dss_capi: An unofficial C API for EPRI's OpenDSS

If you are looking for the Python bindings, see dss_python.

Version 0.9.1, based on OpenDSS revision 2123. This is a work-in-progress but it's deemed stable enough to be made public.

This library exposes the OpenDSS engine in a plain C interface that tries to reproduce most of the COM methods. In fact, most of the code is derived from the COM implementation files. The resulting DLL can be using directly or through the `dss_python` module in Python, a module that mimics the COM structure (as exposed via `win32com` or `comtypes`), effectively enabling multi-platform compatibility at Python level.

Instead of using extra numeric parameters as in the official DDLL interface, each original COM property is exposed as a pair of functions. For example, the load kVA property is exposed as:

```
double Loads_Get_kva();
void Loads_Set_kva(double Value);
```

Besides low-level details such as memory management, most of the COM documentation can be used as-is.

Missing features and limitations

- Only the 64-bit version of OpenDSS is built. A 32-bit version should be possible with a few changes.
- Currently not implemented:
  - `DSSEvents` from `DLL/ImplEvents.pas`: seems too dependent on COM.
  - `DSSProgress` from `DLL/ImplDSSProgress.pas`: would need a reimplementation depending on the target UI (GUI, text, headless, etc.)
- Linux binaries are not yet available. For the time being, you need to build them yourself.

Extra features

Besides most of the COM methods, some of the unique DDLL methods are also exposed in adapted forms, namely the methods from `DYMatrix.pas`, especially `GetCompressedYMatrix` (check the source files for more information).

Building

To build the DLL yourself:

- Get the official OpenDSS source code for the target revision in the same root folder:

  svn checkout https://svn.code.sf.net/p/electricdss/code/trunk electricdss -r2123

- Install the FreePascal compiler. If you have the Lazarus IDE installed, you most likely already have the compiler too. Add the folder containing the compiler (`fpc.exe`) to your PATH environment variable.

- Get this repository:

  git clone https://github.com/PMeira/dss_capi.git

On Windows

If you just need the DLL, you can download it from the releases page. You might need to install the runtime for Microsoft Visual Studio 2017. Otherwise:

- Download the official repository.
- Install the x64 FreePascal compiler – see the wiki for further instructions.
- If you want to use the DLL from Visual Studio, you need to generate an import library. This can be done by starting the next step from the "x64 Native Tools Command Prompt for VS 2017" (or equivalent for your Visual Studio version) – you only need the `dumpbin.exe` and `lib.exe` utilities.
- Open a command prompt on the `dss_capi` folder and run `build.bat`

For the Windows build process, the `KLUSolve.dll` from the official OpenDSS repository is used. This may change in the future.

The output files will be placed into the `lib` folder.

On Linux

The current recommendation is to build your own KLUSolve, so you need to download install its dependencies. Since most distributions should include compatible SuiteSparse packages (which include the KLU library), a modified version of KLUSolve is included in the `klusolve` subfolder. Overall instructions:

- Install CMake and a C++ compiler

- Install the SuiteSparse development packages, preferably from your official distribution
- Install the x64 FreePascal compiler – see the wiki for further instructions.
- Build KLUSolve:

  ```
  cd dss_capi/klusolve
  cmake .
  make
  cd ..
  ```

- Build the main project:

  ```
  bash build.sh
  ```

Example

A minimal C example follows:

```
#include <stdint.h>
#include <stdio.h>
#include "dss_capi.h"

int main(void)
{
    double *voltages;
    int numNodes = 0, i;

    DSS_Start(0);
    Text_Set_Command("compile master.dss");
    Solution_Solve();
    Circuit_Get_AllBusVolts(&voltages, &numNodes);

    if (numNodes == 0)
    {
        return -1;
    }

    for (i = 0; i < numNodes; ++i)
    {
        printf("node %d: %f + j%f\n", i, voltages[2*i], voltages[2*(i + 1)]);
    }
```

```
    DSS_Dispose_PDouble(&voltages);

    return 0;
}
```

Testing

Currently all testing/validation is based on dss_python.

Roadmap

Besides bug fixes, the main funcionality of this library is mostly done. Notable desirable features that may be implemented are:

- More and better documentation, including the integration of the help strings from the IDL/COM definition files.
- Automate package building for some Linux distributions
- Automate validation of the Linux binaries (compare the outputs to the Windows version)
- C++ wrappers: Expose the API to C++ using namespaces for organization, overload methods, etc.

Other features that may include more invasive changes in the code base will probably be developed in another repository.

Questions?

If you have any question, feel free to open a ticket on Github or contact me through Twitter. Please allow me a few days to respond.

Credits / Acknowlegement

This project is derived from EPRI's OpenDSS and the same license is used. See `LICENSE` and `OPENDSS_LICENSE`, also check each subfolder for more details.

Note that, since OpenDSS depends on KLU via KLUSolve, the KLU licensing conditions (LGPL or GPL, depending on how you build KLU) apply to the resulting binaries; check the files `klusolve/COPYING`, `klusolve/lgpl_2_1.txt` and the SuiteSparse documentation.

I thank my colleagues at the University of Campinas, Brazil, for providing feedback and helping me test this project.