

# ColumnBlaze Concrete Column Library

## Quick Guide

### Computations & Graphics, Inc.

Highlands Ranch, CO 80130, USA

Email: [info@cg-inc.com](mailto:info@cg-inc.com)

Web: [www.cg-inc.com](http://www.cg-inc.com)

# Introduction

Computations & Graphics, Inc. (CGI) ColumnBlaze Library is a powerful concrete column API (Application Programming Interface) to calculate section capacity of rectangular, circular, and generic sections. It is based on the solver engine in our cColumn concrete analysis and design software. You can use this reliable and user-friendly API to develop your custom software royalty-free.

The following are the main features:

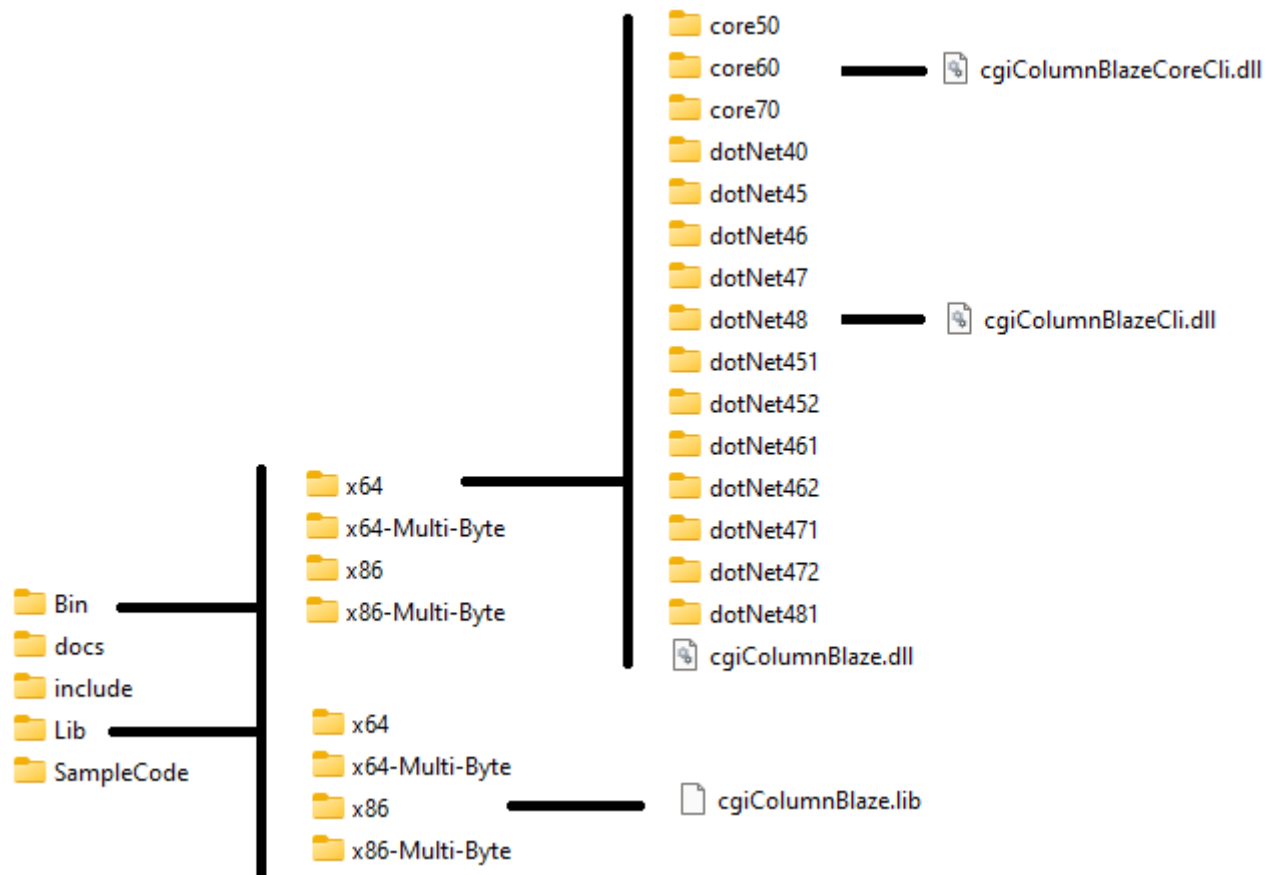
- Supports rectangular, circular, and generic (with openings) sections.
- Easy-to-use programming interfaces to define input, solve, and retrieve results for one or multiple sections.
- Supports P-Mx-My interaction diagram at different angles.
- Supports Mx-My diagram for a given axial load.
- Calculates unity ratio for a section under a given load (P, Mx, My).
- Provides interface to export sections input to cColumn input file. You can then use cColumn to open and graphically view geometry, reinforcement, and result diagrams.
- Support various American Concrete Institute codes including ACI 318-19, 14, -11, -08, -05, -02, and -99.
- Supports .NET Framework 4.0–4.8 and modern .NET 5.0–10.0 in both C# and VB.NET.
- Supports native C++ language.
- Supports both 64-bit and 32-bit CPUs architecture on Windows.
- Supports Unicode and Non-Unicode in Microsoft Visual Studio 2015 and newer versions, including Visual Studio 2026.
- Available in both binary library and source code forms.
- A free copy of Professional cColumn Program.
- Numerous source code samples in C++ and C# are freely available.
- Technical support by the ColumnBlaze author.
- Reasonably priced and royalty-free.

*Sample code included in the install can be readily compiled and run using Visual Studio 2026. You should be able to open the sample code in Visual Studio 2015 and above. For distribution, you may need to install Visual C++ 2015~2026 x86 or x64 redistributables as the ColumnBlaze library links dynamically with the visual C++ runtime library.*

*Since ColumnBlaze is based on the concrete column software cColumn, it is highly recommended that you install cColumn and get familiar with its technical backgrounds and capabilities. You can download an evaluation of cColumn from <http://www.cg-inc.com/download/download>.*

The following image shows the file structure of the SDK. It includes a **Bin** folder which contains executables for both native DLLs, .Net Framework and .Net (Core) assemblies for different CPUs platform and character sets, an **Include** folder which contains C++ header files, a **Lib** folder which contains static library for C++ linking, a **SampleCode** folder which contains example C++ and .Net code for Visual Studio 2026.

The file `cgiColumnBlaze.dll` is a Windows native DLL. Make sure these files have consistent CPU architecture and are present in the executable directory.



# C++ Interface

The C++ library contains both a 64-bit and 32-bit Windows DLL `cgiColumnBlaze.dll`. The interface includes the following header files: `_cgiDefines.h`, `_cgiIColumnStructure.h`, `_cgiPlatform.h` and `cgiColumnBlaze.h`. It also includes a `cgiColumnBlaze.lib` for linking to your projects.

The `_cgiDefines.h` defines all input and output data structures. `_cgiIColumnStructure.h` is the one and only interface to set input, perform analysis and retrieve output. The best way to learn how you use these data structures and interfaces is to study the examples included in the C++ console application project `cgiColumnBlazeClient`.

The following lists all the interface functions exposed by `cgiIColumnStructure`:

```
// function callbacks to notify the clients
typedef void (*fnLISTMSG)(LPCTSTR sz0, LPCTSTR sz1);
typedef void (*fnSTATUSMSG)(LPCTSTR sz);
typedef int (*fnMKLPROGRESS)(int* ithr, int* step, char* stage, int len);

struct CGICOLUMNBLAZE_API cgiIColumnStructure
{
    virtual void setListMessageFunction(fnLISTMSG fnListMsg) = 0;
    virtual void setStatusMessageFunction(fnSTATUSMSG fnStatusMsg) = 0;

    virtual void getExePath(TCHAR exePath[], int size) const = 0;

    virtual void setProjPath(const TCHAR projPath[]) = 0;
    virtual void getProjPath(TCHAR projPath[]) const = 0;

    virtual void setModelName(const TCHAR modelName[]) = 0;
    virtual void getModelName(TCHAR modelName[]) const = 0;

    virtual void setDesignCompany(const TCHAR designCompany[]) = 0;
    virtual void getDesignCompany(TCHAR designCompany[]) const = 0;

    virtual void setEngineer(const TCHAR engineer[]) = 0;
    virtual void getEngineer(TCHAR engineer[]) const = 0;

    virtual void setNotes(const TCHAR notes[]) = 0;
    virtual void getNotes(TCHAR notes[]) const = 0;

    // LENGTH=ft; DIMENSION=in; FORCE=kip; FORCE_LINE=kip/ft; MOMENT=kip-ft; FORCE_SURFACE=lb/ft^2;
    // DISPLACEMENT_TRANS=in; DISPLACEMENT_ROTATE=rad; MODULUS=kip/in^2; WEIGHT_DENSITY=lb/ft^3; STRESS=lb/in^2
    // SPRING_TRANS_1D=lb/in; SPRING_ROTATE_1D=lb-in/rad; SPRING_TRANS_2D=kip/in^2; SPRING_TRANS_3D=kip/in^3
    // TEMPERATURE=Fahrenheit
};
```

```

virtual void setStandardEnglishUnits() = 0;
// LENGTH=m; DIMENSION=mm; FORCE=kN; FORCE_LINE=kN/m; MOMENT=kN-m; FORCE_SURFACE=kN/m^2;
// DISPLACEMENT_TRANS=mm; DISPLACEMENT_ROTATE=rad; MODULUS=kN/mm^2; WEIGHT_DENSITY=kN/m^3; STRESS=N/mm^2
// SPRING_TRANS_1D=N/mm; SPRING_ROTATE_1D=N-mm/rad; SPRING_TRANS_2D=N/mm^2; SPRING_TRANS_3D=N/bb^3
// TEMPERATURE=Celsius
virtual void setStandardMetricUnits() = 0;
// lb; in; rad
virtual void setConsistentEnglishUnits() = 0;
// N; m; rad
virtual void setConsistentMetricUnits() = 0;
virtual void getUnit(TCHAR szUnit[], int nUnit) const = 0;

virtual bool saveDocument(LPCTSTR lpszPathName) = 0;

virtual int getLastError() const = 0;
virtual void clearLastError() = 0;
virtual void setShowSolverMessageBox(bool bShow) = 0;
virtual bool getShowSolverMessageBox() const = 0;

virtual void setAbortSolutionKey(int key) = 0;
virtual int getAbortSolutionKey() const = 0;

virtual void setCurrentBarDatabase(int currentBarDatabase) = 0;
virtual int getCurrentBarDatabase() const = 0;
virtual bool getBarInfo(int& barIndex, double& barDiameter, double& barArea, const TCHAR barSizeDesignation[]) const = 0;
virtual bool getBarSizeDesignation(TCHAR barSizeDesignation[], int barIndex) const = 0;

virtual bool addRectangularSection(const TCHAR sectionLabel[], double width, double height, double coverToBarCenter, double fc,
double fy, const TCHAR barSizeDesignation[], int topBars, int leftBars, int confinement) = 0;
virtual bool addCircularSection(const TCHAR sectionLabel[], double diameter, double coverToBarCenter, double fc, double fy,
const TCHAR barSizeDesignation[], int totalBars, double startAngle, int confinement) = 0;
virtual bool addGenericSection(const TCHAR sectionLabel[], cgiColumnBlazeNamespace::cgiRcPoint* boundary, int boundaryPointCount,
cgiColumnBlazeNamespace::cgiRcPoint* opening1, int openingPointCount1,
cgiColumnBlazeNamespace::cgiRcPoint* opening2, int openingPointCount2,
cgiColumnBlazeNamespace::cgiRcPoint* opening3, int openingPointCount3,
cgiColumnBlazeNamespace::cgiLineBar* lineBars, int lineBarCount, double fc, double fy, int confinement) = 0;
virtual int generateRectangularSections(double fc, double fy, double b, double h, int startBarIndex, int endBarIndex,
int stirrupBarIndex, double clearCoverToStirrup, int minLeftOrRightBars,
int maxLeftOrRightBars, int minTopOrBottomBars, int maxTopOrBottomBars, int barLayoutSpec,
int confinement) = 0;

```

```

virtual int generateCircularSections(double fc, double fy, double diameter, int startBarIndex, int endBarIndex, int stirrupBarIndex,
                                   double clearCoverToStirrup, int minBars, int maxBars, int confinement, double startAngle = 0) = 0;
virtual bool setConcreteCode(int code) = 0;
virtual int getConcreteCode()const = 0;
virtual void setConcreteSolverOption(int angleSteps, int neutralSteps, int axialCapacitySteps, bool excludeDisplaced) = 0;
virtual void getConcreteSolverOption(int& angleSteps, int& neutralSteps, int& axialCapacitySteps, bool& excludeDisplaced)const = 0;
virtual void setMinMaxReinforcementRatios(double minReinforcementRatio = 0.01, double maxReinforcementRatio = 0.08) = 0;
// used in generating sections
virtual void getMinMaxReinforcementRatios(double& minReinforcementRatio, double& maxReinforcementRatio) const = 0;
virtual void setStrengthReductionMode(int strengthReductionMode = 0) = 0;
virtual int getStrengthReductionMode()const = 0;
virtual void setUnityRatioMethod(int method = 0) = 0;
virtual int getUnityRatioMethod()const = 0;

virtual bool solve() = 0;
// check against all available sections in the structure
virtual bool checkLoad(double& unityRatio, int& criticalSectionIndex, const cgiColumnBlazeNamespace::cgiSectionLoad& load)const = 0;
virtual bool checkLoad_singleSection(double& unityRatio, int sectionIndex, const cgiColumnBlazeNamespace::cgiSectionLoad& load)const=0;

virtual int getConcreteSections(cgiColumnBlazeNamespace::cgiConcreteSection*& sections, int& count)const = 0;
virtual int getSectionsInteractionData(cgiColumnBlazeNamespace::cgiSection_P_M_Interaction*& sections_pm_interaction, int& count)const = 0;
virtual bool getSectionsInteractionData_singleSection(cgiColumnBlazeNamespace::cgiSection_P_M_Interaction*& sections_pm_interaction, int& count,
                                                    int sectionIndex)const = 0;
virtual bool getMxMyData_singleSection(cgiColumnBlazeNamespace::cgiP_M*& mx_my, int& count, int sectionIndex, double p)const = 0;
virtual bool getControlPointDescription(TCHAR controlPointDescription[50], int controlPoint)const = 0;
virtual void clearConcreteSections() = 0;
virtual void clearConcreteSectionResult() = 0;

virtual void deleteMemoryArray(cgiColumnBlazeNamespace::cgiConcreteSection* p)const = 0;
virtual void deleteMemoryArray(cgiColumnBlazeNamespace::cgiSection_P_M_Interaction* p)const = 0;
virtual void deleteMemoryArray(cgiColumnBlazeNamespace::cgiP_M* p)const = 0;
};

CGICOLUMNBLAZE_API cgiIColumnStructure* CreateColumnStructure(LPCTSTR szRebarDatabaseFile = nullptr);
CGICOLUMNBLAZE_API void DeleteColumnStructure(cgiIColumnStructure* pStructure);

```

The following lists a few common enums

```
// unit
enum { LENGTH, DIMENSION,
FORCE, FORCE_LINE, MOMENT, MOMENT_LINE, FORCE_SURFACE,
DISPLACEMENT_TRANS, DISPLACEMENT_ROTATE,
TEMPERATURE,
MODULUS, WEIGHT_DENSITY, REINFORCEMENT_AREA, STRESS,
SPRING_TRANS_1D, SPRING_ROTATE_1D, SPRING_TRANS_2D, SPRING_TRANS_3D,
UNIT_END};

enum {
kErrorNone, kInvalidInput, kSolverError, kAbnormalSolverTermination, kFileAccessError, kLicenseError,
kNoResultAvailable, kInvalidSectionIndex, kInvalidRebar, kInvalidConcreteDesignCode, kInvalidAxialLoadForMxMy,
kUnknownError
};

enum { kReduction_Auto, kReduction_Always_1 };
enum { kConfine_Tied, kConfine_Spiral };
enum { kACI_1999, kACI_2002, kACI_2005, kACI_2008, kACI_2011, kACI_2014, kACI_2019, kCode_End };
enum { kBar_Designation_Max_Size = 10 };
enum { kRectangularC, kCircular, kTee, kInversedL, kI, kSemiCircular, kPolygon, kRectangularB, kGeneric };
enum {
CONTROL_FY, CONTROL_05FY, CONTROL_025FY, CONTROL_0FY, CONTROL_FY_, CONTROL_005,
CONTROL_FLEXURE, CONTROL_PMAX, MAX_CONTROL_POINTS
};
enum { BAR_LAYOUT_ALL_SIDES, BAR_LAYOUT_MAJOR_SIDES, BAR_LAYOUT_MINOR_SIDES, BAR_LAYOUT_EQUAL_SIDES };

enum { kUnityRatio_Max_Axial_Moment, kUnityRatio_Moment_Only, kUnityRatio_End };
```

The following lists a few common constants

```
const int LABEL_SIZE = 128;

const int RC_LABEL_SIZE = 128; // must be consistent with LABEL_SIZE defined in cgiDefines.h
```

The following lists a few result data structures

```
struct CGICOLUMNBLAZE_API cgiRcPoint
{
    double xy[2];
};

struct CGICOLUMNBLAZE_API cgiLineBar
{
    cgiRcPoint ptStart;
    cgiRcPoint ptEnd;
    TCHAR barDesignation[kBar_Designation_Max_Size];
    int nCount;
};

struct CGICOLUMNBLAZE_API cgiConcreteSection
{
    TCHAR szLabel[RC_LABEL_SIZE];
    int iType;
    int iConfinement;
    double b;
    double h;
    double cover;
    double start_angle;    // rebar start angle for circular section
    double fc;
    double fy;
    TCHAR barDesignation[kBar_Designation_Max_Size];
    int nTopBars;
    int nBottomBars;
    int nLeftBars;
    int nRightBars;

    cgiRcPoint* vPt;
    int nPtCount;
    cgiRcPoint* vOpeningPt1;
    int nOpeningPtCount1;
    cgiRcPoint* vOpeningPt2;
    int nOpeningPtCount2;
    cgiRcPoint* vOpeningPt3;
    int nOpeningPtCount3;
    cgiLineBar* vLineBar;
    int nLineBarCount;

    double fTotalAs;
    double fSectionA;
    cgiRcPoint ptCentroid;
    double fInertiaIx;
    double fInertiaIy;
};
```

```

    double fInertiaIxy;
    double fBarClearSpacing;
    double fCapacityRatio;
    int iCriticalLoadIndex;
    double fIsx;
    double fIsy;
};

struct CGICOLUMNBLAZE_API cgiP_M
{
    double fC;      // inch
    double fStrain;
    double fPhi;
    double fP;      // kip
    double fMx;     // kip-ft
    double fMy;
};

// P_M interaction at an angle
struct CGICOLUMNBLAZE_API cgiP_M_Interaction
{
    double fAngle;
    cgiP_M pm_control[MAX_CONTROL_POINTS]; // corresponds to steel stress Fy, 0.5Fy, 0.25Fy, 0Fy, -Fy and a strain of 0.005
    cgiP_M* vPM;
    int nPM_Count;
};

struct CGICOLUMNBLAZE_API cgiSection_P_M_Interaction
{
    // do not allow copy and assign due to memory deallocation issue
    cgiSection_P_M_Interaction& operator= (const cgiSection_P_M_Interaction&) = delete;
    cgiSection_P_M_Interaction(const cgiSection_P_M_Interaction&) = delete;

    cgiP_M_Interaction* vPM_Int;
    int nPM_Int_Count;
    cgiP_M_Interaction* vPM_Int_Display;
    int nPM_Int_Display_Count;
    cgiP_M_Interaction* vPM_Int_Display_Control;
    int nPM_Int_Display_Control_Count;
};

struct CGICOLUMNBLAZE_API cgiSectionLoad
{
    double fP;
    double fMx;
    double fMy;
};

bool CGICOLUMNBLAZE_API is_equal(double val1, double val2, double epsilon = gEpsilon);

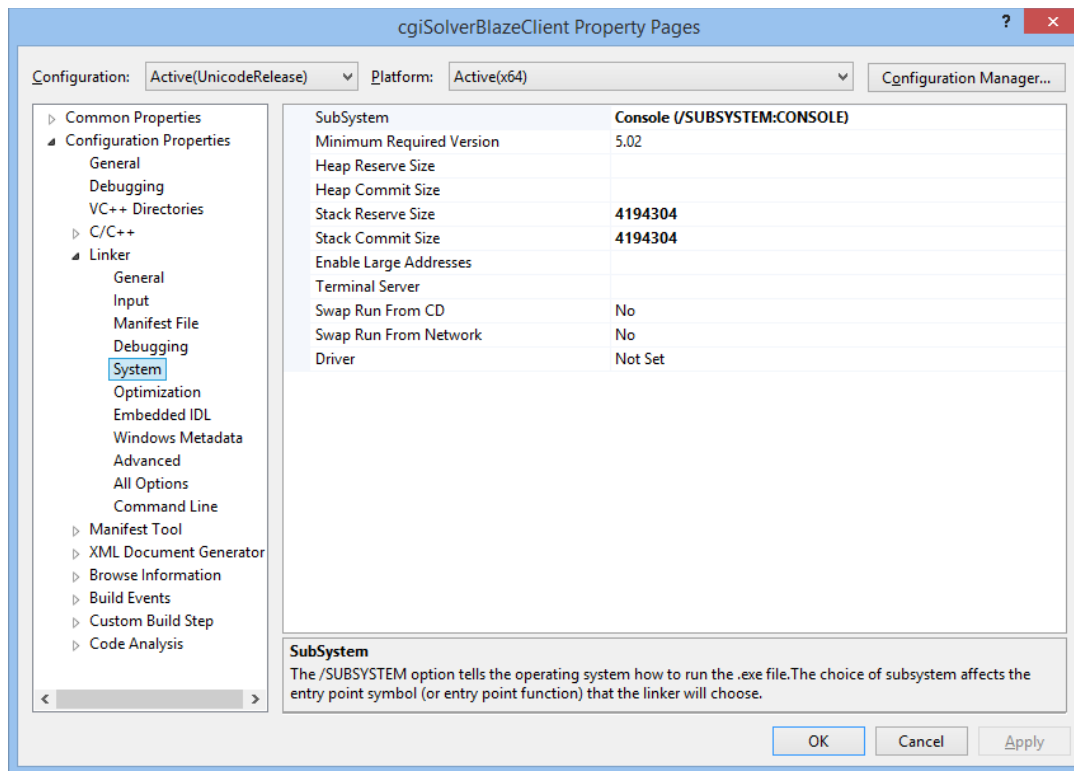
```

```

struct CGICOLUMNBLAZE_API cgiPoint
{
    double xyz[3];
};

```

Four versions of `cgiColumnBlaze.lib` and `cgiColumnBlaze.dll` are provided for your configuration needs: Unicode or Multi-Bytes, 86x or 64x CPUs. You should link your projects to the correct version of the lib file. Make sure you also copy the correct version of the DLLs (`cgiColumnBlaze.dll`) to your executable directory. If your project is configured for 64-bit CPU, you may need to increase the stack reserve size and stack commit size from the default 1 MB to something larger (say 4 MB) as shown below.



## Example

In this example, we are going to analyze a rectangular, circular, and generic sections, one at a time.

Section 1 (also see Example001 in cColumn)

[Example 7.11 and Example 7.12, “Design Guide on the ACI 318 Building Code Requirements for Structural Concrete” 1st Edition, Concrete Reinforcing Steel Institute, 2020]

Uniaxial Analysis, Square, ACI 318-19

*Rectangular section 18x18 inch, concrete cover 2.44 inch, 4#9,  $f_c' = 4$  ksi,  $f_y = 60$  ksi and  $f_y = 100$  ksi, tied confinement.*

Section 2 (also see Example002 in cColumn)

[Example 7.13, “Design Guide on the ACI 318 Building Code Requirements for Structural Concrete” 1st Edition, Concrete Reinforcing Steel Institute, 2020]

Uniaxial Analysis, Square, ACI 318-19

*Circular section diameter 20 inch, concrete cover 2.44 inch, 4#9,  $f_c' = 4$  ksi,  $f_y = 60$  ksi, tied confinement*

Section 3 (also see Example008 in cColumn)

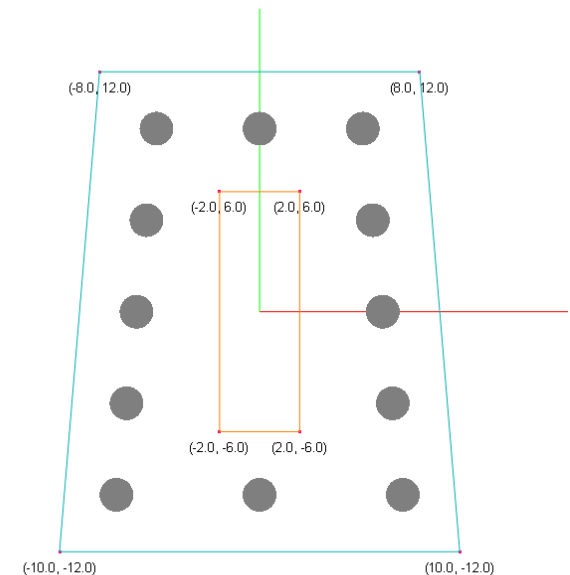
Investigate the trapezoid section with opening as shown here, according to ACI 318-2019. The section top and bottom widths are 16 and 20 inches respectively. The section height is 24 inches. The opening is 4 x 12 inches rectangle. The figure below shows the vertex coordinates of the section.

Find the section capacity ( $\phi P_n$  and  $\phi M_n$ ) corresponding to 50% tension at the bottom steel bars.

Material properties:  $f_c' = 6$  ksi,  $f_y = 60$  ksi

Concrete cover to the center of main bars = 2.8465 inches.

Reinforcement bars: 12 #14 as shown below.



**The following is the complete C++ source code to set up the Example. We will examine the code in detail in a moment.**

```
#include "_cgiIColumnStructure.h"
#include <iostream>
#include <iomanip>
#include <vector>
using namespace std;
using namespace cgiColumnBlazeNamespace;

#ifdef _UNICODE
#define COUT wcout
#else
#define COUT cout
#endif

static void ListMsg(LPCTSTR sz0, LPCTSTR sz1)
{
    COUT << sz0 << "\t" << sz1 << endl;
}

static void StatusMsg(LPCTSTR sz)
{
    COUT << "STATUS _____ " << sz << endl;
}

void displayResults(cgiIColumnStructure* pStructure, bool useSingleSection)
{
    COUT << "----- " << endl;
    if (useSingleSection) {
        COUT << "----- use single section api ----- " << endl;
    }
    else {
        COUT << "----- not use single section api ----- " << endl;
    }

    COUT << "----- " << endl;

    int angleSteps, neutralSteps, axialCapacitySteps;
    bool excludeDisplaced;
    pStructure->getConcreteSolverOption(angleSteps, neutralSteps, axialCapacitySteps, excludeDisplaced);
    COUT << "angleStepse " << angleSteps << endl;
    COUT << "neutralSteps " << neutralSteps << endl;
    COUT << "axialCapacitySteps " << axialCapacitySteps << endl;
    COUT << "excludeDisplaced " << excludeDisplaced << endl;
    COUT << "----- " << endl;

    if (true)
    {
        // test getConcreteSections()
        cgiColumnBlazeNamespace::cgiConcreteSection* sections = nullptr;
    }
}
```

```

int count;
count = 0;
pStructure->getConcreteSections(sections, count);

for (int i = 0; i < count; i++)
{
    const auto& section = sections[i];
    COUT << section.szLabel << endl;
    COUT << "section type " << section.iType << endl;
    COUT << "b " << section.b << endl;
    COUT << "h " << section.h << endl;
    COUT << "nPtCount " << section.nPtCount << endl;
    COUT << "nOpeningPtCount1 " << section.nOpeningPtCount1 << endl;
    COUT << "nOpeningPtCount2 " << section.nOpeningPtCount2 << endl;
    COUT << "nOpeningPtCount3 " << section.nOpeningPtCount3 << endl;
    COUT << "nLineBarCount " << section.nLineBarCount << endl;
    COUT << "fSectionA " << section.fSectionA << endl;
    COUT << "fTotalAs " << section.fTotalAs << endl;
    COUT << "fInertiaIx " << section.fInertiaIx << endl;
    COUT << "fInertiaIy " << section.fInertiaIy << endl;
    COUT << "fInertiaIxy " << section.fInertiaIxy << endl;
    COUT << "fBarClearSpacing " << section.fBarClearSpacing << endl;
    COUT << "fIsx " << section.fIsx << endl;
    COUT << "fIsy " << section.fIsy << endl;
}
pStructure->deleteMemoryArray(sections);
}

if (useSingleSection)
{
    // test getSectionsInteractionData_singleSection()
    cgiColumnBlazeNamespace::cgiSection_P_M_Interaction* sections_pm_interaction = nullptr;
    int count;
    int sectionIndex = 0;
    pStructure->getSectionsInteractionData_singleSection(sections_pm_interaction, count, sectionIndex);
    assert(count == 1);
    const auto& section_pm_interaction = sections_pm_interaction[0];

    for (int j = 0; j < section_pm_interaction.nPM_Int_Display_Control_Count; j++)
    {
        const auto& pm_interaction = section_pm_interaction.vPM_Int_Display_Control[j];

        COUT << "-----" << endl;
        COUT << "-----" << endl;
        COUT << "angle = " << pm_interaction.fAngle << endl;

        COUT << "control points " << endl;
        for (int k = 0; k < MAX_CONTROL_POINTS; k++)
        {
            const auto& pm = pm_interaction.pm_control[k];
            TCHAR controlPointDescription[50];
            pStructure->getControlPointDescription(controlPointDescription, k);

```

```

        COUT << controlPointDescription << ", fC=" << pm.fC << ", fP=" << pm.fP << ", fMx=" << pm.fMx << ", fMy=" << pm.fMy << ", fStrain=" <<
            pm.fStrain << ", fPhi=" << pm.fPhi << endl;
    }
    COUT << endl;

    COUT << "P-M interaction points " << endl;
    for (int k = 0; k < pm_interaction.nPM_Count; k++)
    {
        const auto& pm = pm_interaction.vPM[pm_interaction.nPM_Count - 1 - k];
        COUT << "k=" << k + 1 << ", fC=" << pm.fC << ", fP=" << pm.fP << ", fMx=" << pm.fMx << ", fMy=" << pm.fMy << ", fStrain=" << pm.fStrain <<
            ", fPhi=" << pm.fPhi << endl;
    }
    COUT << "-----" << endl;
    COUT << "-----" << endl;
    break; // first angle only
}
pStructure->deleteMemoryArray(sections_pm_interaction);
}
else
{
    // test getSectionsInteractionData()
    int count = 0;
    cgiColumnBlazeNamespace::cgiSection_P_M_Interaction* sections_pm_interaction = nullptr;
    pStructure->getSectionsInteractionData(sections_pm_interaction, count);
    for (int i = 0; i < count; i++)
    {
        const auto& section_pm_interaction = sections_pm_interaction[i];

        for (int j = 0; j < section_pm_interaction.nPM_Int_Display_Control_Count; j++)
        {
            const auto& pm_interaction = section_pm_interaction.vPM_Int_Display_Control[j];

            COUT << "-----" << endl;
            COUT << "-----" << endl;
            COUT << "angle = " << pm_interaction.fAngle << endl;

            for (int k = pm_interaction.nPM_Count - 1; k >= 0; k--)
            {
                const auto& pm = pm_interaction.vPM[k];
                COUT << "k=" << k << ", fC=" << pm.fC << ", fP=" << pm.fP << ", fMx=" << pm.fMx << ", fMy=" << pm.fMy << ", fStrain=" << pm.fStrain <<
                    ", fPhi=" << pm.fPhi << endl;
            }
            COUT << "-----" << endl;
            COUT << "-----" << endl;
            break;
        }
        break;
    }
    pStructure->deleteMemoryArray(sections_pm_interaction);
}
}
if (true)

```

```

{ // test getSectionsInteractionData_singleSection()
  cgiColumnBlazeNamespace::cgiP_M* mx_my = nullptr;
  int count = 0;
  int sectionIndex = 0;
  double p = 600;
  bool success = pStructure->getMxMyData_singleSection(mx_my, count, sectionIndex, p);
  auto lastError = pStructure->getLastError();
  COUT << "-----" << endl;
  COUT << "-----" << endl;
  for (int j = 0; j < count; j++)
  {
    const auto& mxy = mx_my[j];
    COUT << "fP=" << mxy.fP << ", fMx=" << mxy.fMx << ", fMy=" << mxy.fMy << ", neutral depth=" << mxy.fC << ", max tensile strain=" << mxy.fStrain
      << ", angle=" << j * 360. / count << endl;
  }
  COUT << "-----" << endl;
  COUT << "-----" << endl;
  pStructure->deleteMemoryArray(mx_my);
}
}

void verify_example_singleSection()
{
  COUT << _T("-----") << endl;
  COUT << _T("----- Verifying Single section -----") << endl;
  COUT << _T("-----") << endl;

  // create a structural model, pass a custom rebar database file if needed here
  cgiIColumnStructure* pStructure = CreateColumnStructure(nullptr);

  // message functions, can be set null in which case no messages will be printed during solution
  pStructure->setListMessageFunction(ListMsg);
  pStructure->setStatusMessageFunction(StatusMsg);

  // LENGTH=ft; DIMENSION=in; FORCE=kip; FORCE_LINE=kip/ft; MOMENT=kip-ft; FORCE_SURFACE=lb/ft^2;
  // DISPLACEMENT_TRANS=in; DISPLACEMENT_ROTATE=rad; MODULUS=kip/in^2; WEIGHT_DENSITY=lb/ft^3; STRESS=lb/in^2
  // SPRING_TRANS_1D=lb/in; SPRING_ROTATE_1D=lb-in/rad; SPRING_TRANS_2D=kip/in^2; SPRING_TRANS_3D=kip/in^3
  // TEMPERATURE=Fahrenheit
  pStructure->setStandardEnglishUnits();

  {
    pStructure->clearConcreteSections();
    pStructure->addRectangularSection(_T("section1"), 18, 18, 2.44, 4.0, 60.0, _T("#9"), 2, 2, kConfine_Tied);
    pStructure->setStrengthReductionMode(kReduction_Auto);
    pStructure->setConcreteSolverOption(16, 250, 20, true);

    bool success = pStructure->solve();
    if (!success)
    {
      COUT << "error occured: " << pStructure->getLastError() << endl;
    }
    //pStructure->saveDocument(_T("c:\\temp\\test.rcs"));
  }
}

```

```

displayResults(pStructure, true);
displayResults(pStructure, false);
}

{
pStructure->clearConcreteSections();
int confinement = kConfine_Spiral; // 0=tied, 1=spiral
pStructure->addCircularSection(_T("section2"), 20, 2.44, 4.0, 60.0, _T("#9"), 4, 0, confinement);
pStructure->setStrengthReductionMode(kReduction_Auto);
pStructure->setConcreteSolverOption(16, 250, 20, true);

bool success = pStructure->solve();
if (!success)
{
    COUT << "error occurred: " << pStructure->getLastError() << endl;
}

displayResults(pStructure, true);
displayResults(pStructure, false);
}

{
pStructure->clearConcreteSections();
cgiRcPoint boundary[4] = { {-10, -12}, {10, -12}, {8, 12}, {-8, 12} };
cgiRcPoint opening1[4] = { {-2, -6}, {2, -6}, {2, 6}, {-2, 6} };
cgiLineBar lineBar1(cgiRcPoint(-7.1535, -9.1535), cgiRcPoint(-5.1535, 9.1535), _T("#14"), 5);
cgiLineBar lineBar2(cgiRcPoint(7.1535, -9.1535), cgiRcPoint(5.1535, 9.1535), _T("#14"), 5);
cgiLineBar lineBar3(cgiRcPoint(0, -9.1535), cgiRcPoint(0, 9.1535), _T("#14"), 2);
cgiLineBar lineBars[3] = { lineBar1, lineBar2, lineBar3 };
pStructure->addGenericSection(_T("section3"), boundary, 4, opening1, 4, nullptr, 0, nullptr, 0, lineBars, 3, 6.0, 60.0, kConfine_Tied);
pStructure->setStrengthReductionMode(kReduction_Auto);
pStructure->setConcreteSolverOption(128, 250, 250, true);

bool success = pStructure->solve();
if (!success)
{
    COUT << "error occurred: " << pStructure->getLastError() << endl;
}

displayResults(pStructure, true);
displayResults(pStructure, false);
}
}

```

## The following is the detailed explanations for each steps in Example Model

To start, you create a `cgiIColumnStructure` interface object that represents the one and only structural model like this:

```
cgiIColumnStructure* pStructure = CreateColumnStructure(nullptr);
```

You can supply two optional callback functions for the solver to notify your application the progress of solution:

```
typedef void (* fnLISTMSG)(LPCTSTR sz0, LPCTSTR sz1);
typedef void (* fnSTATUSMSG)(LPCTSTR sz);

static void ListMsg(LPCTSTR sz0, LPCTSTR sz1)
{
    COUT << sz0 << "\t" << sz1 << endl;
}

static void StatusMsg(LPCTSTR sz)
{
    COUT << "STATUS _____ " << sz << endl;
}

pStructure->setListMessageFunction(ListMsg);
pStructure->setStatusMessageFunction(StatusMsg);
```

Four unit systems are available in the library. They are Standard English unit, Standard Metric unit, Consistent English unit and Consistent Metric unit. For Standard English unit system, the following units are used for length, section dimension, force, moment etc.

```
// LENGTH=ft; DIMENSION=in;
// FORCE=kip; FORCE_LINE=kip/ft; MOMENT=kip-ft; FORCE_SURFACE=lb/ft^2;
// DISPLACEMENT_TRANS=in; DISPLACEMENT_ROTATE=rad; MODULUS=kip/in^2; WEIGHT_DENSITY=lb/ft^3; STRESS=lb/in^2
// SPRING_TRANS_1D=lb/in; SPRING_ROTATE_1D=lb-in/rad; SPRING_TRANS_2D=kip/in^2; SPRING_TRANS_3D=kip/in^3
// TEMPERATURE=Fahrenheit
pStructure->setStandardEnglishUnits();
```

For standard Metric unit system, the following units are used for length, section dimension, force, moment etc.

```
// LENGTH=m; DIMENSION=mm;
// FORCE=kN; FORCE_LINE=kN/m; MOMENT=kN-m; FORCE_SURFACE=kN/m^2;
// DISPLACEMENT_TRANS=mm; DISPLACEMENT_ROTATE=rad; MODULUS=kN/mm^2; WEIGHT_DENSITY=kN/m^3; STRESS=N/mm^2
// SPRING_TRANS_1D=N/mm; SPRING_ROTATE_1D=N-mm/rad; SPRING_TRANS_2D=N/mm^2; SPRING_TRANS_3D=N/mm^3
// TEMPERATURE=Celsius
```

```
pStructure->setStandardMetricUnits ();
```

**By default, message boxes will be displayed if warnings or errors are encountered during the solution.** You can suppress the display of the message boxes by calling `setShowSolverMessageBox(false)`. It is important to check the errors immediately after calling the static or frequency analysis solver by `getLastError()`. The following are the error codes that ColumnBlaze currently may emit.

```
enum {
    kErrorNone, kInvalidInput, kSolverError, kAbnormalSolverTermination, kFileAccessError, kLicenseError,
    kNoResultAvailable, kInvalidSectionIndex, kInvalidRebar, kInvalidConcreteDesignCode, kInvalidAxialLoadForMxMy,
    kUnknownError
};
```

The last error code is cleared (set to `kErrorNone`) at the beginning of each solution.

Since we are solving one section at a time, we clear the existing sections before adding a new one.

```
pStructure->clearConcreteSections();
```

A new section is added by `addRectangularSection()`, `addCircularSection()`, or `addGenericSection()`. The section label must be less than 127 characters long. A new section will not be added if a section with the label already exists in the model. You can check the return value of `addRectangularSection()`, `addCircularSection()`, and `addGenericSection()` to see if a section is added successfully.

```
pStructure->addRectangularSection(_T("section1"), 18, 18, 2.44, 4.0, 60.0, _T("#9"), 2, 2, kConfine_Tied);

int confinement = kConfine_Spiral; // 0=tied, 1=spiral
pStructure->addCircularSection(_T("section2"), 20, 2.44, 4.0, 60.0, _T("#9"), 4, 0, confinement);

cgiRcPoint boundary[4] = { {-10, -12}, {10, -12}, {8, 12}, {-8, 12} };
cgiRcPoint opening1[4] = { {-2, -6}, {2, -6}, {2, 6}, {-2, 6} };
cgiLineBar lineBar1(cgiRcPoint(-7.1535, -9.1535), cgiRcPoint(-5.1535, 9.1535), _T("#14"), 5);
cgiLineBar lineBar2(cgiRcPoint(7.1535, -9.1535), cgiRcPoint(5.1535, 9.1535), _T("#14"), 5);
cgiLineBar lineBar3(cgiRcPoint(0, -9.1535), cgiRcPoint(0, 9.1535), _T("#14"), 2);
cgiLineBar lineBars[3] = { lineBar1, lineBar2, lineBar3 };
pStructure->addGenericSection(_T("section3"), boundary, 4, opening1, 4, nullptr, 0, nullptr, 0, lineBars,
    3, 6.0, 60.0, kConfine_Tied);
```

There are two modes in strength reduction: `kReduction_Auto` and `kReduction_Always_1`. You should always use `kReduction_Auto` where strength reduction factor  $\phi$  is automatically applied. `kReduction_Always_1` option is provided for academic reasons.

```
pStructure->setStrengthReductionMode(kReduction_Auto);
```

For accuracy and solver solution time reasons, it is important to set up proper solver options by calling `setConcreteSolverOption()`.

```
pStructure->setConcreteSolverOption(16, 250, 20, true);
```

The angle steps (1st parameter) must be of multiple of 4. For uniaxial solution, the value of 4 is enough. For biaxial solution, the value must be 16 or larger. The neutral steps (2nd parameter) should be large enough to capture capacities at the different neutral axis positions (e.g.: 250). The axial capacity steps (3<sup>rd</sup> parameter) is for result data presentation and display. It should be larger than 20. Lastly, you should set the exclude the concrete displaced by the steel reinforcement (4<sup>th</sup> parameter) to be true. The option to set it to false is provided for academic reasons.

By default, the solution will abort by pressing ESC key during the solution. You can customize the key by calling `setAbortSolutionKey()`.

To solve the model, simply call

```
bool success = pStructure->solve();
```

You may optionally save the input to a file that can be opened by cColumn software. This can be useful if you would like to view the result data in spreadsheets or graphical diagrams.

```
pStructure->saveDocument(_T("c:\\temp\\test.rcs"));
```

The sections including some calculated properties can be retrieved by `getConcreteSections()`. It is important to call `deleteMemoryArray()` function to free memories allocated by the solver instead of use C++ delete operator directly.

```
cgiColumnBlazeNamespace::cgiConcreteSection* sections = nullptr;
int count;
count = 0;
pStructure->getConcreteSections(sections, count);

for (int i = 0; i < count; i++)
{
    const auto& section = sections[i];
    COUT << section.szLabel << endl;
    COUT << "section type " << section.iType << endl;
    COUT << "b " << section.b << endl;
    COUT << "h " << section.h << endl;
    COUT << "nPtCount " << section.nPtCount << endl;
}
```

```

    COUT << "nOpeningPtCount1 " << section.nOpeningPtCount1 << endl;
    COUT << "nOpeningPtCount2 " << section.nOpeningPtCount2 << endl;
    COUT << "nOpeningPtCount3 " << section.nOpeningPtCount3 << endl;
    COUT << "nLineBarCount " << section.nLineBarCount << endl;
    COUT << "fSectionA " << section.fSectionA << endl;
    COUT << "fTotalAs " << section.fTotalAs << endl;
    COUT << "fInertiaIx " << section.fInertiaIx << endl;
    COUT << "fInertiaIy " << section.fInertiaIy << endl;
    COUT << "fInertiaIxy " << section.fInertiaIxy << endl;
    COUT << "fBarClearSpacing " << section.fBarClearSpacing << endl;
    COUT << "fIsx " << section.fIsx << endl;
    COUT << "fIsy " << section.fIsy << endl;
}
pStructure->deleteMemoryArray(sections);

```

The section P-M interaction results can be retrieved easily by calling either `getSectionsInteractionData_singleSection()` function for single section or `getSectionsInteractionData()` function for all sections as illustrated in the following. The section Mx-My result data at a given axial load can be retrieved by calling `getSectionsInteractionData_singleSection()`. You need to delete memory allocated by the solver by calling the interface function `deleteMemoryArray()` instead of calling C++ delete operator.

```

// use getSectionsInteractionData_singleSection()
cgiColumnBlazeNamespace::cgiSection_P_M_Interaction* sections_pm_interaction = nullptr;
int count;
int sectionIndex = 0;
pStructure->getSectionsInteractionData_singleSection(sections_pm_interaction, count, sectionIndex);
assert(count == 1);
const auto& section_pm_interaction = sections_pm_interaction[0];

for (int j = 0; j < section_pm_interaction.nPM_Int_Display_Control_Count; j++)
{
    const auto& pm_interaction = section_pm_interaction.vPM_Int_Display_Control[j];

    COUT << "angle = " << pm_interaction.fAngle << endl;

    COUT << "control points " << endl;
    for (int k = 0; k < MAX_CONTROL_POINTS; k++)
    {
        const auto& pm = pm_interaction.pm_control[k];
        TCHAR controlPointDescription[50];
        pStructure->getControlPointDescription(controlPointDescription, k);
        COUT << controlPointDescription << ", fC=" << pm.fC << ", fP=" << pm.fP << ", fMx=" << pm.fMx << ", fMy="
            << pm.fMy << ", fStrain=" << pm.fStrain << ", fPhi=" << pm.fPhi << endl;
    }
}

```

```

}
COUT << endl;

COUT << "P-M interaction points " << endl;
for (int k = 0; k < pm_interaction.nPM_Count; k++)
{
    const auto& pm = pm_interaction.vPM[pm_interaction.nPM_Count - 1 - k];
    COUT << "k=" << k + 1 << ", fC=" << pm.fC << ", fP=" << pm.fP << ", fMx=" << pm.fMx << ", fMy=" << pm.fMy
        << ", fStrain=" << pm.fStrain << ", fPhi=" << pm.fPhi << endl;
}
break; // first angle only
}
pStructure->deleteMemoryArray(sections_pm_interaction);

// use getSectionsInteractionData()
int count = 0;
cgiColumnBlazeNamespace::cgiSection_P_M_Interaction* sections_pm_interaction = nullptr;
pStructure->getSectionsInteractionData(sections_pm_interaction, count);
for (int i = 0; i < count; i++)
{
    const auto& section_pm_interaction = sections_pm_interaction[i];

    for (int j = 0; j < section_pm_interaction.nPM_Int_Display_Control_Count; j++)
    {
        const auto& pm_interaction = section_pm_interaction.vPM_Int_Display_Control[j];

        COUT << "angle = " << pm_interaction.fAngle << endl;

        for (int k = pm_interaction.nPM_Count - 1; k >= 0; k--)
        {
            const auto& pm = pm_interaction.vPM[k];
            COUT << "k=" << k << ", fC=" << pm.fC << ", fP=" << pm.fP << ", fMx=" << pm.fMx << ", fMy=" << pm.fMy
                << ", fStrain=" << pm.fStrain << ", fPhi=" << pm.fPhi << endl;
        }
        break; // first angle only
    }
    break; // first section only
}

pStructure->deleteMemoryArray(sections_pm_interaction);

```

```

// test getSectionsInteractionData_singleSection()
cgiColumnBlazeNamespace::cgiP_M* mx_my = nullptr;
int count = 0;
int sectionIndex = 0;
double p = 600;
bool success = pStructure->getMxMyData_singleSection(mx_my, count, sectionIndex, p);
auto lastError = pStructure->getLastError();
for (int j = 0; j < count; j++)
{
    const auto& mxy = mx_my[j];
    COUT << "fP=" << mxy.fP << ", fMx=" << mxy.fMx << ", fMy=" << mxy.fMy << ", neutral depth=" << mxy.fC << ",
        max tensile strain=" << mxy.fStrain << ", angle=" << j * 360. / count << endl;
}
pStructure->deleteMemoryArray(mx_my);

```

ColumnBlaze uses unity ratio to check one or more loads against one or more sections as shown in the following:

```

pStructure->setUnityRatioMethod(cgiColumnBlazeNamespace::kUnityRatio_Max_Axial_Moment);
cgiSectionLoad load = cgiSectionLoad(100, 20, 10);
double unityRatio = 999.0;
int sectionIndex = 0;
bool ok = pStructure->checkLoad_singleSection(unityRatio, sectionIndex, load);

double criticalUnityRatio = 999.0;
int criticalSectionIndex = 0;
pStructure->checkLoad(criticalUnityRatio, criticalSectionIndex, load);

```

There are two methods to calculate unity ratio: `kUnityRatio_Max_Axial_Moment` and `kUnityRatio_Moment_Only` (not recommended). A unity ratio of less than 1.0 means the section(s) is adequate against a given load. Conversely, a unity ratio of greater than 1.0 means the section(s) is not adequate against a given load. Use `checkLoad_singleSection()` to check a load against a single section. Use `checkLoad()` to check a load against all sections.

## .NET Interface

The .NET Framework interface is a class library assembly `cgiColumnBlazeCli.dll`. It is written in C++/CLI and forwards calls to a native Windows DLL called `cgiColumnBlaze.dll`. The actual interface functions are contained in a single class called `cgiColumnBlazeCli` as listed in the following. The .NET Core interface is a class library assembly `cgiColumnBlazeCoreCli.dll`. The interface functions are identical to those in .NET Framework interface.

Different versions of `cgiColumnBlazeCli.dll` are provided to work with .NET Framework 4.0-4.8 on x86 and x64 CPUs. Different versions of `cgiColumnBlazeCoreCli.dll` are provided to work with .NET 5.0-10.0 on x86 and x64 CPUs. Make sure your project has reference to the correct version of `cgiColumnBlazeCli.dll` or `cgiColumnBlazeCoreCli.dll`. If your project is configured to build for both x86 and x64 CPUs, make sure to set “Copy Local” to false for the DLL reference and manually copy the correct version of `cgiColumnBlazeCli.dll` or `cgiColumnBlazeCoreCli.dll` to the executable directory. Also make sure a copy of the native dependent DLLs (`cgiColumnBlaze.dll`) are placed in the executable directory. For .NET Core, `ijwhost.dll` is a shim for finding and loading the runtime. All C++/CLI libraries are linked to this shim, such that `ijwhost.dll` is found/loaded when the C++/CLI library is loaded. Therefore, make sure a copy of `ijwhost.dll` (e.g.: `C:\Program Files\dotnet\packs\Microsoft.NETCore.App.Host.win-x64\7.0.0\runtimes\win-x64\native\Ijwhost.dll`) is placed in the executable directory as well.

`cgiColumnBlazeCli` is the one and only interface to set input, perform analysis and retrieve output. The best way to learn how you use these data structures and interfaces is to study the examples included in the C# console application project `cgiColumnBlazeTestCSharp`.

The following lists all the interface functions exposed by `cgiColumnBlazeClass` in `cgiColumnBlazeCli` namespace for .NET Framework and `cgiColumnBlazeCoreCli` namespace for .NET Core:

```
namespace cgiColumnBlazeCli
{
    public class cgiColumnBlazeClass : IDisposable
    {
        public void setListMessageFunction(cgiColumnBlazeClass.ListMessageDelegate fnListMsg);
        public void setStatusMessageFunction(cgiColumnBlazeClass.StatusMessageDelegate fnStatusMsg);

        public bool createStructure(string rebarDatabaseFile);
        public void getExePath(ref StringBuilder exePath);
        public void setProjPath(string projPath);
        public void getProjPath(ref StringBuilder projPath);
        public void setModelName(string modelName);
        public void getModelName(ref StringBuilder modelName);
        public void setDesignCompany(string designCompany);
        public void getDesignCompany(ref StringBuilder designCompany);
        public void setEngineer(string engineer);
    }
}
```

```

public void getEngineer(ref StringBuilder engineer);
public void setNotes(string notes);
public void getNotes(ref StringBuilder notes);
public void setStandardEnglishUnits();
public void setStandardMetricUnits();
public void setConsistentEnglishUnits();
public void setConsistentMetricUnits();
public void getUnit(ref StringBuilder szUnit, int nUnit);
public bool saveDocument(string pathName);
public cgiErrorEnum getLastError();
public void clearLastError();
public void setShowSolverMessageBox(bool bShow);
public bool getShowSolverMessageBox();

public void setAbortSolutionKey(int key);
public int getAbortSolutionKey();
public void setCurrentBarDatabase(int currentBarDatabase);
public int getCurrentBarDatabase();
public bool getBarInfo(ref int barIndex, ref double barDiameter, ref double barArea, string barSizeDesignation);
public bool getBarSizeDesignation(ref StringBuilder barSizeDesignation, int barIndex);

public bool addRectangularSection(string sectionLabel, double width, double height, double coverToBarCenter, double fc, double fy,
    string barSizeDesignation, int topBars, int leftBars, cgiConfinementEnum confinement);
public bool addCircularSection(string sectionLabel, double diameter, double coverToBarCenter, double fc, double fy, string barSizeDesignation,
    int totalBars, double startAngle, cgiConfinementEnum confinement);
public bool addGenericSection(string sectionLabel, List<cgiRcPointCli> boundary, List<cgiRcPointCli> opening1, List<cgiRcPointCli> opening2,
    List<cgiRcPointCli> opening3, List<cgiLineBarCli> lineBars, double fc, double fy, cgiConfinementEnum confinement);
public int generateRectangularSections(double fc, double fy, double b, double h, int startBarIndex, int endBarIndex, int stirrupBarIndex,
    double clearCoverToStirrup, int minLeftOrRightBars, int maxLeftOrRightBars, int minTopOrBottomBars,
    int maxTopOrBottomBars, cgiBarLayoutModeEnum barLayoutSpec, cgiConfinementEnum confinement);
public int generateCircularSections(double fc, double fy, double diameter, int startBarIndex, int endBarIndex, int stirrupBarIndex,
    double clearCoverToStirrup, int minBars, int maxBars, cgiConfinementEnum confinement, double startAngle);

public bool setConcreteCode(cgiConcreteCodeEnum code);
public cgiConcreteCodeEnum getConcreteCode();
public void setConcreteSolverOption(int angleSteps, int neutralSteps, int axialCapacitySteps, bool excludeDisplaced);
public void getConcreteSolverOption(ref int angleSteps, ref int neutralSteps, ref int axialCapacitySteps, ref bool excludeDisplaced);
public void setMinMaxReinforcementRatios(double minReinforcementRatio, double maxReinforcementRatio);
public void getMinMaxReinforcementRatios(ref double minReinforcementRatio, ref double maxReinforcementRatio);
public void setStrengthReductionMode(cgiStrengthReductionModeEnum strengthReductionMode);
public cgiStrengthReductionModeEnum getStrengthReductionMode();
public void setUnityRatioMethod(cgiUnityRatioMethodEnum method);
public cgiUnityRatioMethodEnum getUnityRatioMethod();

public bool solve();
public bool checkLoad(ref double unityRatio, ref int criticalSectionIndex, cgiSectionLoadCli load);
public bool checkLoad_singleSection(ref double unityRatio, int sectionIndex, cgiSectionLoadCli load);
public int getConcreteSections(ref List<cgiConcreteSectionCli> sections);
public int getSectionsInteractionData(ref List<cgiSection_P_M_InteractionCli> sections_pm_interaction);

```

```

public bool getSectionsInteractionData_singleSection(ref List<cgiSection_P_M_InteractionCli> sections_pm_interaction, int sectionIndex);
public bool getMxMyData_singleSection(ref List<cgiP_M_Cli> mx_my, int sectionIndex, double p);

public void getControlPointDescription(ref StringBuilder controlPointDescription, cgiControlPointEnum controlPoint);
public void clearConcreteSections();
public void clearConcreteSectionResult();

public delegate void ListMessageDelegate(string A_0, string A_1);
public delegate void StatusMessageDelegate(string A_0);
}
}

```

The following lists a few useful enums.

```

namespace cgiColumnBlazeCli
{
    public enum cgiBarLayoutModeEnum
    {
        BAR_LAYOUT_ALL_SIDES,
        BAR_LAYOUT_MAJOR_SIDES,
        BAR_LAYOUT_MINOR_SIDES,
        BAR_LAYOUT_EQUAL_SIDES,
    }

    public enum cgiConcreteCodeEnum
    {
        kACI_1999,
        kACI_2002,
        kACI_2005,
        kACI_2008,
        kACI_2011,
        kACI_2014,
        kACI_2019,
        kCode_End,
    }

    public enum cgiConfinementEnum
    {
        kConfine_Tied,
        kConfine_Spiral,
    }

    public enum cgiControlPointEnum
    {
        CONTROL_FY,
        CONTROL_05FY,
        CONTROL_025FY,
        CONTROL_0FY,
        CONTROL_FY_,
        CONTROL_005,
    }
}

```

```

CONTROL_FLEXURE,
CONTROL_PMAX,
MAX_CONTROL_POINTS,
}

public enum cgiErrorEnum
{
    kErrorNone,
    kInvalidInput,
    kSolverError,
    kAbnormalSolverTermination,
    kFileAccessError,
    kLicenseError,
    kNoResultAvailable,
    kInvalidSectionIndex,
    kInvalidRebar,
    kInvalidConcreteDesignCode,
    kInvalidAxialLoadForMxMy,
    kUnknownError,
}

public enum cgiIncludeEnum
{
    kInclude,
    kExclude,
}

public enum cgiSectionShapeEnum
{
    kRectangularC,
    kCircular,
    kTee,
    kInversedL,
    kI,
    kSemiCircular,
    kPolygon,
    kRectangularB,
    kGeneric,
}

public enum cgiStrengthReductionModeEnum
{
    kReduction_Auto,
    kReduction_Always_1,
}

public enum class cgiUnityRatioMethodEnum
{
    kUnityRatio_Max_Axial_Moment,
    kUnityRatio_Moment_Only
};

```

```

public enum cgiUnitEnum
{
    LENGTH,
    DIMENSION,
    FORCE,
    FORCE_LINE,
    MOMENT,
    MOMENT_LINE,
    FORCE_SURFACE,
    DISPLACEMENT_TRANS,
    DISPLACEMENT_ROTATE,
    TEMPERATURE,
    MODULUS,
    WEIGHT_DENSITY,
    REINFORCEMENT_AREA,
    STRESS,
    SPRING_TRANS_1D,
    SPRING_ROTATE_1D,
    SPRING_TRANS_2D,
    SPRING_TRANS_3D,
    UNIT_END,
}
}

```

The following lists result data structures.

```

namespace cgiColumnBlazeCli
{
    public class cgiPointCli
    {
        public double x;
        public double y;
        public double z;
    }

    public class cgiRcPointCli
    {
        public double x;
        public double y;
    }

    public class cgiConcreteSectionCli
    {
        public string szLabel;
        public cgiSectionShapeEnum iType;
        public cgiConfinementEnum iConfinement;
        public double b;
    }
}

```

```

public double h;
public double cover;
public double start_angle;
public double fc;
public double fy;
public string barDesignation;
public int nTopBars;
public int nBottomBars;
public int nLeftBars;
public int nRightBars;
public List<cgiRcPointCli> vPt;
public List<cgiRcPointCli> vOpeningPt1;
public List<cgiRcPointCli> vOpeningPt2;
public List<cgiRcPointCli> vOpeningPt3;
public double fTotalAs;
public double fSectionA;
public cgiRcPointCli ptCentroid;
public double fInertiaIx;
public double fInertiaIy;
public double fInertiaIxy;
public double fBarClearSpacing;
public double fIsx;
public double fIsy;
}

public class cgiLineBarCli
{
public cgiRcPointCli ptStart;
public cgiRcPointCli ptEnd;
public string barDesignation;
public int nCount;
}

public class cgiP_M_Cli
{
public double fC;
public double fStrain;
public double fPhi;
public double fP;
public double fMx;
public double fMy;
}

public class cgiP_M_InteractionCli
{
public double fAngle;
public List<cgiP_M_Cli> pm_control;
public List<cgiP_M_Cli> vPM;
}

```

```
public class cgiSection_P_M_InteractionCli
{
    public List<cgiP_M_InteractionCli> vPM_Int;
    public List<cgiP_M_InteractionCli> vPM_Int_Display;
    public List<cgiP_M_InteractionCli> vPM_Int_Display_Control;
}

public class cgiSectionLoadCli
{
    public double fP;
    public double fMx;
    public double fMy;
}
}
```

## Example

In this example, we are going to analyze a rectangular, circular, and generic sections, one at a time.

Section 1 (also see Example001 in cColumn)

[Example 7.11 and Example 7.12, “Design Guide on the ACI 318 Building Code Requirements for Structural Concrete” 1st Edition, Concrete Reinforcing Steel Institute, 2020]

Uniaxial Analysis, Square, ACI 318-19

*Rectangular section 18x18 inch, concrete cover 2.44 inch, 4#9,  $f_c' = 4$  ksi,  $f_y = 60$  ksi and  $f_y = 100$  ksi, tied confinement.*

Section 2 (also see Example002 in cColumn)

[Example 7.13, “Design Guide on the ACI 318 Building Code Requirements for Structural Concrete” 1st Edition, Concrete Reinforcing Steel Institute, 2020]

Uniaxial Analysis, Square, ACI 318-19

*Circular section diameter 20 inch, concrete cover 2.44 inch, 4#9,  $f_c' = 4$  ksi,  $f_y = 60$  ksi, tied confinement*

Section 3 (also see Example008 in cColumn)

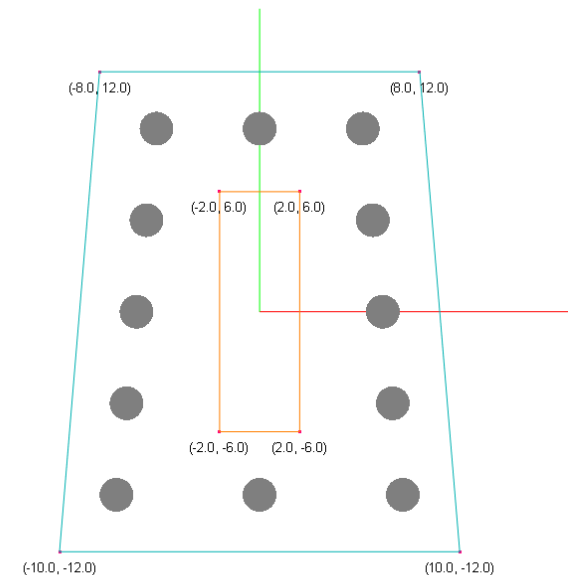
Investigate the trapezoid section with opening as shown here, according to ACI 318-2019. The section top and bottom widths are 16 and 20 inches respectively. The section height is 24 inches. The opening is 4 x 12 inches rectangle. The figure below shows the vertex coordinates of the section.

Find the section capacity ( $\phi P_n$  and  $\phi M_n$ ) corresponding to 50% tension at the bottom steel bars.

Material properties:  $f_c' = 6$  ksi,  $f_y = 60$  ksi

Concrete cover to the center of main bars = 2.8465 inches.

Reinforcement bars: 12 #14 as shown below.



## The following is the complete C# source code to set up the Example Model

```
static void Callback(string s0, string s)
{
    Console.WriteLine("{0}, {1}", s0, s);
}

static void StatusCallback(string s)
{
    Console.WriteLine("{0}", s);
}

static public void verify()
{
    RunModel();
}

static public void RunModel()
{
    cgiColumnBlazeClass solver = new cgiColumnBlazeClass();
    solver.createStructure(null); // pass a custom rebar database file if needed here

    cgiColumnBlazeClass.ListMessageDelegate listMsg = new cgiColumnBlazeClass.ListMessageDelegate(Callback);
    cgiColumnBlazeClass.StatusMessageDelegate statusMsg = new cgiColumnBlazeClass.StatusMessageDelegate(StatusCallback);
    solver.setListMessageFunction(listMsg);
    solver.setStatusMessageFunction(statusMsg);

    // LENGTH=ft; DIMENSION=in; FORCE=kip; FORCE_LINE=kip/ft; MOMENT=kip-ft; FORCE_SURFACE=lb/ft^2;
    // DISPLACEMENT_TRANS=in; DISPLACEMENT_ROTATE=rad; MODULUS=kip/in^2; WEIGHT_DENSITY=lb/ft^3; STRESS=lb/in^2
    // SPRING_TRANS_1D=lb/in; SPRING_ROTATE_1D=lb-in/rad; SPRING_TRANS_2D=kip/in^2; SPRING_TRANS_3D=kip/in^3
    // TEMPERATURE=Fahrenheit
    solver.setStandardEnglishUnits();

    if(true)
    { // cColumn Example001_60ksi
        solver.clearConcreteSections();
        solver.addRectangularSection("section1", 18, 18, 2.44, 4.0, 60.0, "#9", 2, 2, cgiConfinementEnum.kConfine_Tied);
        solver.setStrengthReductionMode(cgiStrengthReductionModeEnum.kReduction_Auto);
        solver.setConcreteSolverOption(16, 250, 20, true);

        bool success = solver.solve();
        if (!success)
        {
            var lastError = solver.getLastErrorMessage();
            Console.WriteLine("!!!!!!! error occurred: " + lastError);
        }
        //solver.saveDocument("c:\\temp\\test.rcs");

        DisplayResults(solver, true);
        DisplayResults(solver, false);
    }
}
```

```

if(true)
{
    // cColumn Example002
    solver.clearConcreteSections();
    solver.addCircularSection("section2", 20, 2.44, 4.0, 60.0, "#9", 4, 0, cgiConfinementEnum.kConfine_Spiral);
    solver.setStrengthReductionMode(cgiStrengthReductionModeEnum.kReduction_Auto); // kReduction_Always_1
    solver.setConcreteSolverOption(16, 250, 20, true);

    bool success = solver.solve();
    if (!success)
    {
        var lastError = solver.getLastError();
        Console.WriteLine("!!!!!!! error occured: " + lastError);
    }

    DisplayResults(solver, true);
    DisplayResults(solver, false);
}

if(true)
{
    // cColumn Example008
    solver.clearConcreteSections();
    List<cgiRcPointCli> boundary = new List<cgiRcPointCli>();
    boundary.Add(new cgiRcPointCli(-10, -12));
    boundary.Add(new cgiRcPointCli(10, -12));
    boundary.Add(new cgiRcPointCli(8, 12));
    boundary.Add(new cgiRcPointCli(-8, 12));

    List<cgiRcPointCli> opening1 = new List<cgiRcPointCli>();
    opening1.Add(new cgiRcPointCli(-2, -6));
    opening1.Add(new cgiRcPointCli(2, -6));
    opening1.Add(new cgiRcPointCli(2, 6));
    opening1.Add(new cgiRcPointCli(-2, 6));

    cgiLineBarCli lineBar1 = new cgiLineBarCli(new cgiRcPointCli(-7.1535, -9.1535), new cgiRcPointCli(-5.1535, 9.1535), "#14", 5);
    cgiLineBarCli lineBar2 = new cgiLineBarCli(new cgiRcPointCli(7.1535, -9.1535), new cgiRcPointCli(5.1535, 9.1535), "#14", 5);
    cgiLineBarCli lineBar3 = new cgiLineBarCli(new cgiRcPointCli(0, -9.1535), new cgiRcPointCli(0, 9.1535), "#14", 2);

    List<cgiLineBarCli> lineBars = new List<cgiLineBarCli>();
    lineBars.Add(lineBar1);
    lineBars.Add(lineBar2);
    lineBars.Add(lineBar3);
    solver.addGenericSection("section3", boundary, opening1, null, null, lineBars, 6.0, 60.0, cgiConfinementEnum.kConfine_Tied);
    solver.setStrengthReductionMode(cgiStrengthReductionModeEnum.kReduction_Auto);
    solver.setConcreteSolverOption(128, 250, 250, true);

    bool success = solver.solve();
    if (!success)
    {
        var lastError = solver.getLastError();
        Console.WriteLine("!!!!!!! error occured: " + lastError);
    }
}

```

```

        DisplayResults(solver, true);
        DisplayResults(solver, false);
    }
}

static private void DisplayResults(cgiColumnBlazeClass solver, bool useSingleSection)
{
    int angleSteps = 0, neutralSteps = 0, axialCapacitySteps = 0;
    bool excludeDisplaced = true;
    solver.getConcreteSolverOption(ref angleSteps, ref neutralSteps, ref axialCapacitySteps, ref excludeDisplaced);

    if (useSingleSection)
    {
        Console.WriteLine("----- use single section ----- ");
    }
    else
    {
        Console.WriteLine("----- not use single section ----- ");
    }

    Console.WriteLine("angleStepse " + angleSteps);
    Console.WriteLine("neutralSteps " + neutralSteps);
    Console.WriteLine("axialCapacitySteps " + axialCapacitySteps);
    Console.WriteLine("excludeDisplaced " + excludeDisplaced);
    Console.WriteLine("----- ");

    List<cgiConcreteSectionCli> sections = new List<cgiConcreteSectionCli>();
    solver.getConcreteSections(ref sections);

    foreach (cgiConcreteSectionCli section in sections)
    {
        Console.WriteLine(section.szLabel);
        Console.WriteLine("section type " + section.iType);
        Console.WriteLine("b " + section.b);
        Console.WriteLine("h " + section.h);
        Console.WriteLine("fSectionA " + section.fSectionA);
        Console.WriteLine("fTotalAs " + section.fTotalAs);
        Console.WriteLine("fInertiaIx " + section.fInertiaIx);
        Console.WriteLine("fInertiaIy " + section.fInertiaIy);
        Console.WriteLine("fInertiaIxy " + section.fInertiaIxy);
        Console.WriteLine("fBarClearSpacing " + section.fBarClearSpacing);
        Console.WriteLine("fIsx " + section.fIsx);
        Console.WriteLine("fIsy " + section.fIsy);
    }

    if (useSingleSection)
    {
        List<cgiSection_P_M_InteractionCli> sections_pm_interaction = new List<cgiSection_P_M_InteractionCli>();
        bool ok = false;
        ok = solver.getSectionsInteractionData_singleSection(ref sections_pm_interaction, sectionIndex: 1); // invalid section index
    }
}

```

```

Debug.Assert(sections_pm_interaction.Count == 0);
ok = solver.getSectionsInteractionData_singleSection(ref sections_pm_interaction, sectionIndex: 0);
Debug.Assert(sections_pm_interaction.Count == 1);
foreach (var pm_interaction in sections_pm_interaction[0].vPM_Int_Display_Control)
{
    Console.WriteLine("----- P-Mx diagram at angle 0 ----- ");
    Console.WriteLine("angle = " + pm_interaction.fAngle);

    Console.WriteLine("control points");
    for (int k = 0; k < pm_interaction.pm_control.Count; k++)
    {
        var pm = pm_interaction.pm_control[k];
        StringBuilder controlPointDescription = new StringBuilder();
        solver.getControlPointDescription(ref controlPointDescription, (cgiControlPointEnum)k);
        Console.WriteLine("{0}, c={1:f2}, P={2:f1}, Mx={3:f1}, My={4:f1}, max tensile strain={5:f5}, phi={6:f3}",
            controlPointDescription.ToString(), pm.fC, pm.fP, pm.fMx, pm.fMy, pm.fStrain, pm.fPhi);
    }
    Console.WriteLine();

    Console.WriteLine("P-M interaction points");
    for (int k = 0; k < pm_interaction.vPM.Count; k++)
    {
        var pm = pm_interaction.vPM[pm_interaction.vPM.Count - 1 - k];
        Console.WriteLine("{0:D4}, c={1:f2}, P={2:f1}, Mx={3:f1}, My={4:f1}, max tensile strain={5:f5}, phi={6:f3}",
            k + 1, pm.fC, pm.fP, pm.fMx, pm.fMy, pm.fStrain, pm.fPhi);
    }
    Console.WriteLine("----- ");
    break;
}
}
else
{
    List<cgiSection_P_M_InteractionCli> sections_pm_interaction = new List<cgiSection_P_M_InteractionCli>();
    solver.getSectionsInteractionData(ref sections_pm_interaction);
    foreach (cgiSection_P_M_InteractionCli section_pm_interaction in sections_pm_interaction)
    {
        foreach (var pm_interaction in section_pm_interaction.vPM_Int_Display_Control)
        {
            Console.WriteLine("----- P-Mx diagram at angle 0 ----- ");
            Console.WriteLine("angle = " + pm_interaction.fAngle);

            for (int k = 0; k < pm_interaction.vPM.Count; k++)
            {
                var pm = pm_interaction.vPM[pm_interaction.vPM.Count - 1 - k];
                Console.WriteLine("{0:D4}, c={1:f2}, P={2:f1}, Mx={3:f1}, My={4:f1}, max tensile strain={5:f5}, phi={6:f3}",
                    k + 1, pm.fC, pm.fP, pm.fMx, pm.fMy, pm.fStrain, pm.fPhi);
            }
            Console.WriteLine("----- ");
            break;
        }
        break;
    }
}
}

```

```

}

List<cgiP_M_Cli> mx_my = new List<cgiP_M_Cli>();
int sectionIndex = 0;
double p = 600;
solver.getMxMyData_singleSection(ref mx_my, sectionIndex, p);
Console.WriteLine("----- Mx-My diagram at P=600kips ----- ");
int i = 0;
foreach (var mxy in mx_my)
{
    Console.WriteLine("i={0:D3}, P={1:f1}, Mx={2:f1}, My={3:f1}, c={4:f2}, max tensile strain={5:f5}, angle={6:f3}",
        i + 1, mxy.fP, mxy.fMx, mxy.fMy, mxy.fC, mxy.fStrain, i * 360.0 / mx_my.Count);
        i++;
}
Console.WriteLine("----- ");
}

```

## The following is the detailed explanations for each steps in Example Model

To start, you create a `cgiColumnBlazeClass` object that represents the one and only structural model like this:

```
cgiColumnBlazeClass solver = new cgiColumnBlazeClass();
solver.createStructure(null); // pass a custom rebar database file if needed here
```

You can supply two optional callback functions for the solver to notify your application the progress of solution:

```
public delegate void ListMessageDelegate(string A_0, string A_1);
public delegate void StatusMessageDelegate(string A_0);

static void Callback(string s0, string s)
{
    Console.WriteLine("{0}, {1}", s0, s);
}

static void StatusCallback(string s)
{
    Console.WriteLine("{0}", s);
}

cgiColumnBlazeClass.ListMessageDelegate listMsg = new cgiColumnBlazeClass.ListMessageDelegate(Callback);
cgiColumnBlazeClass.StatusMessageDelegate statusMsg = new cgiColumnBlazeClass.StatusMessageDelegate(StatusCallback);
solver.setListMessageFunction(listMsg);
solver.setStatusMessageFunction(statusMsg);
```

Four unit systems are available in the library. They are Standard English unit, Standard Metric unit, Consistent English unit and Consistent Metric unit. For Standard English unit system, the following units are used for length, section dimension, force, moment etc.

```
// LENGTH=ft; DIMENSION=in;
// FORCE=kip; FORCE_LINE=kip/ft; MOMENT=kip-ft; FORCE_SURFACE=lb/ft^2;
// DISPLACEMENT_TRANS=in; DISPLACEMENT_ROTATE=rad; MODULUS=kip/in^2; WEIGHT_DENSITY=lb/ft^3; STRESS=lb/in^2
// SPRING_TRANS_1D=lb/in; SPRING_ROTATE_1D=lb-in/rad; SPRING_TRANS_2D=kip/in^2; SPRING_TRANS_3D=kip/in^3
// TEMPERATURE=Fahrenheit
solver.setStandardEnglishUnits();
```

For standard Metric unit system, the following units are used for length, section dimension, force, moment etc.

```
// LENGTH=m; DIMENSION=mm;
```

```
// FORCE=kN; FORCE_LINE=kN/m; MOMENT=kN-m; FORCE_SURFACE=kN/m^2;
// DISPLACEMENT_TRANS=mm; DISPLACEMENT_ROTATE=rad; MODULUS=kN/mm^2; WEIGHT_DENSITY=kN/m^3; STRESS=N/mm^2
// SPRING_TRANS_1D=N/mm; SPRING_ROTATE_1D=N-mm/rad; SPRING_TRANS_2D=N/mm^2; SPRING_TRANS_3D=N/mm^3
// TEMPERATURE=Celsius
solver.setStandardMetricUnits();
```

**By default, message boxes will be displayed if warnings or errors are encountered during the solution.** You can suppress the display of the message boxes by calling `solver.setShowSolverMessageBox(false)`. It is important to check the errors immediately after calling the static or frequency analysis solver by `solver.getLastError()`. The following are the error codes that ColumnBlaze currently may emit.

```
public enum cgiErrorEnum
{
    kErrorNone,
    kInvalidInput,
    kSolverError,
    kAbnormalSolverTermination,
    kFileAccessError,
    kLicenseError,
    kNoResultAvailable,
    kInvalidSectionIndex,
    kInvalidRebar,
    kInvalidConcreteDesignCode,
    kInvalidAxialLoadForMxMy,
    kUnknownError,
}
```

The last error code is cleared (set to `kErrorNone`) at the beginning of each solution.

Since we are solving one section at a time, we clear the existing sections before adding a new one.

```
solver.clearConcreteSections();
```

A new section is added by `addRectangularSection()`, `addCircularSection()`, or `addGenericSection()`. The section label must be less than 127 characters long. A new section will not be added if a section with the label already exists in the model. You can check the return value of `addRectangularSection()`, `addCircularSection()`, and `addGenericSection()` to see if a section is added successfully.

```
solver.addRectangularSection("section1", 18, 18, 2.44, 4.0, 60.0, "#9", 2, 2, cgiConfinementEnum.kConfine_Spiral);
solver.addCircularSection("section2", 20, 2.44, 4.0, 60.0, "#9", 4, 0, cgiConfinementEnum.kConfine_Spiral);

List<cgiRcPointCli> boundary = new List<cgiRcPointCli>();
```

```

boundary.Add(new cgiRcPointCli(-10, -12));
boundary.Add(new cgiRcPointCli(10, -12));
boundary.Add(new cgiRcPointCli(8, 12));
boundary.Add(new cgiRcPointCli(-8, 12));

List<cgiRcPointCli> opening1 = new List<cgiRcPointCli>();
opening1.Add(new cgiRcPointCli(-2, -6));
opening1.Add(new cgiRcPointCli(2, -6));
opening1.Add(new cgiRcPointCli(2, 6));
opening1.Add(new cgiRcPointCli(-2, 6));

cgiLineBarCli lineBar1 = new cgiLineBarCli(new cgiRcPointCli(-7.1535, -9.1535),
new cgiRcPointCli(-5.1535, 9.1535), "#14", 5);
cgiLineBarCli lineBar2 = new cgiLineBarCli(new cgiRcPointCli(7.1535, -9.1535),
new cgiRcPointCli(5.1535, 9.1535), "#14", 5);
cgiLineBarCli lineBar3 = new cgiLineBarCli(new cgiRcPointCli(0, -9.1535), new cgiRcPointCli(0, 9.1535), "#14", 2);

List<cgiLineBarCli> lineBars = new List<cgiLineBarCli>();
lineBars.Add(lineBar1);
lineBars.Add(lineBar2);
lineBars.Add(lineBar3);
solver.addGenericSection("section3", boundary, opening1, null, null, lineBars, 6.0, 60.0,
cgiConfinementEnum.kConfine_Tied);

```

There are two modes in strength reduction: `kReduction_Auto` and `kReduction_Always_1`. You should always use `kReduction_Auto` where strength reduction factor  $\phi$  is automatically applied. `kReduction_Always_1` option is provided for academic reasons.

```
solver.setStrengthReductionMode(cgiStrengthReductionModeEnum.kReduction_Auto);
```

For accuracy and solver solution time reasons, it is important to set up proper solver options by calling `setConcreteSolverOption()`.

```
solver.setConcreteSolverOption(128, 250, 250, true);
```

The angle steps (1st parameter) must be of multiple of 4. For uniaxial solution, the value of 4 is enough. For biaxial solution, the value must be 16 or larger. The neutral steps (2nd parameter) should be large enough to capture capacities at the different neutral axis positions (e.g.: 250). The axial capacity steps (3<sup>rd</sup> parameter) is for result data presentation and display. It should be larger than 20. Lastly, you should set the exclude the concrete displaced by the steel reinforcement (4<sup>th</sup> parameter) to be true. The option to set it to false is provided for academic reasons.

By default, the solution will abort by pressing ESC key during the solution. You can customize the key by calling `setAbortSolutionKey()`.

To solve the model, simply call

```
bool success = solver.solve();
```

You may optionally save the input to a file that can be opened by cColumn software. This can be useful if you would like to view the result data in spreadsheets or graphical diagrams.

```
solver.saveDocument("c:\\temp\\test.rcs");
```

The sections including some calculated properties can be retrieved by `getConcreteSections()`.

```
List<cgiConcreteSectionCli> sections = new List<cgiConcreteSectionCli>();
solver.getConcreteSections(ref sections);

foreach (cgiConcreteSectionCli section in sections)
{
    Console.WriteLine(section.szLabel);
    Console.WriteLine("section type " + section.iType);
    Console.WriteLine("b " + section.b);
    Console.WriteLine("h " + section.h);
    Console.WriteLine("fSectionA " + section.fSectionA);
    Console.WriteLine("fTotalAs " + section.fTotalAs);
    Console.WriteLine("fInertiaIx " + section.fInertiaIx);
    Console.WriteLine("fInertiaIy " + section.fInertiaIy);
    Console.WriteLine("fInertiaIxy " + section.fInertiaIxy);
    Console.WriteLine("fBarClearSpacing " + section.fBarClearSpacing);
    Console.WriteLine("fIsx " + section.fIsx);
    Console.WriteLine("fIsy " + section.fIsy);
}
```

The section P-M interaction results can be retrieved easily by calling either `getSectionsInteractionData_singleSection()` function for single section or `getSectionsInteractionData()` function for all sections as illustrated in the following. The section Mx-My result data at a given axial load can be retrieved by calling `getSectionsInteractionData_singleSection()`.

```
// use getSectionsInteractionData_singleSection()
List<cgiSection_P_M_InteractionCli> sections_pm_interaction = new List<cgiSection_P_M_InteractionCli>();
bool ok = false;
// invalid section index
```

```

ok = solver.getSectionsInteractionData_singleSection(ref sections_pm_interaction, sectionIndex: 1);
Debug.Assert(sections_pm_interaction.Count == 0);
ok = solver.getSectionsInteractionData_singleSection(ref sections_pm_interaction, sectionIndex: 0);
Debug.Assert(sections_pm_interaction.Count == 1);
foreach (var pm_interaction in sections_pm_interaction[0].vPM_Int_Display_Control)
{
    Console.WriteLine("----- P-Mx diagram at angle 0 ----- ");
    Console.WriteLine("angle = " + pm_interaction.fAngle);

    Console.WriteLine("control points");
    for (int k = 0; k < pm_interaction.pm_control.Count; k++)
    {
        var pm = pm_interaction.pm_control[k];
        StringBuilder controlPointDescription = new StringBuilder();
        solver.getControlPointDescription(ref controlPointDescription, (cgiControlPointEnum)k);
        Console.WriteLine("{0}, c={1:f2}, P={2:f1}, Mx={3:f1}, My={4:f1}, max tensile strain={5:f5}, phi={6:f3}",
            controlPointDescription.ToString(), pm.fC, pm.fP, pm.fMx, pm.fMy, pm.fStrain, pm.fPhi);
    }
    Console.WriteLine();

    Console.WriteLine("P-M interaction points");
    for (int k = 0; k < pm_interaction.vPM.Count; k++)
    {
        var pm = pm_interaction.vPM[pm_interaction.vPM.Count - 1 - k];
        Console.WriteLine("{0:D4}, c={1:f2}, P={2:f1}, Mx={3:f1}, My={4:f1}, max tensile strain={5:f5},
            phi={6:f3}", k + 1, pm.fC, pm.fP, pm.fMx, pm.fMy, pm.fStrain, pm.fPhi);
    }
    break;
}

// use getSectionsInteractionData()
List<cgiSection_P_M_InteractionCli> sections_pm_interaction = new List<cgiSection_P_M_InteractionCli>();
solver.getSectionsInteractionData(ref sections_pm_interaction);
foreach (cgiSection_P_M_InteractionCli section_pm_interaction in sections_pm_interaction)
{
    foreach (var pm_interaction in section_pm_interaction.vPM_Int_Display_Control)
    {
        Console.WriteLine("----- P-Mx diagram at angle 0 ----- ");
        Console.WriteLine("angle = " + pm_interaction.fAngle);
        for (int k = 0; k < pm_interaction.vPM.Count; k++)
        {
            var pm = pm_interaction.vPM[pm_interaction.vPM.Count - 1 - k];
            Console.WriteLine("{0:D4}, c={1:f2}, P={2:f1}, Mx={3:f1}, My={4:f1}, max tensile strain={5:f5},
                phi={6:f3}", k + 1, pm.fC, pm.fP, pm.fMx, pm.fMy, pm.fStrain, pm.fPhi);
        }
    }
}

```

```

        phi={6:f3}", k + 1, pm.fC, pm.fP, pm.fMx, pm.fMy, pm.fStrain, pm.fPhi);
    }
    break;
}
break;
}

// use getMxMyData_singleSection
List<cgiP_M_Cli> mx_my = new List<cgiP_M_Cli>();
int sectionIndex = 0;
double p = 600;
solver.getMxMyData_singleSection(ref mx_my, sectionIndex, p);
Console.WriteLine("----- Mx-My diagram at P=600 kips ----- ");
int i = 0;
foreach (var mxy in mx_my)
{
    Console.WriteLine("i={0:D3}, P={1:f1}, Mx={2:f1}, My={3:f1}, c={4:f2}, max tensile strain={5:f5},
        angle={6:f3}", i + 1, mxy.fP, mxy.fMx, mxy.fMy, mxy.fC, mxy.fStrain, i * 360.0 / mx_my.Count);
    i++;
}

```

ColumnBlaze uses unity ratio to check one or more loads against one or more sections as shown in the following:

```

solver.setUnityRatioMethod(cgiUnityRatioMethodEnum.kUnityRatio_Max_Axial_Moment);
cgiSectionLoadCli load = new cgiSectionLoadCli(100, 20, 10);
int sectionIndex = 0;
double unityRatio = 999.0;
bool ok = solver.checkLoad_singleSection(ref unityRatio, sectionIndex, load);

double criticalUnityRatio = 999.0;
int criticalSectionIndex = 0;
solver.checkLoad(ref criticalUnityRatio, ref criticalSectionIndex, load);

```

There are two methods to calculate unity ratio: `kUnityRatio_Max_Axial_Moment` and `kUnityRatio_Moment_Only` (not recommended). A unity ratio of less than 1.0 means the section(s) is adequate against a given load. Conversely, a unity ratio of greater than 1.0 means the section(s) is not adequate against a given load. Use `checkLoad_singleSection()` to check a load against a single section. Use `checkLoad()` to check a load against all sections.

This last page is left blank on purpose