CENTER FOR CONSTRUCTION ENGINEERING AND MANAGEMENT

PLOTSTROBE

Dynamic Excel Chart Plotter for the Stroboscope Simulation System

By

Amir H. Behzadan and Vineet R. Kamat

UMCEE Report No. 06-01
Civil and Environmental Engineering Department
UNIVERSITY OF MICHIGAN
Ann Arbor, Michigan
January 2006

PREFACE

The idea of the present work has been brought up by professor Iris Tommelein (from the University of California at Berkeley) and professor Kamat during a recent discussion and was offered to the students in one of the classes I took with professor Kamat in winter 2005 called CEE633 (Construction Management Information Systems) here at the University of Michigan. That session and after having several lectures about how to write add-ons for Stroboscope, a corresponding homework was assigned containing the original idea which later served as my motivation to prepare this report.

I would like to thank professor Kamat especially for his continuous support and innovative ideas. His unique way of teaching classes and introducing new concepts to the students has always been a good motivation for all his students including me. I also have the opportunity to do my Ph.D. work under his supervision which makes me more resolute and determined in learning new things related to my field of research.

I would also like to take this opportunity to thank professor Martinez (from Virginia Polytechnic Institute and State University). Stroboscope, which I have extensively used as a reference simulation tool during my previous work and courses such as CEE631 (Construction Decisions under Uncertainty) and CEE633 (Construction Management Information Systems), is in fact the final outcome of his Ph.D. thesis here at the University of Michigan.

TABLE OF CONTENTS

SECTION 1 – Introduction **4**

SECTION 2 – PLOTSTROBE Statements and Functions **4**

2.1. OPENEXCEL Statement 4
2.2. CREATECHART Statement 5
2.3. ADDSERIES Statement 5
2.4. APPENDDATA Function 6
2.5. SETSLEEP Statement 7
2.6. CLOSEEXCEL Statement 7

SECTION 3 – PLOTSTROBE Example **7**

SECTION 4 – Developing PLOTSTROBE Add-On **11**

4.1. Add-On Interface 11
4.2. Registering Statements 13
4.3. Registering Functions 13

SECTION 5 – VINEXCEL Dynamic Link Library Functions **14**

5.1. Open_Excel() 15
5.2. Create_New_XY_Chart() 15
5.3. Add_New_Data_Series() 16
5.4. Append_Existing_Data_Series() 16
5.5. Close_Excel() 17

SECTION 6 – PLOTSTROBE Listing **17**

1. Introduction

One of the powerful features of Stroboscope is its capability of being extended beyond its initial scope of application. This report describes the development process of a new add-on named *Dynamic Excel Chart Plotter (PLOTSTROBE)* using standard compiled programming language of C++.

PLOTSTROBE is intended to be used as a real time chart plotting tool which can be conveniently integrated in the simulation input file to add graphical output of almost every properties of different elements in a model upon user's discretion. PLOTSTROBE provides the Stroboscope user with a set of new statements and functions that add real time communication capability between Stroboscope core language and Microsoft Excel. To be more specific, five new statements of OPENEXCEL, CREATECHART, ADDSERIES, SETSLEEP, and CLOSEEXCEL and one new function called APPENDDATA have been added to Stroboscope using PLOTSTROBE add-on.

2. PLOTSTROBE Statements and Functions

As mentioned earlier, there are five statements and one function provided by PLOTSTROBE. In the following sub-sections, more details about each of these statements and functions are presented.

2.1. OPENEXCEL Statement

This statement takes no argument. Its role is to open Microsoft Excel. To use this statement, the following syntax is recommended:

Syntax: `OPENEXCEL;`

Example: `OPENEXCEL;`

2.2. CREATECHART Statement

This statement takes seven arguments. Its role is to create a new chart in Excel, add one data series to that chart, and plot the first point on the chart. To use this statement, the following syntax is recommended:

```
Syntax:     CREATECHART ChartName ChartTitle XLabel
                            YLabel FirstSeriesLabel
                            X1 Y1;
Example:    CREATECHART Concrete Hopper1 time
                            content series1
                            0 1;
```

The corresponding code for this statement uses two internal numbering systems for distinguishing between different charts and data series that are created independently in any Stroboscope input file. To do so, an ascending numbering concept is used which means the first chart/ series created in the input file is given number 1 for future purposes and so on. This is especially important when additional points need to be added to an existing chart using `APPENDDATA` function. To be more specific, each time `CREATECHART` is called, two new indices are assigned one to the new chart and the other to the first series in that chart.

2.3. ADDSERIES Statement

This statement takes four arguments. Its role is to create a new data series to an existing chart in Excel and plot the first point in that data series. To use this statement, the following syntax is recommended:

```
Syntax:     ADDSERIES ChartName SeriesLabel  X1 Y1;
Example:    ADDSERIES Concrete series2  5 7;
```

The corresponding code for this statement also uses the internal numbering system for data series described in previous section. Again, each time `ADDSERIES` is called, a new index is assigned to the new data series in that chart. In present version of PLOTSTROBE, the two indexing systems (i.e. for charts and series) are designed and implemented independently. That means each time a new chart is created, the value of the corresponding index is incremented by one independently of what the value of the other index is. Similarly, each time a new data series is created the corresponding index is incremented by one no matter to which chart that series belong to.

To give an example, if the Stroboscope user uses `CREATECHART` to create two charts with the first one having two data series (the second of which created using `ADDSERIES`) and the second one has only one data series, the chart indices will be 1 and 2 (1 for the first chart and 2 for the second chart) while the data series indices will be 1, 2, and 3 (1 and 2 for the data series in the first chart and 3 for the data series in the second chart).

2.4. APPENDDATA Function

This statement takes four arguments. Its role is to add one data point at a time to an existing series in an existing chart. To use this statement, the following syntax is recommended:

Syntax: `APPENDDATA [ChartNumber,SeriesNumber,X,Y];`

Example: `APPENDDATA [1,2,5,15];`

Note that `ChartNumber` and `SeriesNumber` are based on the numbering system described earlier. As a result, the user should keep track of which chart and data series has been created first, second, and so on.

2.5. SETSLEEP Statement

This statement takes one argument which is the number of milliseconds between plotting of two consecutive points on the chart. Its role is to delay the immediate plotting of data points on the chart so the chart is plotted smoothly on the screen as opposed to being plotted as fast as the simulation time progresses. To use this statement, the following syntax is recommended:

Syntax:	`SETSLEEP DelayInMS;`
Example:	`SETSLEEP 5000;`

2.6. CLOSEEXCEL Statement

This statement takes no argument. Its role is to close Microsoft Excel. To use this statement, the following syntax is recommended:

Syntax:	`CLOSEEXCEL;`
Example:	`CLOSEEXCEL;`

3. PLOTSTROBE Example

The following is a part of a Stroboscope input file which loads PLOTSTROBE add-on and integrates the provided statements and functions to plot two charts in Excel.

```
…………
SAVEVALUE C1S1 0;
SAVEVALUE C1S2 0;
SAVEVALUE C2S1 0;

VARIABLE SimulationTime SimTime;
VARIABLE ConcQAmt ConcInPierHpr.CurCount;
VARIABLE PipesQNum PipesOnRack.CurCount;

LOADADDON PlotStrobe.dll;

SETSLEEP 1000;

OPENEXCEL;

CREATECHART ChartC2 'Pipes on Rack Statistics' 'Simulation Time'
                      'Pipes on Rack' 'Pipes' 0 0;

CREATECHART ChartC1 'Available Concrete Statistics'
                      'Simulation Time' 'Concrete In Hopper'
                      'Total Concrete' 0 0;

ADDSERIES ChartC1 'Concrete Stored' 0 0;

ONENTRY ConcInPierHpr ASSIGN C1S1
                      APPENDDATA[2,2,SimulationTime,ConcQAmt];

ONDRAW CO3 ASSIGN C1S1
                      APPENDDATA[2,2,SimulationTime,ConcQAmt];

ONENTRY ConcInPierHpr ASSIGN C1S2
                      APPENDDATA[2,3,SimulationTime,ConcQAmt];

ONENTRY PipesOnRack ASSIGN C2S1
                      APPENDDATA[1,1,SimulationTime,PipesQNum];
…………
```

The goal in this example is to plot two separate real-time charts one showing the amount of concrete in the hopper and the other showing the number of concrete pipe segments on the rack as the simulation time progresses. For this reason, three variables are defined to keep track of simulation time `SimTime` (i.e. `SimulationTime`), amount of concrete in the hopper `ConcInPierHpr.CurCount` (i.e. `ConcQAmt`), and number of pipe segements on rack `PipesOnRack.CurCount` (i.e. `PipesQNum`). `LOADADDON` (which is a registered core Stroboscope statement) is used then to load the corresponding DLL file to PLOTSTROBE add-on (i.e. `PlotStrobe.dll`).

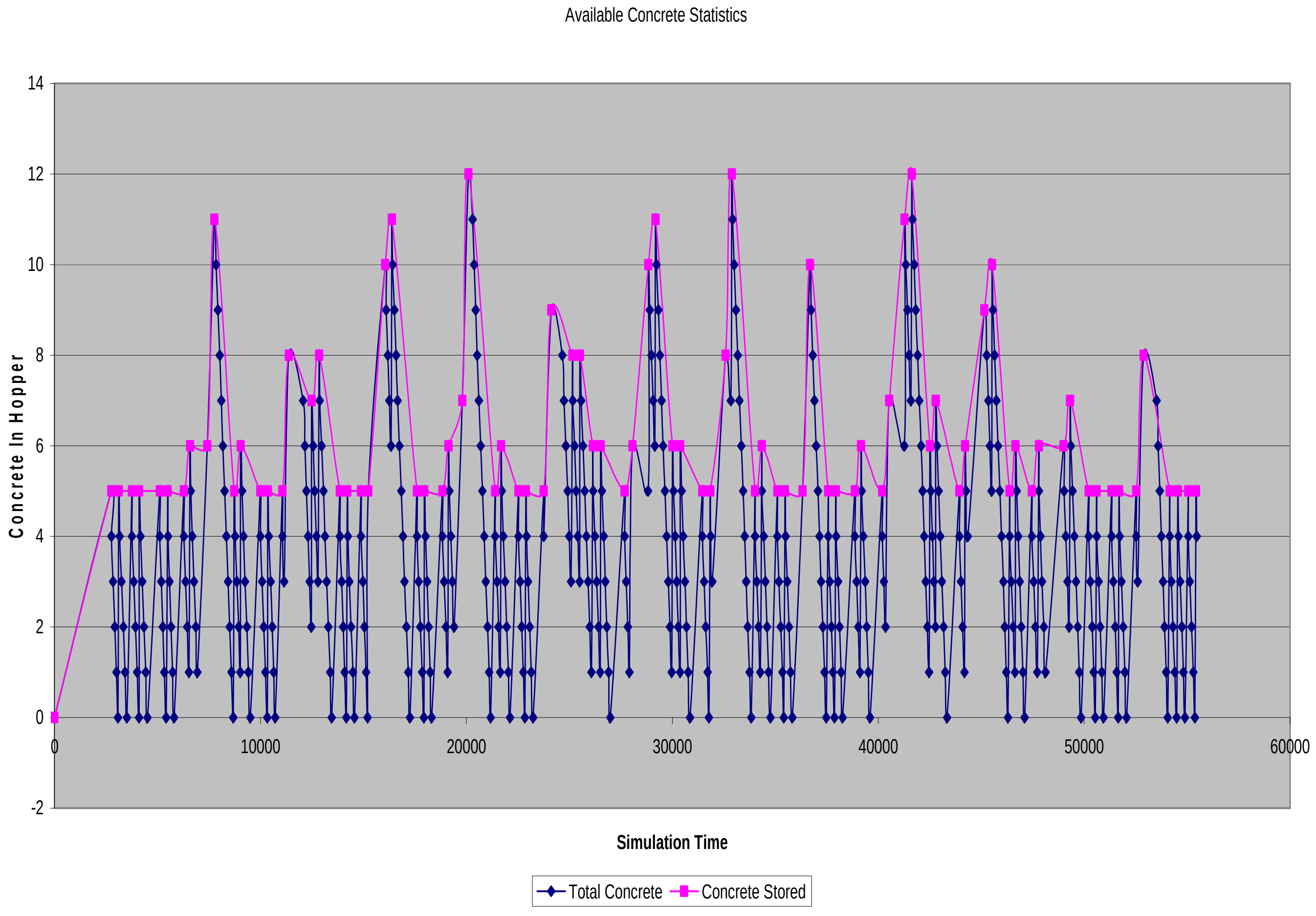
The corresponding charts and first data series in each chart together with the first point can be created using `CREATECHART` statement. After this statement is executed, the created charts will be given indices 1 and 2 respectively for future purposes. It is clear that the `CREATECHART` statement as executed above will create two charts named `ChartC1` and `ChartC2`. The second data series to the first chart is also created using `ADDSERIES`.

The next step is to complete the chart by adding one point at a time as the simulation time progresses. To do so, Stroboscope core statements `ONENTRY` and `ONDRAW` are used. For the first chart, `ONENTRY` keeps track of each concrete resource entering the hopper via queue `ConcInPierHpr` and `ONDRAW` keeps track of each concrete resource leaving the hopper through link `CO3` (note that *concrete* is treated as a generic resource in this example and as a result is a countable entity). The way these statements are used in the current example, causes a new point to be appeared on the chart as soon as the resource concrete is either entering the hopper or leaving it. For the second chart, `ONENTRY` keeps track of each pipe segment placed on the rack via queue `PipesOnRack`.

Because `ONENTRY` and `ONDRAW` accept only target functions as second argument, `APPENDATA` can not be directly used in front of them. As a result, dummy variables `C1S1`, `C1S2`, and `C2S1` are defined (using `SAVEVALUE` core statement) and used. This way, each time `ONENTRY` or `ONDRAW` are executed, the value of the corresponding dummy variable in front of them is updated using the return value of `APPENDATA` (which is always 1). During this being done, `APPENDATA` is evaluated and a new point is added to the chart.

Note that this is in fact a little trick to overcome the limitation of only using action targets in front of Stroboscope core statements. Action targets are limited to a number of predefined statements including `ASSIGN`. Also note that since indices 1 and 2 have been assigned to the two charts, and also indices 1, 2, and 3 have been assigned to the data series, the correct indices should be used as arguments for `APPENDATA` every time a new point should be added to each chart.

Also by using `SETSLEEP` statement and set the sleep time to 1000 milliseconds, there is a one second delay between plotting two consecutive points on the chart. Figure 1 shows two Excel outputs created using PLOTSTROBE add-on in the current example.



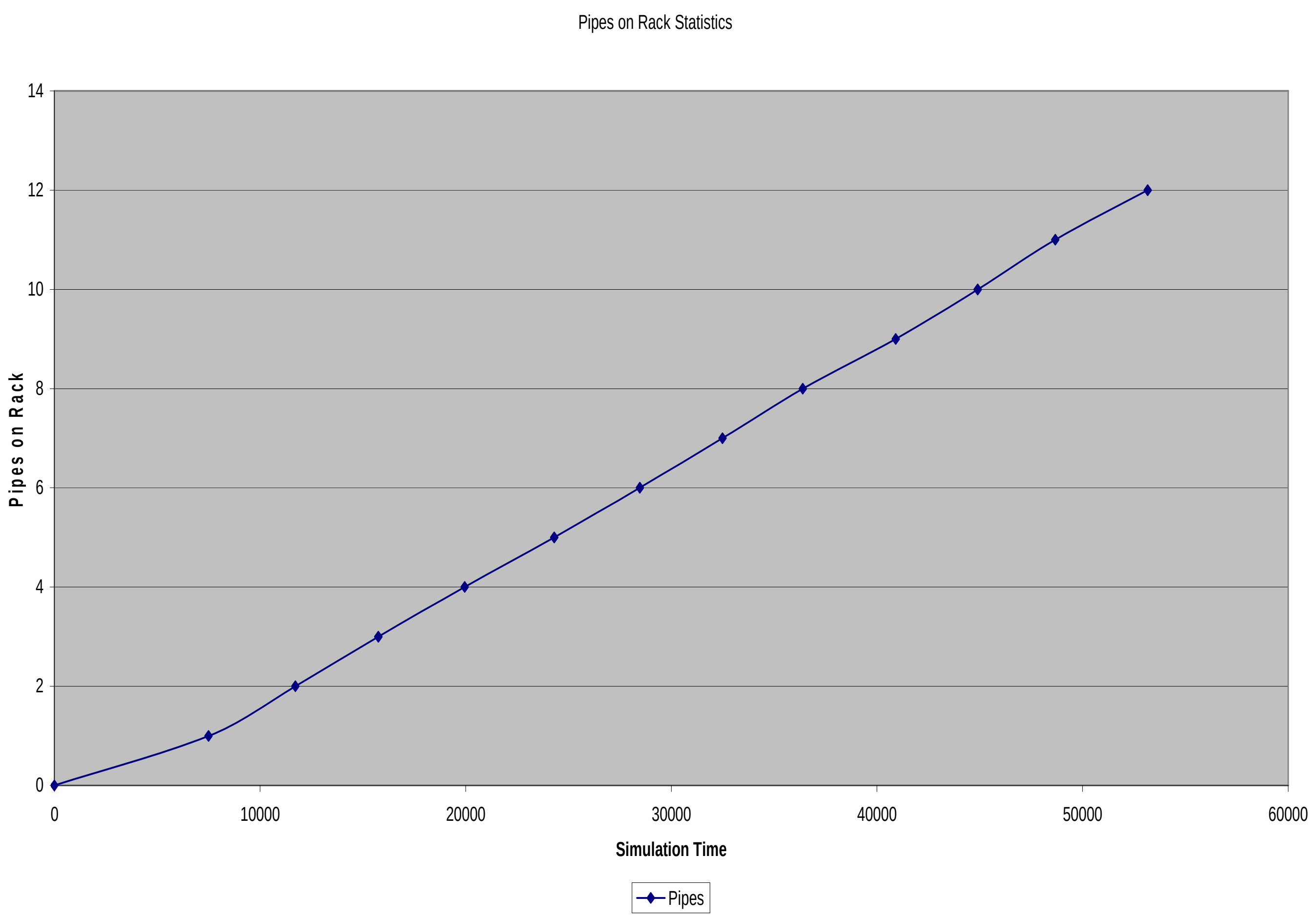


Figure 1 – PLOTSTROBE sample Excel outputs

4. Developing PLOTSTROBE Add-On

The function prototypes and constant definitions used in the add-on listing can be found in *Strobosc.h* and also *Strobosc.lib* can be used for linking. Complete details about developing new add-ons for Stroboscope can be found in Stroboscope manual.

4.1. Add-On Interface

When Stroboscope encounters a LOADADDON statement, it loads the dynamic link library (DLL) specified as argument and searches for a function exported as *StroboAddOnInit*. If Stroboscope does not find *StroboAddOnInit* it issues a compilation error indicating that the add-on initialization function was not found. *StroboAddOnInit* must follow the *__stdcall* calling convention, return a 32-bit integer, and take a NULL terminated constant ANSI string as argument. PLOTSTROBE uses the following listing as the *StroboAddOnInit* function:

```
int __stdcall StroboAddOnInit(const char* szModelName)
{
//show on the error device that Stroboscope
//has loaded this Add-on
     PrintToStdError("\r\nPlotStrobe Add-On for Stroboscope");

     //Register with Stroboscope the statements
     //exposed by the add-on
     RegStatement("OPENEXCEL",&OpenExcel);
     RegStatement("CREATECHART",&CreateChart);
     RegStatement("ADDSERIES",&AddSeries);
     RegFunction("APPENDDATA",&AppendData,0,FALSE);
     RegStatement("SETSLEEP",&SetSleep);
     RegStatement("CLOSEEXCEL",&CloseExcel);

     return true;
    }//end StroboAddOnInit
```

The function is exported by using the following entry in the exports section of the module definition (*.def) file:

```
EXPORTS
     StroboAddOnInit       @2
```

The main reason to do that is to provide a place for registering `StroboAddOnInit` function. Otherwise, Stroboscope will not recognize the add-on statements and functions as they are called in Stroboscope input file.

4.2. Registering Statements

The statements exported by the add-on should be coded as functions with the following prototype:

```
int __stdcall Statement (const char* StatementArguments);
```

Four such statements have been used in PLOTSTROBE as follows:

```
int __stdcall OpenExcel(const char* szArguments);
int __stdcall CreateChart(const char* szArguments);
int __stdcall AddSeries(const char* szArguments);
int __stdcall SetSleep(const char* szArguments);
int __stdcall CloseExcel(const char* szArguments);
```

These statements have been registered by calling *RegStatement* from within Stroboscope add-on function as follows:

```
RegStatement("OPENEXCEL",&OpenExcel);
RegStatement("CREATECHART",&CreateChart);
RegStatement("ADDSERIES",&AddSeries);
RegStatement("SETSLEEP",&SetSleep);
RegStatement("CLOSEEXCEL",&CloseExcel);
```

4.3. Registering Functions

The functions exported by the add-on should be coded as functions with one of the following prototypes depending on the number of arguments required by the function:

```
double __stdcall FunctionWithNoArgs();
double __stdcall FunctionWith1Arg (double Arg1);
double __stdcall FunctionWith2Args (double Arg1,
                                        double Arg2);
```

```
    double __stdcall FunctionWith3Args (double Arg1,
                                        double Arg2,
                                        double Arg3);
    double __stdcall FunctionWith4Args (double Arg1,
                                        double Arg2,
                                        double Arg3,
                                        double Arg4);
```

One such function has been used in PLOTSTROBE as follows:

```
    double __stdcall AppendData(double Arg1, double Arg2,
                                double Arg3, double Arg4);
```

This function has been registered by calling *RegFunction* from within Stroboscope add-on function as follows:

```
    RegFunction("APPENDDATA",&AppendData,4,FALSE);
```

This function is also exported by using the following entry in the exports section of the module definition (*.def) file:

```
EXPORTS
    AppendData       @3
```

5. VINEXCEL Dynamic Link Library Functions

In order to provide graphical output, Microsoft Excel is used as the main platform in PLOTSTROBE add-on. Communication with Excel is done through a provided dynamic link library (DLL) called *vinexcel.dll* which includes several internal functions to perform different tasks in Excel.

In PLOTSTROBE add-on, five main functions provided by this DLL are used which are *Open_Excel(), Create_New_XY_Chart(), Add_New_Data_Series(),* *Append_Existing_Data_Series()*, and *Close_Excel()*. These functions basically give the ability to the user to open Excel, create a chart, manipulate data in real time, and finally close Excel.

5.1. Open_Excel()

This function takes no argument and returns 1 for success. Its role is to locate Microsoft Excel in the system, open the application, and give handle to the other functions in the same DLL file. To use this function, the following syntax is recommended:

```
Syntax:         Open_Excel();
Example:        Open_Excel();
```

5.2. Create_New_XY_Chart()

This function takes seven arguments in form of C++ standard string and returns 1 for success. Its role is to open a new chart in Excel, open a data series in that chart, make chart title, chart name, X axis label, Y axis label, and finally plot the first point on that chart. To use this function, the following syntax is recommended:

```
Syntax:         Create_New_XY_Chart(string ChartName,
                                    string ChartTitle,
                                    string XLabel,
                                    string YLabel,
                                    string FirstSeriesLabel,
                                    string X1,
                                    string Y1);

Example:        Create_New_XY_Chart(“My Chart”,
                                    “Average Hopper Content”,
                                    “Time (s)”,
                                    “Concrete (CY)”,
                                    “Simulation 1”,
                                    “0”,
                                    “10”);
```

5.3. Add_New_Data_Series()

This function takes four arguments in form of C++ standard string and returns 1 for success. Its role is to add a new data series to an existing chart in Excel and plot the first point on that series. To use this function, the following syntax is recommended:

```
Syntax:     Create_New_XY_Chart(string ChartName,
                                   string NewSeriesLabel,
                                   string X1,
                                   string Y1);

Example:    Create_New_XY_Chart(“My Chart”,
                                   “Simulation 2”,
                                   “5”,
                                   “17”);
```

5.4. Append_Existing_Data_Series()

This function takes four arguments in form of C++ standard string and returns 1 for success. Its role is to add one point per call to an existing data series in an existing chart. To use this function, the following syntax is recommended:

```
Syntax:     Add_New_Data_Series(string szExistingChartName,
                                   string szNewSeriesLabel,
                                   string szX1,
                                   string szY1);
```

```
Example:        Add_New_Data_Series(“My Chart”,
                                    “Simulation 1”,
                                    “5”,
                                    “23”);
```

5.5. Close_Excel()

This function takes no argument and returns 1 for success. Its role is to close the Excel application in context. To use this function, the following syntax is recommended:

```
Syntax:         Close_Excel();
Example:        Close_Excel();
```

6. PLOTSTROBE Listing

The following is the source code for the PLOTSTROBE add-on. It is written in C++ using the string, and map classes. The source code has been compiled in release version with Microsoft Visual C++.NET version 2003. The add-on is implemented in two files: *StroboAddOn.def* and *StroboAddOn.cpp*. *StroboAddOn.def* is very short:

```
LIBRARY      StroboAddOn

EXPORTS
        StroboAddOnInit  @2
        AppendData      @3
```

StroboAddOn.cpp is much longer. The comments should make the source code self-explanatory:

```
// Stroboscope Add-On : PLOTSTROBE (Dynalic Excel Chart Plotter)
// By : Amir H. Behzadan, Vineet R. Kamat
// CEE Department - University of Michigan, Ann Arbor, MI
// January 2006

// Stroboscope header and library files are loaded

#include "C:\Program Files\Stroboscope\SDK\strobosc.h"
#include "C:\ Program Files\VinExcel\vinexcel.h"
#pragma comment (lib, "C:\\Program Files\\Stroboscope\\SDK
                                            \\strobosc.lib")
#pragma comment (lib, "C:\\ Program Files\\VinExcel
                                            \\vinexcel.lib")

// Standard C++ header files and data types are loaded

#include <windows.h>
#include <list>
#include <map>
#include <string>

using namespace std;

// Working buffer for string manipulation

char  szScratch[MAX_STATEMENT_LENGTH];

// Global variables

map <int,string> ChartN;
map <int,string> SeriesN;
static int m = 1;
static int n = 1;
int SleepTime = 0;

// Function prototypes for the statements registered by the
// Add-On.

// Appendata should be exported in DEF file since it is a
// function.

int __stdcall OpenExcel(const char* szArguments);
int __stdcall CreateChart(const char* szArguments);
int __stdcall AddSeries(const char* szArguments);
double __stdcall AppendData(double Arg1, double Arg2, double
Arg3, double Arg4);
int __stdcall SetSleep(const char* szArguments);
int __stdcall CloseExcel(const char* szArguments);

//The Add-On initialization function - must be exported in DEF
// file.
//This is the function first looked for after the Add-On is
```

```
// loaded using the corresponding DLL file.

int __stdcall StroboAddOnInit(const char* szModelName)
{
     // Show on the error device that Stroboscope has loaded
     // this Add-on

     PrintToStdError("\r\nPlotStrobe Add-On for Stroboscope
                     Version 1,0,0,0");
     PrintToStdError("\r\nBy Amir H. Behzadan and Vineet R.
                     Kamat");
     PrintToStdError("\r\n(C) 2006 University of Michigan\r\n");

     //Show on the output device that Stroboscope has loaded
     // this Add-on

     PrintToStdOutput("\r\nPlotStrobe Add-On for Stroboscope
                     Version 1,0,0,0");
     PrintToStdOutput("\r\nBy Amir H. Behzadan and Vineet R.
                     Kamat");
     PrintToStdOutput("\r\n(C) 2006 University of
                     Michigan\r\n");

     // Register with Stroboscope the statements exposed by the
     // add-on

     RegStatement("OPENEXCEL",&OpenExcel);
     RegStatement("CREATECHART",&CreateChart);
     RegStatement("ADDSERIES",&AddSeries);
     RegFunction("APPENDDATA",&AppendData,4,FALSE);
     RegStatement("SETSLEEP",&SetSleep);
     RegStatement("CLOSEEXCEL",&CloseExcel);

     return true;
} // End of StroboAddOnInit

// The function registered with Stroboscope as OPENEXCEL

int __stdcall OpenExcel(const char* szArguments)
{
     if ( false == Open_Excel() )
     {
          char szMessage[256];
          sprintf(szMessage, "VinExcel encountered an error in
                     opening Excel");
          MessageBox(NULL, szMessage, "PlotStrobe Error", MB_OK
                     | MB_ICONERROR);

          // Print the error on Stroboscope error console too

          PrintToStdError("\r\nVinExcel encountered an error in
                          opening Excel\r\n");
```

```
            // Returns the error to Stroboscope

            StrbMathError("OPENEXCEL", STR_UNDEFINED);


            // Return false to Stroboscope to indicate failure

            return false;
        }

        return true;
}

// The function registered with Stroboscope as CREATECHART

int __stdcall CreateChart(const char* szArguments)
{
        int k=0;
        int h=0;
        list <string> ArgC;
        list<string>::iterator i;
        string StrC[7];

        strncpy(szScratch,szArguments,MAX_STATEMENT_LENGTH);
        char* szWork=szScratch;

        // Extract all the arguments

        while(strlen(szWork))
            ArgC.push_back(ExtractArgument(szWork));

        // Check the number of arguments

        size_t nArgsInList = ArgC.size();

        // Get the arguments
        // The arguments are sotred in StrC array consequently.

        for(i = ArgC.begin(); i != ArgC.end(); i++){
          StrC[k]=*i;
            k++;
        }

        // A dynamic mapping system is used to assign a number to
        // each chart which includes one data series itself. It is
        // based on one-unit incremental each time a NEW chart or a
        // NEW data series is created.
        // Each time a new chart is created, the indices for both
        // the chart names and the series names are incremented by
        // one to point to the next number in the mapping system.

        if ( false == Create_New_XY_Chart(StrC[0], StrC[1],
            StrC[2], StrC[3], StrC[4], StrC[5], StrC[6]) )
        {
            char szMessage[256];
            sprintf(szMessage, "VinExcel encountered an error
                    creating chart -> %s", StrC[0].c_str() );
            MessageBox(NULL, szMessage, "PlotStrobe Error", MB_OK
                       | MB_ICONERROR);

            // Print the error on Stroboscope error console too

            PrintToStdError(szMessage);

            // Closes Excel and returns the error to Stroboscope

            Close_Excel();
            StrbMathError("CREATECHART", STR_UNDEFINED);

            // Return false to Stroboscope to indicate failure

            return false;
        }

        ChartN[m]=StrC[0];
        SeriesN[n]=StrC[4];
        m++;
        n++;

        return true;
}

//The function registered with Stroboscope as ADDSERIES

int __stdcall AddSeries(const char* szArguments)
{
        int k=0;
        list <string> ArgC;
        list<string>::iterator i;
        string StrC[4];

        strncpy(szScratch,szArguments,MAX_STATEMENT_LENGTH);
        char* szWork=szScratch;

        // Extract all the arguments

        while(strlen(szWork))
            ArgC.push_back(ExtractArgument(szWork));

        // Check the number of arguments

        size_t nArgsInList = ArgC.size();

        // Get the arguments
        // The arguments are sotred in StrC array consequently.
        for(i = ArgC.begin(); i != ArgC.end(); i++){
            StrC[k]=*i;
                k++;
        }

        // A dynamic mapping system is used to assign a number to
        // each new data series. It is based on one-unit
        // incremental each time a NEW data series is created.
        // Each time a new data series is created, the index for
        // the series names is incremented by one to point to the
        // next number in the mapping system.

        if ( false == Add_New_Data_Series(StrC[0], StrC[1],
                                    StrC[2], StrC[3]) )
        {
                char szMessage[256];
                sprintf(szMessage, "VinExcel encountered an error
                        adding data series -> %s", StrC[0].c_str() );
                MessageBox(NULL, szMessage, "PlotStrobe Error", MB_OK
                                | MB_ICONERROR);

                // Print the error on Stroboscope error console too

                PrintToStdError(szMessage);

                // Closes Excel and returns the error to Stroboscope

                Close_Excel();
                StrbMathError("ADDSERIES", STR_UNDEFINED);

                // Return false to Stroboscope to indicate failure

                return false;
        }

        SeriesN[n]=StrC[1];
        n++;

        return true;
}

// The function registered with Stroboscope as APPENDDATA
// "Arg1" refers to chart index
// "Arg2" refers to data series index
// The numbering system is based on first created - first
// numbered. Hence, the first chart/ series that has been created
// using CREATECHART command, will be called "1" thereafter and
// so on.

double __stdcall AppendData(double Arg1, double Arg2, double
                                Arg3, double Arg4) {

        char Arg3C[1024], Arg4C[1024];
```

```
      string Arg3S, Arg4S;

      sprintf(Arg3C,"%f",Arg3);
      sprintf(Arg4C,"%f",Arg4);

      Arg3S=Arg3C;
      Arg4S=Arg4C;

      if ( false ==
           Append_Existing_Data_Series(ChartN[int(Arg1)],
                      SeriesN[int(Arg2)], Arg3S, Arg4S) )
      {
           char szMessage[256];
           sprintf(szMessage, "VinExcel encountered an error
                      appending data in Series %d in Chart %d",
                      int(Arg1), int(Arg2) );
           MessageBox(NULL, szMessage, "PlotStrobe Error", MB_OK
                      | MB_ICONERROR);

           // Print the error on Stroboscope error console too

           PrintToStdError(szMessage);

           // Closes Excel and returns the error to Stroboscope

           Close_Excel();
           StrbMathError("APPENDDATA", STR_UNDEFINED);

           // Return false to Stroboscope to indicate failure

           return false;
      }

      Sleep(SleepTime);

      return true;
}

// The function registered with Stroboscope as SETSLEEP

int __stdcall SetSleep(const char* szArguments) {

      strncpy(szScratch,szArguments,MAX_STATEMENT_LENGTH);
      char* szWork=szScratch;

    // Extract all the arguments

    std::list<std::string> rgszTokens;
    while(strlen(szWork))
        rgszTokens.push_back(ExtractArgument(szWork));

    // Check the number of arguments
    size_t nArgsInList = rgszTokens.size();
    if (nArgsInList != 1 ) PrintToStdError("\nError: SETSLEEP
                           statement requires 1 argument");

      std::list<std::string>::iterator myListIterator =
                           rgszTokens.begin();
     std::string stringSleepTime  = *myListIterator++;
     SleepTime =  atoi(stringSleepTime.c_str());

     return true;
}


// The function registered with Stroboscope as CLOSEEXCEL

int __stdcall CloseExcel(const char* szArguments)
{
      if ( false == Close_Excel() )
      {
           char szMessage[256];
           sprintf(szMessage, "VinExcel encountered an error in
                                closing Excel");
           MessageBox(NULL, szMessage, "PlotStrobe Error", MB_OK
                      | MB_ICONERROR);

           // Print the error on Stroboscope error console too

           PrintToStdError("\r\nVinExcel encountered an error in
                           closing Excel\r\n");

           // Returns the error to Stroboscope

           StrbMathError("CLOSEEXCEL", STR_UNDEFINED);

           // Return false to Stroboscope to indicate failure

           return false;
      }

      return true;
}
```

It is worth mentioning that for an Add-on to be recognized by Stroboscope, the source code has to be compiled in release version. In other words, only release DLLs can be uploaded and used as add-ons in Stroboscope.