCFString

Creates a CFString from a JavaScript string.

```
JS_EXPORT CFStringRef JSStringCopyCFString(  
    CFAllocatorRef alloc,  
    JSStringRef string);
```

Parameters

`alloc`

The alloc parameter to pass to CFStringCreate.

`string`

The JSString to copy into the new CFString.

Return Value

A CFString containing string. Ownership follows the Create Rule.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JSInterpreter

Declared in

JSStringRefCF.h

JSStringCreateWithCFString

Creates a JavaScript string from a CFString.

```
JS_EXPORT JSStringRef JSStringCreateWithCFString(  
    CFStringRef string);
```

Parameters

string

The CFString to copy into the new JSString.

Return Value

A JSString containing string. Ownership follows the Create Rule.

Discussion

This function is optimized to take advantage of cases when CFStringGetCharactersPtr returns a valid pointer.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

JSInterpreter

Declared in

JSStringRefCF.h

JSValueRef.h Reference

Declared in	JSValueRef.h
-------------	--------------

Overview

Included Headers

- <JavaScriptCore/JSBase.h>
- <JavaScriptCore/WebKitAvailability.h>
- <stdbool.h>

Functions

See the Overview section above for header-level documentation.

JSValueCreateJSONString

Creates a JavaScript string containing the JSON serialized representation of a JS value.

```
JS_EXPORT JSStringRef JSValueCreateJSONString(
    JSContextRef ctx,
    JSValueRef value,
    unsigned indent,
    JSValueRef *exception) AVAILABLE_AFTER_WEBKIT_VERSION_4_0;
```

Parameters

ctx

The execution context to use.

value

The value to serialize.

`indent`

The number of spaces to indent when nesting. If 0, the resulting JSON will not contain newlines. The size of the indent is clamped to 10 spaces.

`exception`

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

A JSString with the result of serialization, or NULL if an exception is thrown.

Availability

Available in OS X v10.7 and later.

Declared in

JSValueRef.h

JSValueGetType

Returns a JavaScript value's type.

```
JS_EXPORT JSType JSValueGetType(  
    JSContextRef ctx,  
    JSValueRef);
```

Parameters

`ctx`

The execution context to use.

`value`

The JSValue whose type you want to obtain.

Return Value

A value of type JSType that identifies value's type.

Availability

Available in OS X v10.5 and later.

Declared in

JSValueRef.h

JSValueIsBoolean

Tests whether a JavaScript value's type is the boolean type.

```
JS_EXPORT bool JSValueIsBoolean(  
    JSContextRef ctx,  
    JSValueRef value);
```

Parameters

ctx

The execution context to use.

value

The JSValue to test.

Return Value

true if value's type is the boolean type, otherwise false.

Availability

Available in OS X v10.5 and later.

Related Sample Code
JavaScriptCoreHeadstart

Declared in

JSValueRef.h

JSValueIsEqual

Tests whether two JavaScript values are equal, as compared by the JS == operator.

```
JS_EXPORT bool JSValueIsEqual(  
    JSContextRef ctx,  
    JSValueRef a,  
    JSValueRef b,  
    JSValueRef *exception);
```

Parameters

ctx

The execution context to use.

a

The first value to test.

b

The second value to test.

exception

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

true if the two values are equal, false if they are not equal or an exception is thrown.

Availability

Available in OS X v10.5 and later.

Declared in

JSValueRef.h

JSValueIsInstanceOfConstructor

Tests whether a JavaScript value is an object constructed by a given constructor, as compared by the JS instanceof operator.

```
JS_EXPORT bool JSValueIsInstanceOfConstructor(  
    JSContextRef ctx,  
    JSValueRef value,  
    JSObjectRef constructor,  
    JSValueRef *exception);
```

Parameters

ctx

The execution context to use.

value

The JSValue to test.

constructor

The constructor to test against.

exception

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

true if value is an object constructed by constructor, as compared by the JS instanceof operator, otherwise false.

Availability

Available in OS X v10.5 and later.

Declared in

JSValueRef.h

JSValueIsNull

Tests whether a JavaScript value's type is the null type.

```
JS_EXPORT bool JSValueIsNull(  
    JSContextRef ctx,  
    JSValueRef value);
```

Parameters

ctx

The execution context to use.

value

The JSValue to test.

Return Value

true if value's type is the null type, otherwise false.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

Declared in

JSValueRef.h

JSValueIsNumber

Tests whether a JavaScript value's type is the number type.

```
JS_EXPORT bool JSValueIsNumber(  
    JSContextRef ctx,  
    JSValueRef value);
```

Parameters

`ctx`

The execution context to use.

`value`

The JSValue to test.

Return Value

true if value's type is the number type, otherwise false.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

Declared in

JSValueRef.h

JSValueIsObject

Tests whether a JavaScript value's type is the object type.

```
JS_EXPORT bool JSValueIsObject(  
    JSContextRef ctx,  
    JSValueRef value);
```

Parameters

`ctx`

The execution context to use.

`value`

The JSValue to test.

Return Value

true if value's type is the object type, otherwise false.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

Declared in

JSValueRef.h

JSValueIsObjectOfClass

Tests whether a JavaScript value is an object with a given class in its class chain.

```
JS_EXPORT bool JSValueIsObjectOfClass(  
    JSContextRef ctx,  
    JSValueRef value,  
    JSClassRef jsClass);
```

Parameters

ctx

The execution context to use.

value

The JSValue to test.

jsClass

The JSClass to test against.

Return Value

true if value is an object and has jsClass in its class chain, otherwise false.

Availability

Available in OS X v10.5 and later.

Declared in

JSValueRef.h

JSValueIsStrictEqual

Tests whether two JavaScript values are strict equal, as compared by the JS === operator.

```
JS_EXPORT bool JSValueIsStrictEqual(  
    JSContextRef ctx,  
    JSValueRef a,  
    JSValueRef b);
```

Parameters

ctx

The execution context to use.

a

The first value to test.

b

The second value to test.

Return Value

true if the two values are strict equal, otherwise false.

Availability

Available in OS X v10.5 and later.

Declared in

JSValueRef.h

JSValueIsString

Tests whether a JavaScript value's type is the string type.

```
JS_EXPORT bool JSValueIsString(  
    JSContextRef ctx,  
    JSValueRef value);
```

Parameters

ctx

The execution context to use.

value

The JSValue to test.

Return Value

true if value's type is the string type, otherwise false.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

Declared in

JSValueRef.h

JSValueIsUndefined

Tests whether a JavaScript value's type is the undefined type.

```
JS_EXPORT bool JSValueIsUndefined(  
    JSContextRef ctx,  
    JSValueRef value);
```

Parameters

ctx

The execution context to use.

value

The JSValue to test.

Return Value

true if value's type is the undefined type, otherwise false.

Availability

Available in OS X v10.5 and later.

Declared in

JSValueRef.h

JSValueMakeBoolean

Creates a JavaScript value of the boolean type.

```
JS_EXPORT JSValueRef JSValueMakeBoolean(  
    JSContextRef ctx,  
    bool boolean);
```

Parameters

ctx

The execution context to use.

boolean

The bool to assign to the newly created JSValue.

Return Value

A JSValue of the boolean type, representing the value of boolean.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

Declared in
JSValueRef.h

JSValueMakeFromJSONString

Creates a JavaScript value from a JSON formatted string.

```
JS_EXPORT JSValueRef JSValueMakeFromJSONString(  
    JSContextRef ctx,  
    JSStringRef string) AVAILABLE_AFTER_WEBKIT_VERSION_4_0;
```

Parameters

ctx

The execution context to use.

string

The JSString containing the JSON string to be parsed.

Return Value

A JSValue containing the parsed value, or NULL if the input is invalid.

Availability

Available in OS X v10.7 and later.

Declared in
JSValueRef.h

JSValueMakeNull

Creates a JavaScript value of the null type.

```
JS_EXPORT JSValueRef JSValueMakeNull(  
    JSContextRef ctx);
```

Parameters

ctx

The execution context to use.

Return Value

The unique null value.

Availability

Available in OS X v10.5 and later.

Related Sample Code
JavaScriptCoreHeadstart

Declared in

JSValueRef.h

JSValueMakeNumber

Creates a JavaScript value of the number type.

```
JS_EXPORT JSValueRef JSValueMakeNumber(  
    JSContextRef ctx,  
    double number);
```

Parameters

ctx

The execution context to use.

number

The double to assign to the newly created JSValue.

Return Value

A JSValue of the number type, representing the value of number.

Availability

Available in OS X v10.5 and later.

Related Sample Code
JavaScriptCoreHeadstart

Declared in

JSValueRef.h

JSValueMakeString

Creates a JavaScript value of the string type.

```
JS_EXPORT JSValueRef JSValueMakeString(  

```

```
JSContextRef ctx,  
JSStringRef string);
```

Parameters

`ctx`

The execution context to use.

`string`

The JSString to assign to the newly created JSValue. The newly created JSValue retains string, and releases it upon garbage collection.

Return Value

A JSValue of the string type, representing the value of string.

Availability

Available in OS X v10.5 and later.

Related Sample Code
JavaScriptCoreHeadstart

Declared in

JSValueRef.h

JSValueMakeUndefined

Creates a JavaScript value of the undefined type.

```
JS_EXPORT JSValueRef JSValueMakeUndefined(  
    JSContextRef ctx);
```

Parameters

`ctx`

The execution context to use.

Return Value

The unique undefined value.

Availability

Available in OS X v10.5 and later.

Related Sample Code
JavaScriptCoreHeadstart

Declared in

JSValueRef.h

JSValueProtect

Protects a JavaScript value from garbage collection.

```
JS_EXPORT void JSValueProtect(  
    JSContextRef ctx,  
    JSValueRef value);
```

Parameters

ctx

The execution context to use.

value

The JSValue to protect.

Discussion

Use this method when you want to store a JSValue in a global or on the heap, where the garbage collector will not be able to discover your reference to it.

A value may be protected multiple times and must be unprotected an equal number of times before becoming eligible for garbage collection.

Availability

Available in OS X v10.5 and later.

Related Sample Code
JavaScriptCoreHeadstart

Declared in

JSValueRef.h

JSValueToBoolean

Converts a JavaScript value to boolean and returns the resulting boolean.

```
JS_EXPORT bool JSValueToBoolean(  
    JSContextRef ctx,  
    JSValueRef value);
```

Parameters

ctx

The execution context to use.

value

The JSValue to convert.

Return Value

The boolean result of conversion.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

Declared in

JSValueRef.h

JSValueToNumber

Converts a JavaScript value to number and returns the resulting number.

```
JS_EXPORT double JSValueToNumber(  
    JSContextRef ctx,  
    JSValueRef value,  
    JSValueRef *exception);
```

Parameters

ctx

The execution context to use.

value

The JSValue to convert.

exception

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

The numeric result of conversion, or NaN if an exception is thrown.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

Declared in
JSValueRef.h

JSValueToObject

Converts a JavaScript value to object and returns the resulting object.

```
JS_EXPORT JSObjectRef JSValueToObject(  
    JSContextRef ctx,  
    JSValueRef value,  
    JSValueRef *exception);
```

Parameters

ctx

The execution context to use.

value

The JSValue to convert.

exception

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

The JSObject result of conversion, or NULL if an exception is thrown.

Availability

Available in OS X v10.5 and later.

Related Sample Code
JavaScriptCoreHeadstart

Declared in
JSValueRef.h

JSValueToStringCopy

Converts a JavaScript value to string and copies the result into a JavaScript string.

```
JS_EXPORT JSStringRef JSValueToStringCopy(  
    JSContextRef ctx,  
    JSValueRef value,  
    JSValueRef *exception);
```

Parameters

`ctx`

The execution context to use.

`value`

The JSValue to convert.

`exception`

A pointer to a JSValueRef in which to store an exception, if any. Pass NULL if you do not care to store an exception.

Return Value

A JSString with the result of conversion, or NULL if an exception is thrown. Ownership follows the Create Rule.

Availability

Available in OS X v10.5 and later.

Related Sample Code

JavaScriptCoreHeadstart

JSInterpreter

Declared in

JSValueRef.h

JSValueUnprotect

Unprotects a JavaScript value from garbage collection.

```
JS_EXPORT void JSValueUnprotect(  
    JSContextRef ctx,  
    JSValueRef value);
```

Parameters

`ctx`

The execution context to use.

`value`

The JSValue to unprotect.

Discussion

A value may be protected multiple times and must be unprotected an equal number of times before becoming eligible for garbage collection.

Availability

Available in OS X v10.5 and later.

Declared in

JSValueRef.h

Data Types

See the Overview section above for header-level documentation.

JSType

A constant identifying the type of a JSValue.

```
typedef enum {  
    kJSTypeUndefined,  
    kJSTypeNull,  
    kJSTypeBoolean,  
    kJSTypeNumber,  
    kJSTypeString,  
    kJSTypeObject  
} JSType;
```

Constants

`kJSTypeUndefined`

The unique undefined value.

Available in OS X v10.5 and later.

Declared in JSValueRef.h.

`kJSTypeNull`

The unique null value.

Available in OS X v10.5 and later.

Declared in JSValueRef.h.

`kJSTypeBoolean`

A primitive boolean value, one of true or false.

Available in OS X v10.5 and later.

Declared in JSValueRef.h.

`kJSTypeNumber`

A primitive number value.

Available in OS X v10.5 and later.

Declared in `JSValueRef.h`.

`kJSTypeString`

A primitive string value.

Available in OS X v10.5 and later.

Declared in `JSValueRef.h`.

`kJSTypeObject`

An object value (meaning that this `JSValueRef` is a `JSObjectRef`).

Available in OS X v10.5 and later.

Declared in `JSValueRef.h`.

Availability

Available in OS X v10.5 and later.

Declared in

`JSValueRef.h`



Apple Inc.
Copyright © 2013 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Mac, and OS X are trademarks of Apple Inc., registered in the U.S. and other countries.

Java is a registered trademark of Oracle and/or its affiliates.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.