NodeJS Addonis

Description

NodeJS Addonis (NJSA) is a simplifying C wrapper for the nodejs addons construction released under the MIT license.

NJSA offers a deterministic and sequential generation interface for native node.js modules hiding all the **Google's V8** C++ stuff and exposing an API that lets you construct simple NodeJS Addons very easily. It lets you add fields, constants, arrays, functions and objects to your module just working with the C standard native types. Additionally, it also allows you to evaluate Javascript code to handle some special needs you may have.

The project page is: http://coderesearchlabs.com/nodejsaddonis

To learn more about how to use it, check the Usage section of this documentation.

NOTES:

- From what I see in the node community, many node programmers are Javascript developers who know little or nothing about C++ programming, and creating an addon for them might be a really difficult task to accomplish. This C wrapper try to simplify things for them.

- The project, originally, was developed using the nodejs 0.5.7 (unstable) compiled from the Bradley Meck's node.js fork branch called "windowsdll". That is not needed anymore: since 0.6.x the native Windows addons support is included in the official distribution, and this project has been updated to work with it.

- Besides it is developed on a 32-bit Windows, the source provided should work without issues on linux.

- Due to it's simplifying nature it might be somewhat restricting for you if you have more-than-simple requirements. To address them, you will have to craft your own solution, extend **NJSA** (in that case please let me know) and/or look somewhere else for other ideas. For example, take a look at:
  - https://github.com/rbranson/node-ffi
  - https://github.com/rmustacc/node-ctype
  - http://bravenewmethod.wordpress.com/2011/03/30/callbacks-from-threaded-node-js-c-extension/

- There are many ways in which **NJSA** can be expanded: enhanced support for functions, support for 64 bits Integer value type, support for Date value type, etc.

- If you want to read additional information on how to build nodejs addons on Windows, just check my article: http://coderesearchlabs.com/articles/BNWCA.pdf

Enjoy,
Javier Santo Domingo
j-a-s-d@coderesearchlabs.com

Usage

To code a nodejs addon with **NJSA** you have to include the node_addonis.h header file and implement in your code the routine *initialize()* which will be called automatically when your module is started up. Note that in case of Windows you will have to add also the DLLMain entry point.

```
#include <node_addonis.h>

void initialize()
{
    // here goes your code
}
```

In that moment is when you can build up your module by using the very straightforward routines that **NJSA** provides for that. For example it provides "new*Field()" routines for adding fields to your module with overloads for each kind of data type supported. The sample applies for constants with routines named like "new*Constant()" and so on.

```
newBooleanField("isEasy", true);
newStringField("message", "hello world");
newDoubleConstant("PI", 3.14);
```

For the special case of arrays and objects, which can hold it's own child members, there are "new", "select" and "close" operations, and "has" and "isSelected" operations for it's respective containers, to address any kind of building sequence you may need (for example, it let's you query an external resource to get information on how to build your module).

```
newObject("version");
newUnsignedIntegerConstant("MAJOR", 1);
newUnsignedIntegerConstant("MINOR", 0);
closeObject();
```

Additionally, **NJSA** allows you to add functions to your module and provides special functions to let you handle the any-amount-of-function-parameters that Javascript offers. That means your C functions are declared without parameters and handle arguments via "getArgumentsCount()" and the rest of the related functions.

```
void test()
{
    if (getArgumentsCount() > 0) {
        printf("test() invoked with parameters\n");
    } else {
        printf("test() invoked without parameters\n");
    }
    fflush(stdout);
}

...

newVoidFunction("test", test);
```

For a working sample of all this and more you can see the test library that accompanies this distribution.

To get a full list of routines provided by **NJSA**, just check the API section of this documentation.

API

NJSA offers the following routines to build up your module:

General

```
bool isModuleSelected();
bool hasMember(const char *name);
bool deleteMember(const char *name);
const char* evaluateScript(const char *script);
```

Fields

```
void newNullField(const char *name);
void newBooleanField(const char *name, bool value);
void newUnsignedIntegerField(const char *name, unsigned int value);
void newIntegerField(const char *name, int value);
void newDoubleField(const char *name, double value);
void newStringField(const char *name, const char *value);
```

Constants

```
void newNullConstant(const char *name);
void newBooleanConstant(const char *name, bool value);
void newUnsignedIntegerConstant(const char *name, unsigned int value);
void newIntegerConstant(const char *name, int value);
void newDoubleConstant(const char *name, double value);
void newStringConstant(const char *name, const char *value);
```

Arrays

```
void newArray(const char *name);
bool hasArray(const char *name);
bool selectArray(const char *name);
bool isArraySelected();
void closeArray();
```

Array Values

```
void newNullArrayValue();
void newBooleanArrayValue(bool value);
void newUnsignedIntegerArrayValue(unsigned int value);
void newIntegerArrayValue(int value);
void newDoubleArrayValue(double value);
void newStringArrayValue(const char *value);

void newNullArrayValues(int count);
void newBooleanArrayValues(bool values[], int count);
void newUnsignedIntegerArrayValues(unsigned int values[], int count);
void newIntegerArrayValues(int values[], int count);
void newDoubleArrayValues(double values[], int count);
void newStringArrayValues(const char *values[], int count);
```

Objects

```
void newObject(const char *name);
bool hasObject(const char *name);
bool selectObject(const char *name);
bool isObjectSelected();
void closeObject();
```

Functions

```
void newVoidFunction(const char *name, void callback());
void newBooleanFunction(const char *name, bool callback());
void newUnsignedIntegerFunction(const char *name, unsigned int callback());
void newIntegerFunction(const char *name, int callback());
void newDoubleFunction(const char *name, double callback());
void newStringFunction(const char *name, const char* callback());
```

Function Arguments

```
int getArgumentsCount();

bool isNullArgument(int argumentNumber);
bool isBooleanArgument(int argumentNumber);
bool isUnsignedIntegerArgument(int argumentNumber);
bool isIntegerArgument(int argumentNumber);
bool isDoubleArgument(int argumentNumber);
bool isStringArgument(int argumentNumber);
bool isArrayArgument(int argumentNumber);
bool isObjectArgument(int argumentNumber);
bool isFunctionArgument(int argumentNumber);

bool getBooleanArgument(int argumentNumber);
unsigned int getUnsignedIntegerArgument(int argumentNumber);
int getIntegerArgument(int argumentNumber);
double getDoubleArgument(int argumentNumber);
const char* getStringArgument(int argumentNumber);
```

License

Tools

In the development of **NodeJS Addonis** the following tools were specifically involved:

for source code editing and compiling
Microsoft's Visual C++ 2010 Express
http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express

When working at **Code Research Laboratories** the following tools are used:

in the test management field
Gurock Software's TestRail
http://www.gurock.com/testrail/

in the version control field
VisualSVN Ltd's VisualSVN
http://www.visualsvn.com/

in the similarity analysing field
RedHill Consulting's Simian
http://www.redhillconsulting.com.au/products/simian/

in the application lifecycle management field
Inedo's BuildMaster
http://www.inedo.com/buildmaster/

in the documentation field
EC Software's Help & Manual
http://www.ec-software.com/products_hm_overview.html

in the Java source code edition field
JetBrains's IntelliJ IDEA
http://www.jetbrains.com/idea/

in the bug tracking field
JetBrains's YouTrack
http://www.jetbrains.com/youtrack/

History

DATE	DESCRIPTION
2011.10.18	initial development
2011.10.26	first public release
2011.11.19	update to get it working with 0.6.x