

SolverBlaze Finite Element Library (SDK)

Quick Guide

Computations & Graphics, Inc.

Highlands Ranch, CO 80130, USA

Email: info@cg-inc.com

Web: www.cg-inc.com

Contents

Introduction.....	3
C++ Interface	6
Example Model One (refer to the sample file verify-example4.cpp included in the library).....	21
Example Model Two (refer to the sample file verify-plate-patch-test.cpp included in the library).....	40
.NET Interface through C++/CLI	47
Example Model Three (refer to the sample file Example-04.cs included in the library)	60
Example Model Four (refer to the sample file verify-plate-patch-test.cs sample included in the library).....	78
.NET Interface through P/Invoke.....	85
Example Model Five (refer to the sample file verify-example4-interop.cs included in the library)	97

Introduction

Computations & Graphics, Inc. (CGI) SolverBlaze Finite Element Library is a powerful structural and finite element analysis API (Application Programming Interface). It is based on the time-tested finite element solver engine in Real3D, which has been sub-licensed to over hundreds civil and structural engineering offices in the United States and around the world. You can use this reliable and user-friendly API to develop your custom software royalty-free.

The following are the main features:

- Supports beam, truss, plate and shell (thin and thick, compatible and incompatible formulations) as well as brick elements.
- Supports nodal, point, line, surface and area loads.
- Supports thermal loads.
- Supports conversion from area loads to line loads, local loads to global loads.
- Supports linear and nonlinear nodal, line and surface springs.
- Supports advanced coupled springs.
- Supports tension only/compression only members.
- Supports moment releases, rigid elements, and rigid diaphragms.
- Supports inactive elements.
- Supports element stiffness modifications.
- Supports forced displacements.
- Supports multi-DOF constraints such as inclined support.
- Supports both English and Metric units.
- Supports static linear and P-Delta analysis.
- Supports frequency (eigenvalue and eigenvector) analysis.
- Supports response spectrum analysis.
- Supports classic skyline solver
- Supports extremely fast sparse solver (x64 only) that can handle models with millions of degrees of freedom.
- Supports unique quad-precision solver (x64 only) for numerically challenging problems such as rigid diaphragms.
- Automatically generate input and result report in plain text and html format
- **Provides bidirectional communications between SolverBlaze and Real3D:** you can open and graphically view SolverBlaze models in Real3D, or generate SolverBlaze source code from the current Real3D model.
- Supports .NET 4.0, 4.5x, 4.6x, 4.7x, 4.8x and .NET Core 5.0, 6.0, 7.0, 8.0, 9.0, 10.0 in C# and VB.NET languages.
- Supports native C++ language.
- Supports x64 CPUs architecture on Windows.

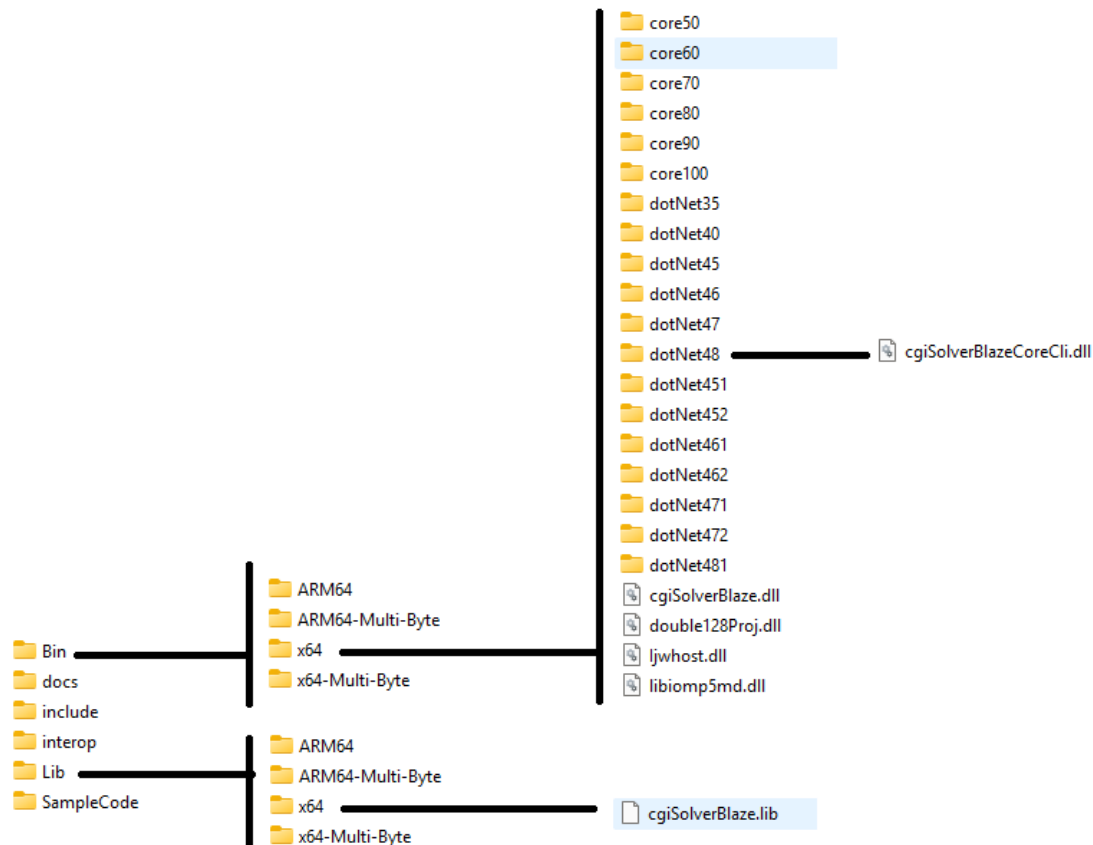
- Supports ARM64 CPUs architecture on Windows through C++ and P/Invoke.
- Supports Unicode and Non-Unicode in Microsoft Visual Studio 2015 or above.
- Available in both binary library and source code forms.
- A free copy of Professional Real3D Program for source code customers.
- Provides numerous source code samples in C++ and C# free of charge.
- Offers technical support by the SolverBlaze 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 x64, ARM64 redistributables as the SolverBlaze library links dynamically with the visual C++ runtime library.

Since SolverBlaze is based on the finite element package Real3D, it is highly recommended that you install Real3D and get familiar with its capabilities. You can download an evaluation of Real3D from <http://www.cg-inc.com/download/download>. The features in SolverBlaze has one-to-one correspondence with those in Real3D. You can use Real3D to visualize 3D structural model created in SolverBlaze, perform analysis, then view the results (numerical, diagrams, and contours). You can also use Real3D to automatically generate ready-to-compile SolverBlaze source code from an existing Real3D model through its File->Advanced->Generate SolverBlaze Source Code menu.

The following image shows the structure of the SDK. It includes **Bin** folder which contains executables for both native DLLs and .Net 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 sample code will output to relevant folders under **SampleCode\Output** folder.

The file `cgiSolverBlaze.dll` is a Windows native DLL. It depends on two other Windows native DLLs `libiomp5md.dll` and `double128Proj.dll` on x64 platform. Make sure these files have consistent CPU architecture and are present in the executable directory.



C++ Interface

The C++ library contains both a x64 and ARM64 Windows DLL `cgiSolverBlaze.dll`. The interface includes the following header files: `_cgiDefines.h`, `_cgiIStructure.h`, `_cgiPlatform.h` and `cgiSolverBlaze.h`. It also includes a `cgiSolverBlaze.lib` for linking to your projects. `cgiSolverBlaze.dll` x64 version depends on two other Windows native DLLs `libiomp5md.dll` and `double128Proj.dll`.

The `_cgiDefines.h` defines all input and output data structures. `_cgiIStructure.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 `cgiSolverBlazeClient`. These examples are taken from the Verification Manual of Real3D, which is a structural analysis and finite element analysis program by CGI. These examples include 2D/3D frame analysis, plate bending analysis, frequency analysis, and response spectrum analysis. A full input and output report can be produced after each analysis. You can compare the reports with those produced from within Real3D.

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

```
struct CGISOLVERBLAZE_API cgiIStructure
{
    virtual void setListMessageFunction(fnLISTMSG fnListMsg)=0;
    virtual void setStatusMessageFunction(fnSTATUSMSG fnStatusMsg)=0;
    virtual void setSparseSolverProgressFunction(fnMKLPROGRESS fnMlkProgress)=0;

    virtual void getExePath(TCHAR exePath[], int size)const=0;
    virtual void getDefaultTestPath(TCHAR testPath[], 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 modelNamep[])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 void clearModel()=0; // clear all input except unit settings

// enum {kModel_Frame3D, kModel_Frame2D, kModel_Truss3D, kModel_Truss2D,
// kModel_PlateBending, kModel_PlaneStress, kModel_Brick, kModel_Grillage, kModel_End};
virtual void setModelType(int nModelType)=0;
virtual int getModelType()const=0;

// 0=free, 1=suppressed
virtual void setSuppressedDOFs(const bool bSuppress[6])=0;
virtual void getSuppressedDOFs(bool bSuppress[6])const=0;

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

virtual void setAnalysisOptions(bool bConsiderBeamShearDeformation, int nMaximumPDeltaIterations,
                                double fPDeltaToleranceInPercentage, int nNumberOfSegmentsForBeamOutput, bool bUseThinPlate,
                                bool bUseCompatibleModes, int nUseAverageStress)=0;
virtual void getAnalysisOptions(bool& bConsiderBeamShearDeformation, int& nMaximumPDeltaIterations,
                                double& fPDeltaToleranceInPercentage, int& nNumberOfSegmentsForBeamOutput, bool& bUseThinPlate,
                                bool& bUseCompatibleModes, int& nUseAverageStress)const=0;

virtual void setFrequencyAnalysisOptions(int nEigenNumber, double fEigenVlaueTolerance, int nMaximumSubspaceIterations,
                                          int nIterationVectors, bool bConvertLoadToMass, int nGravityDirection,
                                          int nLoadCombinationForMass)=0;
virtual void getFrequencyAnalysisOptions(int& nEigenNumber, double& fEigenVlaueTolerance, int& nMaximumSubspaceIterations,
                                          int& nIterationVectors, bool& bConvertLoadToMass, int& nGravityDirection,
                                          int& nLoadCombinationForMass)const =0;

```

```

virtual void setResponseSpectrumAnalysisOptions(cgiSolverBlazeNamespace::cgiSpectrum spectrums[3], double fDampingRatio,
                                               int combinationMethod, double fSpectrumDirectionalFactor[3],
                                               bool bUseDominantModeForSignage)=0;
virtual void getResponseSpectrumAnalysisOptions(cgiSolverBlazeNamespace::cgiSpectrum spectrums[3], double& fDampingRatio,
                                               int& combinationMethod, double fSpectrumDirectionalFactor[3],
                                               bool& bUseDominantModeForSignage) = 0;

virtual void setApplyStiffnessModification(bool bApply)=0;
virtual bool getApplyStiffnessModification()const=0;

virtual void setTolerance(double fTolerance)=0;
virtual double getTolerance()const=0;
virtual void setFictitiousOzFactor(double fFictitiousOzFactor) = 0;
virtual double getFictitiousOzFactor()const = 0;
virtual void setUseOoc(bool bUseOoc) = 0; // Ooc = Out of core solver
virtual bool getUseOoc()const = 0;
virtual void setEpsilon(double fEpsilon) = 0;
virtual double getEpsilon()const = 0;

// 0 for center only, 1 for nodes only, 2 for both center and nodes
virtual void setStressLocation(int nStressLocation)=0;
virtual int getStressLocation()const=0;
virtual void setGravityFactor(double fGravityFactor)=0;
virtual double getGravityFactor()const=0;
virtual void setGravityDirection(int nGravityDirection) = 0;
virtual int getGravityDirection()const = 0;
virtual void setGravityLoadCase(int nGravityLoadCase) = 0;
virtual int getGravityLoadCase()const = 0;
virtual void setUseZAsVerticalAxis(bool useZAsVerticalAxis) = 0;
virtual bool getUseZAsVerticalAxis()const = 0;
virtual void setSolver(int iSolver)=0;
virtual int getSolver()const=0;

virtual void deleteMemoryArray(int* p)const = 0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiMaterial* p)const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiSection* p)const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiThickness* p)const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiSupport* p)const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiSpring* p)const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiCoupledSpring* p)const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiRelease* p)const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiDiaphragm* p)const=0;

```

```

virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiMultiDofConstraint* p) const = 0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiNode* p) const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiBeam* p) const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiShell4* p) const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiBrick* p) const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiLoadCase* p) const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiLoadCombination* p) const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiLoadCaseLoad* p) const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiNodalMass* p) const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiCombinationResult* p) const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiEigenValueVectorResult* p) const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiResponseSpectrumResult* p) const = 0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiEnvelopeValue6* p) const = 0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiEnvelopeBmVMD* p) const = 0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiIntersectionNode* p) const = 0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiMergedInfo* p) const = 0; // single element really
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiBeamStress*) const = 0;

virtual void setMaterials(const cgiSolverBlazeNamespace::cgiMaterial* vMat, int count)=0;
virtual void getMaterials(cgiSolverBlazeNamespace::cgiMaterial*& vMat, int& count) const=0;
virtual void setSections(const cgiSolverBlazeNamespace::cgiSection* vSect, int count)=0;
virtual void getSections(cgiSolverBlazeNamespace::cgiSection*& vSect, int& count) const=0;
virtual void setThicknesses(const cgiSolverBlazeNamespace::cgiThickness* vThick, int count)=0;
virtual void getThicknesses(cgiSolverBlazeNamespace::cgiThickness*& vThick, int& count) const=0;

virtual void setSupports(const cgiSolverBlazeNamespace::cgiSupport* vSupt, int count)=0;
virtual void getSupports(cgiSolverBlazeNamespace::cgiSupport*& vSupt, int& count) const=0;
virtual void setNodalSprings(const cgiSolverBlazeNamespace::cgiSpring* vNdSpring, int count)=0;
virtual void getNodalSprings(cgiSolverBlazeNamespace::cgiSpring*& vNdSpring, int& count) const=0;
virtual void setCoupledSprings(const cgiSolverBlazeNamespace::cgiCoupledSpring* vCoupledSpring, int count)=0;
virtual void getCoupledSprings(cgiSolverBlazeNamespace::cgiCoupledSpring*& vCoupledSpring, int& count) const=0;
virtual void setLineSprings(const cgiSolverBlazeNamespace::cgiSpring* vLnSpring, int count)=0;
virtual void getLineSprings(cgiSolverBlazeNamespace::cgiSpring*& vLnSpring, int& count) const=0;
virtual void setSurfaceSprings(const cgiSolverBlazeNamespace::cgiSpring* vShell4Spring, int count)=0;
virtual void getSurfaceSprings(cgiSolverBlazeNamespace::cgiSpring*& vShell4Spring, int& count) const=0;
virtual void setMomentReleases(const cgiSolverBlazeNamespace::cgiRelease* vRels, int count)=0;
virtual void getMomentReleases(cgiSolverBlazeNamespace::cgiRelease*& vRels, int& count) const=0;

virtual void setDiaphragms(const cgiSolverBlazeNamespace::cgiDiaphragm* vDiaphragm, int count)=0;
virtual void getDiaphragms(cgiSolverBlazeNamespace::cgiDiaphragm*& vDiaphragm, int& count) const=0;
virtual void setDiaphragmOptions(double fDiaphragmStiffnessFactor, bool bConsiderDiaphragm)=0;
virtual void getDiaphragmOptions(double& fDiaphragmStiffnessFactor, bool& bConsiderDiaphragm) const=0;

```

```

virtual void setMultiDofConstraints(const cgiSolverBlazeNamespace::cgiMultiDofConstraint* vMultiDofConstraint, int count) = 0;
virtual void getMultiDofConstraints(cgiSolverBlazeNamespace::cgiMultiDofConstraint*& vMultiDofConstraint, int& count)const = 0;

virtual void setNodes(const cgiSolverBlazeNamespace::cgiNode* vNd, int count)=0;
virtual void getNodes(cgiSolverBlazeNamespace::cgiNode*& vNd, int& count)const=0;
virtual void setBeams(const cgiSolverBlazeNamespace::cgiBeam* vBm, int count)=0;
virtual void getBeams(cgiSolverBlazeNamespace::cgiBeam*& vBm, int& count)const=0;
virtual void setShell4s(const cgiSolverBlazeNamespace::cgiShell4* vShell4, int count)=0;
virtual void getShell4s(cgiSolverBlazeNamespace::cgiShell4*& vShell4, int& count)const=0;
virtual void setBricks(const cgiSolverBlazeNamespace::cgiBrick* vBrick, int count)=0;
virtual void getBricks(cgiSolverBlazeNamespace::cgiBrick*& vBrick, int& count)const=0;

virtual void getSingleNode(cgiSolverBlazeNamespace::cgiNode& node, int nId)const=0;
virtual void getSingleBeam(cgiSolverBlazeNamespace::cgiBeam& beam, int nId)const=0;
virtual void getSingleShell4(cgiSolverBlazeNamespace::cgiShell4& shell4, int nId)const=0;
virtual void getSingleBrick(cgiSolverBlazeNamespace::cgiBrick& brick, int nId)const=0;

virtual double getBeamLocalAngleByThirdPoint(cgiSolverBlazeNamespace::cgiPoint& beamStartPt,
        cgiSolverBlazeNamespace::cgiPoint& beamEndPt, cgiSolverBlazeNamespace::cgiPoint& thirdPt)const = 0;
virtual void getBeamLocalAxes(cgiSolverBlazeNamespace::cgiPoint& xAxis, cgiSolverBlazeNamespace::cgiPoint& yAxis,
        cgiSolverBlazeNamespace::cgiPoint& zAxis, const cgiSolverBlazeNamespace::cgiPoint& point1,
        const cgiSolverBlazeNamespace::cgiPoint& point2, double fGamma)const=0;
virtual void getShellLocalAxes(cgiSolverBlazeNamespace::cgiPoint& xAxis, cgiSolverBlazeNamespace::cgiPoint& yAxis,
        cgiSolverBlazeNamespace::cgiPoint& zAxis, const cgiSolverBlazeNamespace::cgiPoint& point1,
        const cgiSolverBlazeNamespace::cgiPoint& point2, const cgiSolverBlazeNamespace::cgiPoint& point3,
        const cgiSolverBlazeNamespace::cgiPoint& point4, double fGamma)const=0;
virtual bool matchShellLocalxAxis(int sourceShellId, int targetShellId)= 0;
virtual bool matchShellLocalzAxis(int sourceShellId, int targetShellId)= 0;

virtual void setLoadCases(const cgiSolverBlazeNamespace::cgiLoadCase* vLoadCase, int count)=0;
virtual void getLoadCases(cgiSolverBlazeNamespace::cgiLoadCase*& vLoadCase, int& count)const=0;
virtual void setLoadCombinations(const cgiSolverBlazeNamespace::cgiLoadCombination* vLoadComb, int count)=0;
virtual void getLoadCombinations(cgiSolverBlazeNamespace::cgiLoadCombination*& vLoadComb, int& count)const=0;
virtual void setCaseLoads(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* vCaseLoad, int count)=0;
virtual void getCaseLoads(cgiSolverBlazeNamespace::cgiLoadCaseLoad*& vCaseLoad, int& count)const=0;
virtual void setNodalMasses(const cgiSolverBlazeNamespace::cgiNodalMass* vNdMass, int count)=0;
virtual void getNodalMasses(cgiSolverBlazeNamespace::cgiNodalMass*& vNdMass, int& count)const=0;
virtual void getCalculatedNodalMasses(cgiSolverBlazeNamespace::cgiNodalMass*& vNdCombMass, int& count)const=0;
virtual void convertLocalLoadsToGlobalLoads()=0;
virtual void convertAreaLoadsToLineLoads()=0;

virtual void setReportOptions(const cgiSolverBlazeNamespace::cgiReportOptions& reportOptions)=0;
virtual void getReportOptions(cgiSolverBlazeNamespace::cgiReportOptions& reportOptions)const=0;

```

```

virtual int getEquations()const=0;
virtual void getStaticResults(cgiSolverBlazeNamespace::cgiCombinationResult*& result, int& count)const=0;
virtual int getStaticResultsForSingleLoadCombination(cgiSolverBlazeNamespace::cgiCombinationResult*& result,
                                                    int loadCombinationIndex)const=0;
virtual bool getNodalDisplacementsEnvelope(cgiSolverBlazeNamespace::cgiEnvelopeValue6*& result, int& valueCount,
                                           int* envelopeLoadCombinationIndexes, int loadCombCount)const = 0;
virtual bool getSupportsEnvelope(cgiSolverBlazeNamespace::cgiEnvelopeValue6*& result, int& valueCount,
                                 int* envelopeLoadCombinationIndexes, int loadCombCount)const = 0;
virtual bool getBeamVmdEnvelope(cgiSolverBlazeNamespace::cgiEnvelopeBmVMD*& result, int& valueCount,
                                int* envelopeLoadCombinationIndexes, int loadCombCount)const = 0;
// pass a valid shell id for result in shell local directions, or pass referenceShellId=-1 for result in global directions
virtual bool getShell4GroupNodalResultant(cgiSolverBlazeNamespace::cgiNodalResultant& result, int loadCombinationIndex,
                                           int* selectedNodes, int selectNodeCount, int* selectedShell4s, int selectedShell4Count,
                                           const cgiSolverBlazeNamespace::cgiPoint& resultLocation, int referenceShellId)const = 0;

virtual bool getBeamStressesForSingleLoadCombination(cgiSolverBlazeNamespace::cgiBeamStress*& result, int& count,
                                                    int loadCombinationIndex)const = 0;

virtual void getEigenResults(cgiSolverBlazeNamespace::cgiEigenValueVectorResult*& result, int& count)const=0;
virtual int getEigenResultsForSingleMode(cgiSolverBlazeNamespace::cgiEigenValueVectorResult*& result, int modeIndex)const=0;

// count = number of natural periods (or frequencies)
virtual void getResponseSpectrumResults(cgiSolverBlazeNamespace::cgiResponseSpectrumResult*& result, int& count)const=0;
virtual int getResponseSpectrumResultsForSingleMode(cgiSolverBlazeNamespace::cgiResponseSpectrumResult*& result,
                                                    int modeIndex)const=0;

virtual void getModalCombinationResults(cgiSolverBlazeNamespace::cgiCombinationResult& result)const = 0;
virtual void getModalCombinationBaseShears(double& fBaseShearX, double& fBaseShearY, double& fBaseShearZ)const=0;

virtual bool saveDocument(LPCTSTR lpszPathName)=0;
virtual bool openDocument(LPCTSTR lpszPathName)=0;
virtual void clearResult()=0;
virtual void clearStaticResult()=0;
virtual void clearFrequencyResult()=0;
virtual void clearResponseSpectrumResult() = 0;
virtual bool checkInputData(int iMode, bool bReport = false)=0; // MODE_STATIC, MODE_FREQUENCY
virtual bool hasStaticSolution()=0;
virtual bool hasEigenSolution()=0;
virtual bool hasResponseSpectrumSolution() = 0;
virtual bool runStaticAnalysis()=0;
virtual bool runFrequencyAnalysis(bool bCalcMassOnly)=0;

```

```

virtual bool runResponseSpectrumAnalysis() = 0;
virtual bool runReport()=0;
virtual void getContourMinMax(double& fMin, double& fMax, int iContourIndex, int iComb, int iStressAtShellTopBottom)=0;

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

virtual int removeAllOrphanedNodes(int*& nodeIds, int& count) = 0;
virtual bool mergeAllNodesAndElements(cgiSolverBlazeNamespace::cgiMergedInfo*& mergedInfo) = 0;
virtual bool insertNodesAtBeamIntersections(cgiSolverBlazeNamespace::cgiIntersectionNode*& intersectionNodes,
                                             int& intersectionNodeCount, int* beamIds, int beamCount) = 0;
virtual int explodeBeamsAtNodes(int*& affectedBeamIds, int& affectedBeamCount, int* beamIds, int beamCount) = 0;

// clear functions
virtual void clearMaterials() = 0;
virtual void clearSections() = 0;
virtual void clearThicknesses() = 0;
virtual void clearSupports() = 0;
virtual void clearNodalSprings() = 0;
virtual void clearCoupledSprings() = 0;
virtual void clearLineSprings() = 0;
virtual void clearSurfaceSprings() = 0;
virtual void clearMomentReleases() = 0;
virtual void clearDiaphragms() = 0;
virtual void clearMultiDofConstraints() = 0;
virtual void clearNodes() = 0;
virtual void clearBeams() = 0;
virtual void clearShell4s() = 0;
virtual void clearBricks() = 0;
virtual void clearLoadCases() = 0;
virtual void clearLoadCombinations() = 0;
virtual void clearCaseLoads() = 0;
virtual void clearNodalMasses() = 0;
};

CGISOLVERBLAZE_API cgiIStructure* CreateStructure(LPCTSTR szRebarDatabaseFile = nullptr);
CGISOLVERBLAZE_API void DeleteStructure(cgiIStructure* 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};

// model type
enum {kModel_Frame3D, kModel_Frame2D, kModel_Truss3D, kModel_Truss2D,
      kModel_PlateBending, kModel_PlaneStress, kModel_Brick, kModel_Grillage, kModel_End};

// Degree of Freedoms
enum {X=0, Y, Z, OX, OY, OZ};

// load coordinate system (projected is not supported at this time)
enum {LOCAL=0, GLOBAL, PROJECTED};

// distance specification
enum {PERCENT = 0, DISTANCE = 1};

// gravity inclusion/exclusion
enum { kInclude, kExclude };

// member nonlinearity
enum {kMemberLinear, kMemberTensionOnly, kMemberCompressionOnly};

// diaphragm types
enum{kDiaphragm_Generic=0, kDiaphragm_XZ, kDiaphragm_YZ, kDiaphragm_XY, kDiaphragm_End};

// coupled spring stiffness terms
// units are in SPRING_TRANS_1D, FORCE/DISPLACEMENT_ROTATE, and SPRING_ROTATE_1D for corresponding terms
enum {
  Kx_Kx, Kx_Ky, Kx_Kz, Kx_Kox, Kx_Koy, Kx_Koz,
  Ky_Ky, Ky_Kz, Ky_Kox, Ky_Koy, Ky_Koz,
  Kz_Kz, Kz_Kox, Kz_Koy, Kz_Koz,
  Kox_Kox, Kox_Koy, Kox_Koz,
  Koy_Koy, Koy_Koz,
  Koz_Koz,
  kCoupledSpringTermsEnd
};

```

```

// area load distribution methods
enum {kTwoWay, kShortSides, kLongSides, kAB_CD, kBC_AD, kCentroid, kCircumference, kAreaLoad_Distribution_End};

// solver types
enum {kSolver64=0, kSolver128, kSolverSparse64};

// stress averaging types
enum {kStressAveragingNone, kStressAveragingAll, kStressAveragingLocal};

// response spectrum modal combination methods
enum { kCombination_Cqc, kCombination_Srss, kCombination_AbsSum };

// error types
enum {kErrorNone, kInvalidInput, kErrorProcessingElementStiffness, kNoMassDefined, kErrorConvertingAreaLoadsToLineLoads,
      kTooManyDigitsLostDuringFactorization, kSolverError, kAbnormalSolverTermination, kMustRunResponseSpetrumAnalysis,
      kMustRunFrequencyAnalysisFirst, kFileAccessError, kLicenseError, kInvalidLoadCombinationIndex, kInvalidModeIndex,
      kNoResultAvailable, kUnknownError };

```

The following lists a few common constants

```

const int LABEL_SIZE = 128;

const int MAX_SPECTRUM_DATA_POINTS = 128;

```

The following lists a few result data structures

```

struct CGISOLVERBLAZE_API cgiResult6Val{
    int iId;          // node or element number
    double fVal[6];  // values in six degrees of freedom directions
};

struct CGISOLVERBLAZE_API cgiMultiDofConstraintForce {
    int iId;          // constraint id
    int iDof[2];     // first and second constrained degrees of freedom directions
    int iNode[2];    // first and second constrained nodes
    double fVal[2];  // constrained forces or moments depending on the constrained DOFs
};

```

```

};

struct CGISOLVERBLAZE_API cgiVMD {
    double fDist; // ratio
    double fVMD[6]; // forces and moments in six DOF directions
    double fDefl[2]; // deflections in member local y and z directions
};

struct CGISOLVERBLAZE_API cgiBeamEndVMD {
    int iId; // element number
    cgiVMD vmd[2]; // forces and moments at two ends of a member
};

struct CGISOLVERBLAZE_API cgiShellStress {
    int iId; // shell element number
    // stresses @ center + 4-nodes
    double fStress_m[5][3]; // Fxx, Fyy, Fxy (at the center and four nodes of the shell)
    double fStress_b[5][5]; // Mxx, Myy, Mxy, Fv1, Fv2 (at the center and four nodes of the shell)
    double fStress[2][5][6]; // Top & Bottom, Sxx, Syy, Szz, Sxy, Syz, Sxz
    // (combined membrane and bending stresses at the center and four nodes of the shell)
    double fForce_m[8]; // nodal force resultants for membrane, in local coordinate
    // (Fx, Fy component at four nodes of the shell)
    double fForce_b[12]; // nodal moment and force resultants for bending, in local coordinate
    // (Mx, My, Fz components at four nodes of the shell)
};

struct CGISOLVERBLAZE_API cgiBrickStress {
    int iId; // brick element element number
    double fStress[9][6]; // Fxx, Fyy, Fzz, Fxy, Fyz, Fxz at element center + eight nodes
};

// forces, moments and deflection for a member
struct CGISOLVERBLAZE_API cgiBeamShearMomentDeflection {
    cgiBeamShearMomentDeflection(int id = -1);
    ~cgiBeamShearMomentDeflection();
    cgiBeamShearMomentDeflection(const cgiBeamShearMomentDeflection& other);
    cgiBeamShearMomentDeflection& operator=(const cgiBeamShearMomentDeflection& other);
    void setId(int nIdentity);
    int getId()const;
    void getBeamVMDValues(cgiVMD*& buffer, int& count)const;
};

```

```

// currently, only maximum of 8 points are used for regular sections (rectangular, round, tee, wide-flange, channel, rectangular
tube, pipe, single angle)
const int MAX_SECTION_POINTS = 16;

struct CGISOLVERBLAZE_API cgiSectionStress
{
    cgiSectionStress() {
        fDist = 0.0;
        nStressPoints = 0;
        memset(&normalStress[0], gtcharNull, sizeof(double) * MAX_SECTION_POINTS);
        fMaxCompressiveStress = 0;
        fMaxTensileStress = 0;
    }
    static const TCHAR* name() { return _T("cgiSectionStress"); }

    bool operator<(const cgiSectionStress& other) const {
        return fDist < other.fDist;
    }

    double fDist; // ratio
    int nStressPoints;
    double normalStress[MAX_SECTION_POINTS];
    double fMaxCompressiveStress; // minimum negative or 0
    double fMaxTensileStress; // maximum positive or 0
};

struct CGISOLVERBLAZE_API cgiBeamStress
{
    cgiBeamStress(int id = -1);
    ~cgiBeamStress();
    cgiBeamStress(const cgiBeamStress& other);
    cgiBeamStress& operator=(const cgiBeamStress& other);
    static const TCHAR* name() { return _T("cgiBeamStress"); }
    void setId(int nIdentity);
    int getId()const;
    bool operator==(const cgiBeamStress& other) const;
    bool operator<(const cgiBeamStress& other) const;
    void getBeamSectionStresses(cgiSectionStress*& buffer, int& count)const;

    // the following functions are for internal uses only
    const jxBmStress* getBeamStresses()const;
    void setBeamStresses(const jxBmStress* pBeamStress);
};

```

```

private:
    void rebuildCachedBeamSectionStresses();

private:
    jxBmStress* m_pBeamStress;
    cgiSectionStress* m_pBeamSectionStresses;
    int m_nBeamSectionStressCount;
};

// results for a load combination
struct CGISOLVERBLAZE_API cgiCombinationResult {
    cgiResult6Val* m_vNdDisp;           // nodal displacements
    int m_nNdDispCount;
    cgiResult6Val* m_vSuptReact;       // support reactions
    int m_nSuptReactCount;
    cgiMultiDofConstraintForce* m_vMultiDofConstraintForce; // multi-dof-constraint reaction
    int m_nMultiDofConstraintForceCount;
    cgiResult6Val* m_vSpringReact_Node; // nodal spring reactions
    int m_nSpringReact_NodeCount;
    cgiResult6Val* m_vSpringReact_Coupled; // coupled spring reactions
    int m_nSpringReact_CoupledCount;
    cgiResult6Val* m_vSpringReact_Beam; // line spring reactions
    int m_nSpringReact_BeamCount;
    cgiResult6Val* m_vSpringReact_Shell4; // surface spring reactions
    int m_nSpringReact_Shell4Count;
    cgiShellStress* m_vShell4Stress; // shell stresses
    int m_nShell4StressCount;
    cgiBrickStress* m_vBrickStress; // brick stresses
    int m_nBrickStressCount;
    cgiFixedEndMoment* m_vFEM; // member fixed end forces/moments
    int m_nFEMCount;
    cgiBeamEndVMD* m_vBmEndVMD; // vmd (force, moment and deflection) at member ends
    int m_nBmEndVMDCount;
    cgiBeamShearMomentDeflection* m_vBmVMD; // vmd (force, moment and deflection) at segmental points along member
    int m_nBmVMDCount;
};

// eigen result for one mode
struct CGISOLVERBLAZE_API cgiEigenValueVectorResult {
    double m_fEigen; // eigen value ( = circular frequency * circular frequency)
    double m_fErrorMeasure; // error measures on the eigen value
    cgiResult6Val* m_vEigenVector; // eigen vector
    int m_nEigenVectorCount;
};

```

```

};

// response spectrum result for one mode
struct CGISOLVERBLAZE_API cgiResponseSpectrumResult {
    double m_fEigen; // eigen value ( = circular frequency * circular frequency)
    double m_fParticipation[3]; // participation factors in global X, Y and Z directions
    cgiResult6Val* m_vModalDisplacement[3]; // modal displacement in global X, Y and Z directions
    int m_nModalDisplacementCount[3];
    cgiResult6Val* m_vInertialForce[3]; // nodal inertia forces in global X, Y and Z directions
    int m_nInertialForceCount[3];
};

struct CGISOLVERBLAZE_API cgiEnvelopeValue6
{
    int iId;
    double fMax[6];
    double fMin[6];
    int iMaxComb[6];
    int iMinComb[6];
};

struct CGISOLVERBLAZE_API cgiEnvelopeVMD
{
    double fDist; // ratio
    double fMaxVMD[6];
    double fMinVMD[6];
    int iMaxComb[6];
    int iMinComb[6];
};

struct CGISOLVERBLAZE_API cgiEnvelopeBmVMD
{
    cgiEnvelopeBmVMD(int id = -1);
    ~cgiEnvelopeBmVMD();
    cgiEnvelopeBmVMD(const cgiEnvelopeBmVMD& other);
    cgiEnvelopeBmVMD& operator=(const cgiEnvelopeBmVMD& other);
    static const TCHAR* name() { return _T("cgiEnvelopeBmVMD"); }
    void setId(int nIdentity);
    int getId()const;
    bool operator==(const cgiEnvelopeBmVMD& other) const;
    bool operator<(const cgiEnvelopeBmVMD& other) const;
};

```

```

    void getEnvelopeBmVMDValues(cgiEnvelopeVMD*& buffer, int& count)const;
};

struct CGISOLVERBLAZE_API cgiNodalResultant {
    double fVal[6];
};

```

There are a few utility functions within the interface. For example, you can calculate a beam local angle such that the beam local z axis is perpendicular to the plane formed by the two member end points and a 3rd point.

```

cgiPoint pt1(-141.53, -372.022, 210.897);
cgiPoint pt2(-116.462, -510.254, 152.296);
cgiPoint pt3(0, 0, 0);

double fAngleInRadian = pStructure->getBeamLocalAngleByThirdPoint(pt1, pt2, pt3);

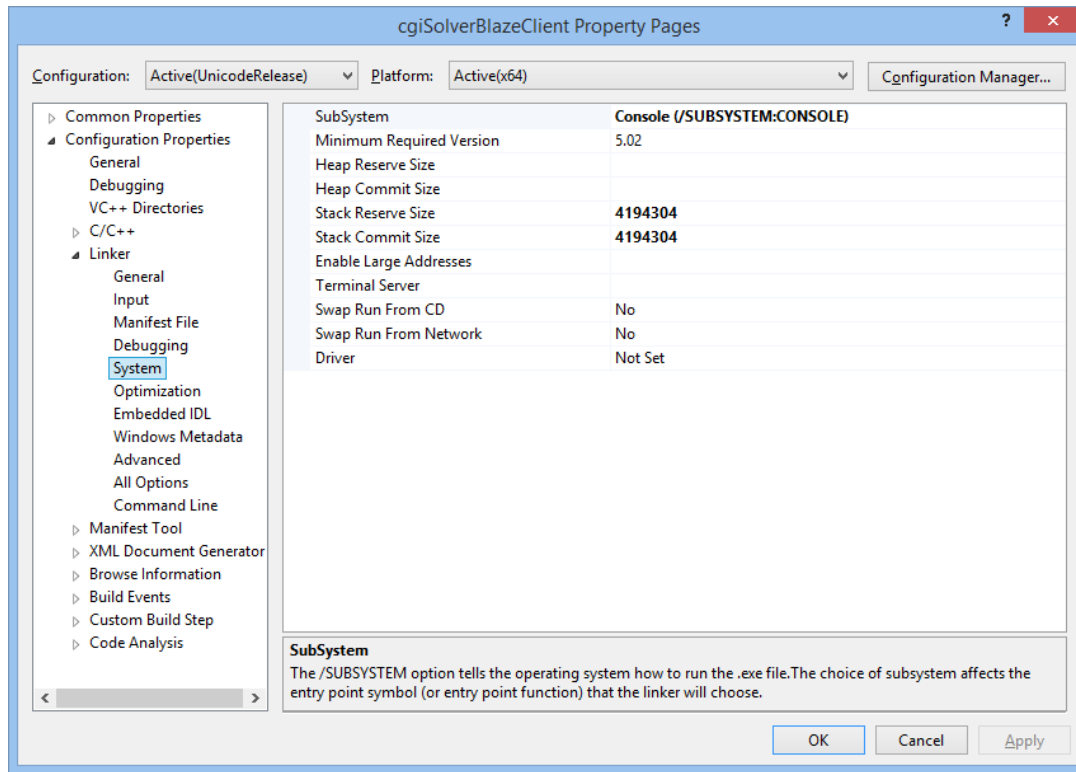
```

SolverBlaze computes beam stresses for regular sections (rectangular, round, tee, wide-flange, channel, rectangular tube, pipe, and single-angle), AISC steel shape sections, and NDS wood sections. The program uses a naming convention for regular sections in which a section-type prefix is followed by dimensions separated by the letter “x”. Examples include: **Rect5.5x7.5**, **Round6**, **WideF10x12x2x1**, **Tee12x10x2x1**, **Chan8x12x2x1**, **Tube6x8x1**, **Pipe7x1**, and **Angle6x8x1**.

AISC sections must follow the AISC manual labeling such as W36X330, C15X33.9 etc. In order to calculate beam stresses for AISC sections, the aisc section database files (aisc16.idx and aisc16.tbl) must be present in the executable folder.

NDS wood sections must follow the convention used in the program such as SawnLum2x5, GluLamW2.5x21, GluLamS2.5x23.375 etc. In order to calculate beam stresses for NDS wood sections, the NDS section database file (nds.tbl) must be present in the executable folder.

Four versions of `cgiSolverBlaze.lib` and `cgiSolverBlaze.dll` are provided for your configuration needs: Unicode or Multi-Bytes, 64x, ARM64 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 (`libiomp5md.dll`, `cgiSolverBlaze.dll` and `double128Proj.dll`) to your executable directory. If your project is configured for 64-bit CPU, you 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.



We will illustrate the use of the SolverBlaze API by using the following structural models taken from example 4 and B-01 (Plate Patch Test) of Real3D Verification Manual, which is freely available for download from <http://www.cg-inc.com/download/REAL3DVerifications.pdf>

It is highly recommended that you study the “Technical Issues” in the Real3D program manual, which is freely available from here <http://www.cg-inc.com/download/REAL3DManual.pdf>. You will get a good understanding of the theoretical background on which SolverBlaze is based.

Example Model One (refer to the sample file verify-example4.cpp included in the library)

The following portal frame has a span of 60 ft and a column height of 24 ft. The beam is vertically loaded with 60 kips placed at 20 ft from the left end of the beam. The right column is vertically loaded with 120 kips. A horizontal load of 6 kips is applied at the joint of the beam and the left column. Each column is modeled with 2 members. The beam is modeled with a single frame element.

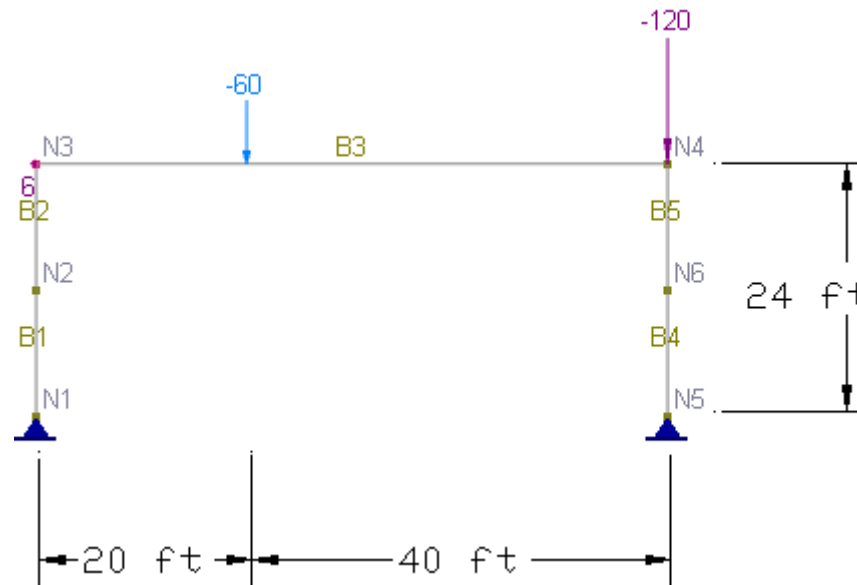
Columns: W10x45, $A = 13.3 \text{ in}^2$, $I_z = 248 \text{ in}^4$

Beam: W27x84, $A = 24.8 \text{ in}^2$, $I_z = 2850 \text{ in}^4$

Material: $E = 2.9e7 \text{ psi}$, $\nu = 0.3$

Perform analysis for the following two cases:

- First order (Linear) elastic analysis
- Second order (P-Delta) elastic analysis



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

```
#include "_cgiIStructure.h"
#include <iostream>
#include <iomanip>
#include <vector>
using namespace std;
using namespace cgiSolverBlazeNamespace;

#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;
}

static int MlkProgress( int* ithr, int* step, char* stage, int len )
{
    COUT << "MLK thread = " << *ithr << "; step = " << *step << "; stage = " << stage << endl;
    return 0;
}

void verify_example4()
{
    // create a structural model
    cgiIStructure* pStructure = CreateStructure();

    TCHAR szTestPath[256];
    TCHAR szInputFileName[256];
    pStructure->getDefaultTestPath(szTestPath, 256);
    _stprintf(szInputFileName, _T("%s\\tests\\newModels\\%s"), szTestPath, _T("Verify-Example4.r3a"));

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

    pStructure->setModelType(kModel_Frame2D); // a 2D Frame

    // 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();

// define materials
vector<cgiMaterial> vMat;
cgiMaterial mat;
mat.setId(1); // material 1 id, to be referred later
mat.setProperties(_T("Default"), 29000, 0.3, 450); // material label, young's modulus, poisson ratio, weight density
vMat.push_back(mat);

// assign all materials to the structural model
pStructure->setMaterials(&vMat[0], vMat.size());

// define sections
vector<cgiSection> vSect;
cgiSection sect;
sect.setId(1);
vSect.push_back(sect); // default section, we are not going to use it
sect.setId(2); // section 2 id
_tcscpy(sect.szLabel, _T("W27X84")); // section label, can not contain spaces
sect.fVal[cgiSection::A] = 24.8; // sectional area: in^2
sect.fVal[cgiSection::Ayy] = 12.282; // major shear area: in^2
sect.fVal[cgiSection::Azz] = 12.7488; // minor shear area: in^2
sect.fVal[cgiSection::Izz] = 2850; // major moment of inertia: in^4
sect.fVal[cgiSection::Iyy] = 106; // minor moment of inertia: in^4
sect.fVal[cgiSection::J] = 2.81; // rotational moment of inertia: in^4
// or you can properties like this
//sect.setProperties(_T("W27X84"), 24.8, 12.282, 12.7488, 2850, 106, 2.81);
vSect.push_back(sect);
sect.setId(3); // section 3 id
sect.setProperties(_T("W10X45"), 13.3, 3.535, 9.9448, 248, 53.4, 1.51); // another way to set section properties
vSect.push_back(sect);

// assign all sections to the structural model
pStructure->setSections(&vSect[0], vSect.size());

// define nodes
vector<cgiNode> vNd;
cgiNode nd;
nd.setId(1); // nodal id
nd.setCoordinates(0, 0, 0); // nodal coordinates
vNd.push_back(nd);
nd.setId(2);
nd.setCoordinates(0, 12, 0);
vNd.push_back(nd);
nd.setId(3);
nd.setCoordinates(0, 24, 0);
vNd.push_back(nd);
nd.setId(5);
nd.setCoordinates(60, 0, 0);

```

```

vNd.push_back(nd);
nd.setId(6);
nd.setCoordinates(60, 12, 0);
vNd.push_back(nd);
nd.setId(4);
nd.setCoordinates(60, 24, 0);
vNd.push_back(nd);

// assign all nodes to the structural model
pStructure->setNodes(&vNd[0], vNd.size());

// define beams
vector<cgiBeam> vBm;
cgiBeam bm;
bm.setId(1); // beam id
bm.setNodes(1, 2); // begin and end node ids of the beam
bm.setProperties(1, 3, 0); // material id, section id, beam angle in radian
vBm.push_back(bm);
bm.setId(2);
bm.setNodes(2, 3);
bm.setProperties(1, 3, 0);
vBm.push_back(bm);
bm.setId(3);
bm.setNodes(3, 4);
bm.setProperties(1, 2, 0);
vBm.push_back(bm);
bm.setId(5);
bm.setNodes(6, 4);
bm.setProperties(1, 3, 0);
vBm.push_back(bm);
bm.setId(4);
bm.setNodes(5, 6);
bm.setProperties(1, 3, 0);
vBm.push_back(bm);

// assign all beams to the structural model
pStructure->setBeams(&vBm[0], vBm.size());

// define supports
vector<cgiSupport> vSupt;
cgiSupport supt;
supt.setId(1); // nodal id that this support is on
// six DOFs, 1 for supported, 0 for free; forced settlements and rotations
supt.setSupportDOFs(_T("111000"), 0, 0, 0, 0, 0, 0);
vSupt.push_back(supt);
supt.setId(5);
supt.setSupportDOFs(_T("111000")); // forced settlements and rotations default to 0s
vSupt.push_back(supt);

// assign all supports to the structural model
pStructure->setSupports(&vSupt[0], vSupt.size());

```

```

// define load cases
vector<cgiloadCase> vLoadCase;
cgiloadCase loadcase;
loadcase.setId(1); // load case id
// load case label; type such as DEAD, LIVE etc; report on this load case
loadcase.setLoadCase(_T("Default"), _T("DEAD"), TRUE);
vLoadCase.push_back(loadcase);

// assign all load cases to the structural model
pStructure->setLoadCases(&vLoadCase[0], vLoadCase.size());

// define load combinations
vector<cgiloadCombination> vLoadComb;
cgiloadCombination loadcomb; // load combination
loadcomb.clearLoadCombItem(); // make sure we start clean with the first load combination
// load combination label, linear combination (no p-delta effect), want report on this combination
loadcomb.setLoadComb(_T("Linear"), FALSE, TRUE);
loadcomb.addLoadCombItem( _T("Default"), 1.0); // add load case and factor to this load combination
vLoadComb.push_back(loadcomb);
loadcomb.clearLoadCombItem(); // make sure we start clean with the second load combination
// load combination label, consider p-delta, want report on this combination
loadcomb.setLoadComb(_T("P-Delta"), TRUE, TRUE);
loadcomb.addLoadCombItem( _T("Default"), 1.0); // add load case and factor to this load combination
vLoadComb.push_back(loadcomb);

// assign all load combinations to the structural model
pStructure->setLoadCombinations(&vLoadComb[0], vLoadComb.size());

// each case load corresponds to each load case
// each case load includes nodal loads, point loads, line loads etc for this load case
vector<cgiloadCaseLoad> vCaseLoad;
cgiloadCaseLoad caseload;
cgiloadNodalLoad ndload;
ndload.setLoad(4, -120.0, Y);
caseload.addNodalLoad(ndload);
ndload.setLoad(3, 6.0, X); // node id, load magnitude and direction
// assign this nodal load to the first caseload
caseload.addNodalLoad(ndload);
cgiloadPointLoad ptload;
// member id, load magnitude, distance from member start in percentage/100, load direction
ptload.setLoad(3, GLOBAL, -60.0, 0.333333, Y);
caseload.addPointLoad(ptload);
// assign this point load to the first caseload
vCaseLoad.push_back(caseload);

// assign all caseloads to the structural model
pStructure->setCaseLoads(&vCaseLoad[0], vCaseLoad.size());

// set report options
cgiloadReportOptions reportOptions;
reportOptions.SelectAll();
reportOptions.bHTML = FALSE; // we do not want html format, just plain text

```

```

pStructure->setReportOptions(reportOptions);

// set analysis options
bool bConsiderBeamShearDeformation = FALSE;
int nMaximumPDeltaIterations = 10;
double fPDeltaToleranceInPercentage = 0.5; // in percentage
int nNumberOfSegmentsForBeamOutput = 20;
bool bUseThinPlate = false;
bool bUseCompatibleModes = true;
int nUseAverageStressMode = 1;
pStructure->setAnalysisOptions(bConsiderBeamShearDeformation, nMaximumPDeltaIterations, fPDeltaToleranceInPercentage,
                             nNumberOfSegmentsForBeamOutput, bUseThinPlate, bUseCompatibleModes, nUseAverageStressMode);

//pStructure->setSolver(kSolverSparse64);
//pStructure->setSolver(kSolver128);

// save the data to a file for possible later use
bool b2 = pStructure->saveDocument(szInputFileName);

// run static analysis
bool bRun = pStructure->runStaticAnalysis();
if(!bRun)
{
    COUT << _T("Error running static analysis... ") << szInputFileName << endl;
    return;
}

// report
pStructure->setListMessageFunction(0); // do not list message
if(!pStructure->runReport())
{// report will be saved in ttestt.htm file
    COUT << _T("Error running report... ") << szInputFileName << endl;
    return;
}

// extract results
cgiCombinationResult* vCombResult = NULL;
int nCombResultCount = 0;
pStructure->getStaticResults(vCombResult, nCombResultCount);
// list displacements for all load combinations
TCHAR szTranslationalDisplacementUnit[LABEL_SIZE];
TCHAR szRotationalDisplacementUnit[LABEL_SIZE];
pStructure->getUnit(szTranslationalDisplacementUnit, DISPLACEMENT_TRANS);
pStructure->getUnit(szRotationalDisplacementUnit, DISPLACEMENT_ROTATE);
COUT << _T("-----") << endl;
COUT << _T("----- Verifying Example 4 -----") << endl;

COUT << endl << _T("Nodal Displacements. Units: ") << szTranslationalDisplacementUnit << _T(", ") << szRotationalDisplacementUnit << endl;
for(int iComb = 0; iComb < nCombResultCount; iComb++)
{
    COUT << _T("Load combination ") << iComb + 1 << ": " << vLoadComb[iComb].getLabel() << endl;

    int nDisps = vCombResult[iComb].m_nNdDispCount;

```

```

    COUT << setprecision(4) << fixed;
    for(int i = 0; i < nDisps; i++) {
        const cgiResult6Val& disp = vCombResult[iComb].m_vNdDisp[i];
        COUT << _T("Node- ") << disp.iId << _T(": ")
            << disp.fVal[X] << "\t" << disp.fVal[Y] << "\t" << disp.fVal[Z] << "\t"
            << disp.fVal[OX] << "\t" << disp.fVal[OY] << "\t" << disp.fVal[OZ] << endl;
    }
    COUT << endl;
}

// list internal forces and moments at beam ends for all load combinations
TCHAR szForceUnit[LABEL_SIZE];
TCHAR szMomentUnit[LABEL_SIZE];
pStructure->getUnit(szForceUnit, FORCE);
pStructure->getUnit(szMomentUnit, MOMENT);
COUT << endl << _T("Beam End Forces and Moments. Units: ") << szForceUnit << _T(", ") << szMomentUnit << endl;
for(int iComb = 0; iComb < nCombResultCount; iComb++)
{
    COUT << _T("Load combination ") << iComb + 1 << ": " << vLoadComb[iComb].getLabel() << endl;

    int nBeams = vCombResult[iComb].m_nBmVMDCount;
    for(int i = 0; i < nBeams; i++) {
        const cgiBeamShearMomentDeflection& bmVMD = vCombResult[iComb].m_vBmVMD[i];

        cgiVMD* buffer = NULL;
        int count = 0;
        bmVMD.getBeamVMDValues(buffer, count);

        int iSeg[2];
        iSeg[0] = 0;
        iSeg[1] = (int)count - 1;
        COUT << setprecision(4) << fixed;
        for(int k = 0; k < 2; k++)
        {
            const cgiVMD& vmd = buffer[iSeg[k]];
            TCHAR* szEnd = (k==0)? _T("Start") : _T("End");
            COUT << _T("Beam ") << bmVMD.getId() << _T(" ") << szEnd << _T(": ")
                << vmd.fVMD[X] << "\t" << vmd.fVMD[Y] << "\t" << vmd.fVMD[Z] << "\t"
                << vmd.fVMD[OX] << "\t" << vmd.fVMD[OY] << "\t" << vmd.fVMD[OZ] << endl;
        } // for(int k = 0; k < bmVMD.vVMD.size(); k++)

    } // for(int i = 0; i < nBeams; i++) {
    COUT << endl;
}

pStructure->deleteMemoryArray(vCombResult);
}

```

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

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

```
cgiIStructure* pStructure = CreateStructure();
```

You can supply three 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);
typedef int (*fnMKLPROGRESS)( int* ithr, int* step, char* stage, int len );

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

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

static int MlkProgress( int* ithr, int* step, char* stage, int len )
{
    COUT << "MLK thread = " << *ithr << "; step = " << *step << "; stage = " << stage << endl;
    return 0;
}

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

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 SolverBlaze currently may emit.

```
enum {kErrorNone, kInvalidInput, kErrorProcessingElementStiffness, kNoMassDefined, kErrorConvertingAreaLoadsToLineLoads,
      kTooManyDigitsLostDuringFactorization, kSolverError, kAbnormalSolverTermination, kMustRunResponseSpectrumAnalysis,
      kMustRunFrequencyAnalysisFirst, kFileAccessError, kLicenseError, kInvalidLoadCombinationIndex, kInvalidModeIndex,
      kNoResultAvailable, kUnknownError};
```

The last error code is cleared (set to `kErrorNone`) at the beginning of each solution. Next you can set the model type as defined in the `_cgiDefines.h`. For example:

```
pStructure->setModelType(kModel_Frame2D); // a 2D Frame
```

Other model types can be found by examining the following enumerations defined in `_cgiDefines.h`

```
enum {kModel_Frame3D, kModel_Frame2D, kModel_Truss3D, kModel_Truss2D,  
      kModel_PlateBending, kModel_PlaneStress, kModel_Brick, kModel_Grillage, kModel_End};
```

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();
```

The input includes definitions for materials, sections, nodes, elements, load cases, load combinations, loads (nodal, point, line etc.) in each load case. You can also set different analysis and report options.

Each material is defined by specifying a unique material id (integer number), a material's label, young's modulus, Poisson ratio and weight density. **The material label must be less than 127 characters long and must not contain spaces. This limitation also applies in other labels used in sections, load cases, load combinations etc.** All materials must then be assigned to the structural model by calling `cgilStructure::setMaterials()`

```
// define materials
```

```

vector<cgiMaterial> vMat;
cgiMaterial mat;
mat.setId(1); // material 1 id, to be referred later
mat.setProperties(_T("Default"), 29000, 0.3, 450); // material label, young's modulus, poisson ratio, weight density
vMat.push_back(mat);

// assign all materials to the structural model
pStructure->setMaterials(&vMat[0], vMat.size());

```

Each beam section is defined by specifying a unique section id (integer number), a section label, sectional area, major and minor shear area, major and minor moment of inertia and rotational moment of inertia. **The section label must be less than 127 characters long and must not contain spaces.** All sections must then be assigned to the structural model by calling `cgilStructure::setSections()`. You can set a section to be rigid link by calling `cgiSection::setRigidLink()`.

```

// define sections
vector<cgiSection> vSect;
cgiSection sect;
sect.setId(1);
vSect.push_back(sect); // default section, we are not going to use it
sect.setId(2); // section 2 id
_tcscpy(sect.szLabel, _T("W27X84")); // section label, cannot contain spaces
sect.fVal[cgiSection::A] = 24.8; // sectional area: in^2
sect.fVal[cgiSection::Ayy] = 12.282; // major shear area: in^2
sect.fVal[cgiSection::Azz] = 12.7488; // minor shear area: in^2
sect.fVal[cgiSection::Izz] = 2850; // major moment of inertia: in^4
sect.fVal[cgiSection::Iyy] = 106; // minor moment of inertia: in^4
sect.fVal[cgiSection::J] = 2.81; // rotational moment of inertia: in^4
// or you can properties like this
//sect.setProperties(_T("W27X84"), 24.8, 12.282, 12.7488, 2850, 106, 2.81);
vSect.push_back(sect);
sect.setId(3); // section 3 id
sect.setProperties(_T("W10X45"), 13.3, 3.535, 9.9448, 248, 53.4, 1.51); // another way to set section properties
vSect.push_back(sect);

// assign all sections to the structural model
pStructure->setSections(&vSect[0], vSect.size());

```

Although it is generally not needed, the interface also allows you to retrieve sections. You should delete the memory using the interface function `deleteMemoryArray()` as shown below. **DO NOT delete the memory using `delete` or `delete[]` directly.**

```

cgiSection* pSection = NULL;

```

```

int sectionCount = 0;
pStructure->getSections(pSection, sectionCount);
// ...
pStructure->deleteMemoryArray(pSection);

```

Next each node is defined by specifying a unique node id (integer number), x, y and z coordinates. All nodes must then be assigned to the structural model by calling `cgiIStructure:: setNodes()`

```

// define nodes
vector<cgiNode> vNd;
cgiNode nd;
nd.setId(1); // nodal id
nd.setCoordinates(0, 0, 0); // nodal coordinates
vNd.push_back(nd);
nd.setId(2);
nd.setCoordinates(0, 12, 0);
vNd.push_back(nd);
nd.setId(3);
nd.setCoordinates(0, 24, 0);
vNd.push_back(nd);
nd.setId(5);
nd.setCoordinates(60, 0, 0);
vNd.push_back(nd);
nd.setId(6);
nd.setCoordinates(60, 12, 0);
vNd.push_back(nd);
nd.setId(4);
nd.setCoordinates(60, 24, 0);
vNd.push_back(nd);

// assign all nodes to the structural model
pStructure->setNodes(&vNd[0], vNd.size());

```

Next beam (or frame member) is defined by specifying a unique beam id (integer number), start and end node ids, a material id, a section id and a beam local angle in radian. By default, a beam behavior is linear. You can set a beam to be tension only or compression only by calling `cgiBeam::setNonlinear()`. You also have the option to modify beam stiffness in Iz, Iy, J, A, Ay, and Az directions by calling `cgiBeam::setStiffnessModificationFactor()`. All beams must then be assigned to the structural model by calling `cgiIStructure:: setBeams()`.

```

// define beams
vector<cgiBeam> vBm;
cgiBeam bm;

```

```

bm.setId(1); // beam id
bm.setNodes(1, 2); // begin and end node ids of the beam
bm.setProperties(1, 3, 0); // material id, section id, beam angle in radian
vBm.push_back(bm);
bm.setId(2);
bm.setNodes(2, 3);
bm.setProperties(1, 3, 0);
vBm.push_back(bm);
bm.setId(3);
bm.setNodes(3, 4);
bm.setProperties(1, 2, 0);
vBm.push_back(bm);
bm.setId(5);
bm.setNodes(6, 4);
bm.setProperties(1, 3, 0);
vBm.push_back(bm);
bm.setId(4);
bm.setNodes(5, 6);
bm.setProperties(1, 3, 0);
vBm.push_back(bm);

// assign all beams to the structural model
pStructure->setBeams(&vBm[0], vBm.size());

```

Next support (or constraint) is defined by specifying a unique node id (integer number), six constraints for each of the six global degrees of freedom, six forced translations and rotations in each of the six global degrees of freedom. ‘1’ represents a constrained DOF while ‘0’ represents a free DOF. All supports must then be assigned to the structural model by calling `cgiStructure:: setSupports ()`

```

// define supports
vector<cgiSupport> vSupt;
cgiSupport supt;
supt.setId(1); // nodal id that this support is on
// six DOFs, 1 for supported, 0 for free; forced settlements and rotations
supt.setSupportDOFs(_T("111000"), 0, 0, 0, 0, 0, 0);
vSupt.push_back(supt);
supt.setId(5);
supt.setSupportDOFs(_T("111000")); // forced settlements and rotations default to 0s
vSupt.push_back(supt);

// assign all supports to the structural model
pStructure->setSupports(&vSupt[0], vSupt.size());

```

The following is the support definition structure defined in the `_cgiDefines.h`

```
struct CGISOLVERBLAZE_API cgiSupport
{
    // support flag must be a 6-character zero terminated string. 0 - free, 1-support, 2-suppressed
    // e.g. "110001" is a support with Dx, Dy and Doz supported, the rest of DOFs are free
    // you can optionally specified forced settlement or rotation on the supported DOFs
    int iId;
    TCHAR szFlag[6 + 1];
    double fDisp[6];
};
```

Please note that moment releases and springs follow similar approaches as supports. For example, the following is the spring definition structure defined in the `_cgiDefines.h` file:

```
struct CGISOLVERBLAZE_API cgiSpring
{
    enum {kLinear, kCompression, kTension};

    // spring flag must be a 6-character zero terminated string. 0 - linear, 1 - compression only, 2 - tension only
    // for line and surface springs, only translational DOFs are supported
    // for nodal springs, all six DOFs are supported.
    // spring coefficients fKx, fKy etc. can be zero
    int iId;
    TCHAR szFlag[6 + 1];
    double fK[6];
};
```

Although not needed in this 2D example, SolverBlaze allows you to define rigid diaphragms in a 3D building by calling `cgiIStructure::setDiaphragms()`.

Next load case is defined by specifying a unique load case id (integer number), a load case label, a load case type and whether to report the load case or not. **The load case label must be less than 127 characters long and must not contain spaces.** All load cases must then be assigned to the structural model by calling `cgiIStructure::setLoadCases()`. The types of load cases are defined in `_cgiDefines.h`

```
const TCHAR gszTypeDes1[kCASETYPE_END][LABEL_SIZE] = {_T("Dead"), _T("Live"), _T("RoofLive"),
                                                       _T("Snow"), _T("Wind"), _T("Earthquake"), _T("Rain")};

// define load cases
vector<cgiLoadCase> vLoadCase;
cgiLoadCase loadcase;
```

```

loadcase.setId(1); // load case id
// load case label; type such as DEAD, LIVE etc; report on this load case
loadcase.setLoadCase(_T("Default"), _T("DEAD"), TRUE);
vLoadCase.push_back(loadcase);

// assign all load cases to the structural model
pStructure->setLoadCases(&vLoadCase[0], vLoadCase.size());

```

Next load combination is defined by specifying a load combination label, p-delta flag and whether to report the load case or not. **The load combination label must be less than 127 characters long and must not contain spaces.** Each load case and its factor is specified by calling `cgiLoadComb::addLoadCombItem()`. All load combinations must then be assigned to the structural model by calling `cgiStructure::setLoadCombinations()`.

```

// define load combinations
vector<cgiLoadCombination> vLoadComb;
cgiLoadCombination loadcomb; // load combination
loadcomb.clearLoadCombItem(); // make sure we start clean with the first load combination
// load combination label, linear combination (no p-delta effect), want report on this combination
loadcomb.setLoadComb(_T("Linear"), FALSE, TRUE);
loadcomb.addLoadCombItem( _T("Default"), 1.0); // add load case and factor to this load combination
vLoadComb.push_back(loadcomb);
loadcomb.clearLoadCombItem(); // make sure we start clean with the second load combination
// load combination label, consider p-delta, want report on this combination
loadcomb.setLoadComb(_T("P-Delta"), TRUE, TRUE);
loadcomb.addLoadCombItem( _T("Default"), 1.0); // add load case and factor to this load combination
vLoadComb.push_back(loadcomb);

// assign all load combinations to the structural model
pStructure->setLoadCombinations(&vLoadComb[0], vLoadComb.size());

```

Each caseload corresponds to all loads for one load case. These loads may be nodal loads, point loads on members, line loads on members etc. A nodal load is defined by specifying a node id, load magnitude and load direction (X, Y, Z, OX, OY or OZ). A nodal load is added to a caseload by calling `cgiCaseLoad::addNodalLoad()`. A point load is defined by a beam id, local or global load system, load magnitude, a distance percentage from member start and load direction (X, Y, Z, OX, OY or OZ). A point load is added to a caseload by calling `cgiCaseLoad::addPointLoad()`. All caseloads must then be assigned to the structural model by calling `cgiStructure::setCaseLoads()`.

```

// each case load corresponds to each load case
// each case load includes nodal loads, point loads, line loads etc for this load case
vector<cgiLoadCaseLoad> vCaseLoad;
cgiLoadCaseLoad caseload;

```

```

cgiNodalLoad ndload;
ndload.setLoad(4, -120.0, Y);
caseload.addNodalLoad(ndload);
ndload.setLoad(3, 6.0, X); // node id, load magnitude and direction
// assign this nodal load to the first caseload
caseload.addNodalLoad(ndload);
cgiPointLoad ptload;
// member id, load magnitude, distance from member start in percentage/100, load direction
ptload.setLoad(3, GLOBAL, -60.0, 0.333333, Y);
caseload.addPointLoad(ptload);
// assign this point load to the first caseload
vCaseLoad.push_back(caseload);

// assign all caseloads to the structural model
pStructure->setCaseLoads(&vCaseLoad[0], vCaseLoad.size());

```

Note: The number of caseloads must equal the number of load cases. In the example, `vCaseLoad.size() == vLoadCase.size()`.

Report options may be specified setting multiple flags in the `cgiReportOptions` defined in `_cgiDefines.h`. The most comprehensive report options can be set by calling `cgiReportOptions::SelectAll()`. Report options must be set to the structural model by calling `cgiIStructure::setReportOptions()`

```

// set report options
cgiReportOptions reportOptions;
reportOptions.SelectAll();
reportOptions.bHTML = FALSE; // we do not want html format, just plain text
pStructure->setReportOptions(reportOptions);

```

The last important step is to set the analysis options. These options include whether you want to consider shear deformations from the beam members, p-delta tolerance and iterations, the number of segments to output beam results etc. The analysis options must be assigned the structural model by calling `cgiIStructure::setAnalysisOptions()`. *If the model contains beam or shell elements with stiffness modifications, then you must call `cgiIStructure::setApplyStiffnessModification()` in order for the stiffness modifications to take effect.*

You can also specify which solver you would like to use by calling `cgiIStructure::setSolver()`. Solver types are defined in `_cgiDefines.h`

```

enum {kSolver64=0, kSolver128, kSolverSparse64};

// set analysis options
bool bConsiderBeamShearDeformation = FALSE;
int nMaximumPDeltaIterations = 10;

```

```

double fPDeltaToleranceInPercentage = 0.5; // in percentage
int nNumberOfSegmentsForBeamOutput = 20;
bool bUseThinPlate = false;
bool bUseCompatibleModes = true;
int nUseAverageStressMode = 1;
pStructure->setAnalysisOptions(bConsiderBeamShearDeformation, nMaximumPDeltaIterations, fPDeltaToleranceInPercentage,
                             nNumberOfSegmentsForBeamOutput, bUseThinPlate, bUseCompatibleModes, nUseAverageStressMode);
pStructure->setSolver(kSolverSparse64);
//pStructure->setSolver(kSolver128);

```

Before you perform structural analysis, you can save the input to a file by calling `cgilStructure:: saveDocument()`. This file can then be opened, visualized or analyzed by Real3D. This can be very useful to verify the correctness of the model input and its results. The actual structural analysis is performed by calling `cgilStructure:: runStaticAnalysis()`. The program checks for input errors (such as duplicate nodes) prior to performing analysis or report. The error messages are listed in the log file that resides in the same directory as the input file. It is always a good idea to check this file even if no errors are reported. By default, a user can abort the solution process by pressing ESC key. You can customize this behavior by calling `cgilStructure:: setAbortSolutionKey ()`.

```

// save the data to a file for possible later use
bool b2 = pStructure->saveDocument(szInputFileName);

// run static analysis
bool bRun = pStructure->runStaticAnalysis();
if(!bRun)
{
    COUT << _T("Error running static analysis... ") << szInputFileName << endl;
    return;
}

// report
pStructure->setListMessageFunction(0); // do not list message
if(!pStructure->runReport())
{
    // report will be saved in ttestt.htm file
    COUT << _T("Error running report... ") << szInputFileName << endl;
    return;
}

```

The analysis results can be retrieved easily by calling `getStaticResults()` function as illustrated in the following. You need to delete memory by calling the interface function `deleteMemoryArray()` as shown below.

```

// extract results

```

```

cgiCombinationResult* vCombResult = NULL;
int nCombResultCount = 0;
pStructure->getStaticResults(vCombResult, nCombResultCount);
// list displacements for all load combinations
TCHAR szTranslationalDisplacementUnit[LABEL_SIZE];
TCHAR szRotationalDisplacementUnit[LABEL_SIZE];
pStructure->getUnit(szTranslationalDisplacementUnit, DISPLACEMENT_TRANS);
pStructure->getUnit(szRotationalDisplacementUnit, DISPLACEMENT_ROTATE);
COUT << _T("-----") << endl;
COUT << _T("----- Verifying Example 4 -----") << endl;

COUT << endl << _T("Nodal Displacements. Units: ") << szTranslationalDisplacementUnit << _T(", ")
    << szRotationalDisplacementUnit << endl;
for(int iComb = 0; iComb < nCombResultCount; iComb++)
{
    COUT << _T("Load combination ") << iComb + 1 << ": " << vLoadComb[iComb].getLabel() << endl;

    int nDisps = vCombResult[iComb].m_nNdDispCount;
    COUT << setprecision(4) << fixed;
    for(int i = 0; i < nDisps; i++) {
        const cgiResult6Val& disp = vCombResult[iComb].m_vNdDisp[i];
        COUT << _T("Node- ") << disp.iId << _T(": ")
            << disp.fVal[X] << "\t" << disp.fVal[Y] << "\t" << disp.fVal[Z] << "\t"
            << disp.fVal[OX] << "\t" << disp.fVal[OY] << "\t" << disp.fVal[OZ] << endl;
    }
    COUT << endl;
}

// list internal forces and moments at beam ends for all load combinations
TCHAR szForceUnit[LABEL_SIZE];
TCHAR szMomentUnit[LABEL_SIZE];
pStructure->getUnit(szForceUnit, FORCE);
pStructure->getUnit(szMomentUnit, MOMENT);
COUT << endl << _T("Beam End Forces and Moments. Units: ") << szForceUnit << _T(", ") << szMomentUnit << endl;
for(int iComb = 0; iComb < nCombResultCount; iComb++)
{
    COUT << _T("Load combination ") << iComb + 1 << ": " << vLoadComb[iComb].getLabel() << endl;

    int nBeams = vCombResult[iComb].m_nBmVMDCount;
    for(int i = 0; i < nBeams; i++) {
        const cgiBeamShearMomentDeflection& bmVMD = vCombResult[iComb].m_vBmVMD[i];

        cgiVMD* buffer = NULL;

```

```

int count = 0;
bmVMD.getBeamVMDValues(buffer, count);

int iSeg[2];
iSeg[0] = 0;
iSeg[1] = (int)count - 1;
COUT << setprecision(4) << fixed;
for(int k = 0; k < 2; k++)
{
    const cgiVMD& vmd = buffer[iSeg[k]];
    TCHAR* szEnd = (k==0)? _T("Start") : _T("End");
    COUT << _T("Beam ") << bmVMD.getId() << _T(" ") << szEnd << _T(": ")
        << vmd.fVMD[X] << "\t" << vmd.fVMD[Y] << "\t" << vmd.fVMD[Z] << "\t"
        << vmd.fVMD[OX] << "\t" << vmd.fVMD[OY] << "\t" << vmd.fVMD[OZ] << endl;
} // for(int k = 0; k < bmVMD.vVMD.size(); k++)

} // for(int i = 0; i < nBeams; i++) {
COUT << endl;
}

pStructure->deleteMemoryArray(vCombResult);

```

You can also retrieve analysis results for a single load combination as shown in the following.

```

cgiCombinationResult* vCombResult2 = NULL;
int loadCombinationIndex = 1;
bool bSuccess = pStructure->getStaticResultsForSingleLoadCombination(vCombResult2, loadCombinationIndex);
if (bSuccess) {
    COUT << _T("----- Single Load combination -----") << endl;
    COUT << _T("Load combination ") << loadCombinationIndex + 1 << ": " << vLoadComb[loadCombinationIndex].getLabel() << endl;

    int nDisps = vCombResult2[0].m_nNdDispCount;
    COUT << setprecision(4) << fixed;
    for (int i = 0; i < nDisps; i++) {
        const cgiResult6Val& disp = vCombResult2[0].m_vNdDisp[i];
        COUT << _T("Node- ") << disp.iId << _T(": ")
            << disp.fVal[X] << "\t" << disp.fVal[Y] << "\t" << disp.fVal[Z] << "\t"
            << disp.fVal[OX] << "\t" << disp.fVal[OY] << "\t" << disp.fVal[OZ] << endl;
    }
    COUT << endl;
    pStructure->deleteMemoryArray(vCombResult2);
}

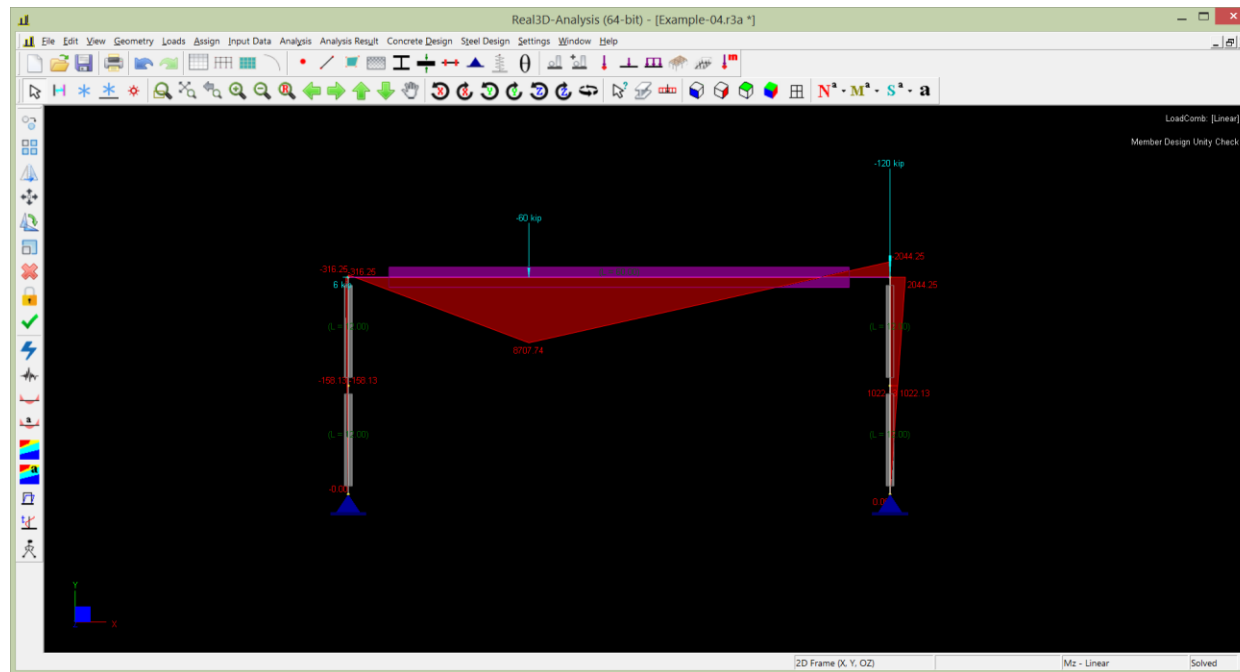
```

}

For your references, the following table shows the critical nodal displacements and member internal moments.

		Real3D	Reference
Linear	Maximum Displacement (in)	4.387	4.4
	Max + moment in beam (in-kips)	8707.7	8708
	Max – moment in beam (in-kips)	2044.3	2044
P-Delta	Maximum Displacement (in)	8.26	8.1
	Max + moment in beam (in-kips)	9079.4	9078
	Max – moment in beam (in-kips)	2663.3	2661

You can use Real3D to view the model, perform analysis and view its results (shown below). You may need to zoom extent from View ->Zoom->Zoom Extent menu and adjust graphics scales from Settings->Graphics Scales.



Example Model Two (refer to the sample file verify-plate-patch-test.cpp included in the library)

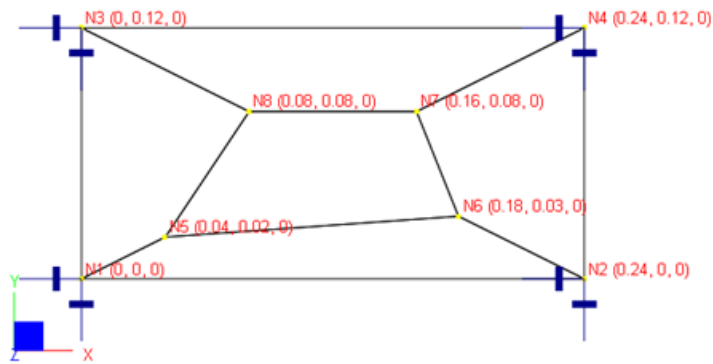
A plate of size 0.12 x 0.24 in is subjected to forced displacements at the four corners as shown below. The boundary conditions are:

$$w = 1.0e^{-3}(x^2 + xy + y^2) / 2$$

$$\theta_x = \frac{\partial w}{\partial y} = 1.0e^{-3}(y + x/2) ; \theta_y = -\frac{\partial w}{\partial x} = 1.0e^{-3}(-x - y/2)$$

Material properties: $E = 1.0e6$ psi, $\nu = 0.25$

Geometry: nodal coordinates are shown in the parenthesis below, thickness $t = 0.001$ in



Forced displacements on boundary nodes (units: displacement – in; rotation – rad)

Boundary Nodes	Displacement Dz	Rotation D _{ox}	Rotation D _{oy}
1	0	0	0
2	2.88e-5	1.20e-4	-2.40e-4
3	7.20e-6	1.20e-4	-6.00e-5
4	5.04e-5	2.40e-4	-3.00e-4

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

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

#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;
}

static int MlkProgress( int* ithr, int* step, char* stage, int len )
{
    COUT << "MLK thread = " << *ithr << "; step = " << *step << "; stage = " << stage << endl;
    return 0;
}

void verify_plate_patch_test()
{
    cgiIStructure* pStructure = CreateStructure();

    TCHAR szTestPath[256];
    TCHAR szInputFileName[256];
    pStructure->getDefaultTestPath(szTestPath, 256);
    _stprintf(szInputFileName, _T("%s\\tests\\newModels\\%s"), szTestPath, _T("Verify-PlatePatchTest.r3a"));

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

    pStructure->setModelType(kModel_PlateBending); // a 2D plate bending

    // in, lb and rad
```

```

pStructure->setConsistentEnglishUnits();

// define materials
vector<cgiMaterial> vMat;
cgiMaterial mat;
mat.setId(1); // material id, to be referred later
mat.setProperties(_T("Default"), 1e+006, 0.25, 0.28); // material label, young's modulus, poisson ratio, weight density
vMat.push_back(mat);
pStructure->setMaterials(&vMat[0], vMat.size());

// define sections
vector<cgiThickness> vThick;
cgiThickness thick;
thick.setId(1);
thick.setProperties(_T("Default"), 0.001); // thickness label, thickness
vThick.push_back(thick);
pStructure->setThicknesses(&vThick[0], vThick.size());

// define nodes
double coordinates[][2] = {{0.00, 0.00},{0.24, 0.00},{0.00, 0.12},{0.24, 0.12},
                          {0.04, 0.02},{0.18, 0.03},{0.16, 0.08},{0.08, 0.08}};
vector<cgiNode> vNd;
for(int i = 0; i < 8; i++)
{
    cgiNode nd;
    nd.setId(i + 1); // nodal id
    nd.setCoordinates(coordinates[i][0], coordinates[i][1], 0); // nodal coordinates
    vNd.push_back(nd);
}
pStructure->setNodes(&vNd[0], vNd.size());

int connectivity[][4] = {{1, 2, 6, 5},{6, 7, 8, 5},{2, 4, 7, 6},{4, 3, 8, 7},{5, 8, 3, 1}};

// define shells
vector<cgiShell4> vShell;
for(int i = 0; i < 5; i++)
{
    cgiShell4 shell;
    shell.setId(i + 1);
    shell.setNodes(connectivity[i][0], connectivity[i][1], connectivity[i][2], connectivity[i][3]);
    shell.setProperties(1, 1, 0.0);
    vShell.push_back(shell);
}
swap(vShell[0], vShell[4]); // for testing purpose
pStructure->setShell4s(&vShell[0], vShell.size());

double fForcedDisplacement[][6] = {{0, 0, 0, 0, 0, 0},
                                   {0, 0, 2.88e-005, 0.00012, -0.00024, 0},
                                   {0, 0, 7.2e-006, 0.00012, -6e-005, 0},
                                   {0, 0, 5.04e-005, 0.00024, -0.0003, 0}};

// define supports
vector<cgiSupport> vSupt;

```

```

for(int i = 0; i < 4; i++)
{
    cgiSupport supt;
    supt.setId(i + 1); // nodal id that this support is on
    // six DOFs, 1 for supported, 0 for free; forced settlements and rotations
    supt.setSupportDOFs(_T("001110"), fForcedDisplacement[i][X], fForcedDisplacement[i][Y], fForcedDisplacement[i][Z],
        fForcedDisplacement[i][OX], fForcedDisplacement[i][OY], fForcedDisplacement[i][OZ]);
    vSupt.push_back(supt);
}
pStructure->setSupports(&vSupt[0], vSupt.size());

// define load cases
vector<cgiLoadCase> vLoadCase;
cgiLoadCase loadcase;
loadcase.setId(1); // load case id
loadcase.setLoadCase(_T("Default"), _T("DEAD"), TRUE); // load case label; type such as DEAD, LIVE etc; report on this load case
vLoadCase.push_back(loadcase);
pStructure->setLoadCases(&vLoadCase[0], vLoadCase.size());

// define load combinations
vector<cgiLoadCombination> vLoadComb;
cgiLoadCombination loadcomb; // load combination
loadcomb.clearLoadCombItem(); // make sure we start clean with this load combination
// load combination label, linear combination (no pdelta effect), want report on this combination
loadcomb.setLoadComb(_T("Patch-Test"), FALSE, TRUE);
loadcomb.addLoadCombItem(_T("Default"), 1.0); // add load case and factor to this load combination
vLoadComb.push_back(loadcomb);
pStructure->setLoadCombinations(&vLoadComb[0], vLoadComb.size());

// note: there is no actual loads for patch test
// or you can say force support displacements are a form of loads

// set report options
cgiReportOptions reportOptions;
reportOptions.SelectAll();
reportOptions.bHTML = FALSE;
pStructure->setReportOptions(reportOptions);

// set analysis options
bool bConsiderBeamShearDeformation = FALSE;
int nMaximumPDeltaIterations = 10;
double fPDeltaToleranceInPercentage = 0.5; // in percentage
int nNumberOfSegmentsForBeamOutput = 20;
bool bUseThinPlate = false;
bool bUseCompatibleModes = true;
int nUseAverageStressMode = 1;
pStructure->setAnalysisOptions(bConsiderBeamShearDeformation, nMaximumPDeltaIterations, fPDeltaToleranceInPercentage,
    nNumberOfSegmentsForBeamOutput, bUseThinPlate, bUseCompatibleModes, nUseAverageStressMode);

// save the data to a file for possible later use
bool b2 = pStructure->saveDocument(szInputFileName);

```

```

// run static analysis
if(!pStructure->runStaticAnalysis())
{
    COUT << _T("Error running static analysis...") << szInputFileName << endl;
    return;
}

// report
pStructure->setListMessageFunction(0); // do not list message
if(!pStructure->runReport())
{
    // report will be saved in ttestt.htm file
    COUT << _T("Error running report...") << szInputFileName << endl;
    return;
}

// extract results
cgiCombinationResult* vCombResult = NULL;
int nCombResultCount = 0;
pStructure->getStaticResults(vCombResult, nCombResultCount);
// list displacements for all load combinations
TCHAR szTranslationalDisplacementUnit[LABEL_SIZE];
TCHAR szRotationalDisplacementUnit[LABEL_SIZE];
pStructure->getUnit(szTranslationalDisplacementUnit, DISPLACEMENT_TRANS);
pStructure->getUnit(szRotationalDisplacementUnit, DISPLACEMENT_ROTATE);
COUT << _T("-----") << endl;
COUT << _T("----- Verifying Plate Patch Test -----") << endl;

COUT << endl << _T("Nodal Displacements. Units: ") << szTranslationalDisplacementUnit << _T(", ") << szRotationalDisplacementUnit << endl;
for(int iComb = 0; iComb < nCombResultCount; iComb++)
{
    COUT << _T("Load combination ") << iComb + 1 << ": " << vLoadComb[iComb].getLabel() << endl;

    int nDisps = vCombResult[iComb].m_nNdDispCount;
    COUT << setprecision(3) << scientific;
    for(int i = 0; i < nDisps; i++) {
        const cgiResult6Val& disp = vCombResult[iComb].m_vNdDisp[i];
        COUT << _T("Node- ") << disp.iId << _T(": ")
            << disp.fVal[X] << "\t" << disp.fVal[Y] << "\t" << disp.fVal[Z] << "\t"
            << disp.fVal[OX] << "\t" << disp.fVal[OY] << "\t" << disp.fVal[OZ] << endl;
    }
    COUT << endl;
}

TCHAR szLinearMomentUnit[LABEL_SIZE];
TCHAR szLinearShearUnit[LABEL_SIZE];
pStructure->getUnit(szLinearMomentUnit, MOMENT_LINE);
pStructure->getUnit(szLinearShearUnit, FORCE_LINE);
COUT << endl << _T("Plate Internal Moments Mxx, Myy, Mxy and Shears Vxx, Vyy. Units: ") << szLinearMomentUnit << _T(", ") << szLinearShearUnit << endl;
for(int iComb = 0; iComb < nCombResultCount; iComb++)
{
    COUT << _T("Load combination ") << iComb + 1 << ": " << vLoadComb[iComb].getLabel() << endl;
}

```

```

int nShell4s = vCombResult[iComb].m_nShell4StressCount;
for(int i = 0; i < nShell4s; i++) {
    const cgiShellStress& stress = vCombResult[iComb].m_vShell4Stress[i];

    cgiShell4 shell4;
    pStructure->getSingleShell4(shell4, stress.iId);
    for(int k = 0; k < 5; k++)
    {
        TCHAR szNode[LABEL_SIZE];
        if(k == 0)
        {
            _tcscpy(szNode, _T("Center"));
        }
        else
        {
            _stprintf(szNode, _T("Node-%d"), shell4.iNode[k - 1]);
        }

        TCHAR szLine[MAX_LINE_SIZE];
        _stprintf(szLine, _T("Plate-%d, %s: %g, %g, %g, %g, %g"), shell4.iId, szNode, stress.fStress_b[k][0], stress.fStress_b[k][1],
            stress.fStress_b[k][2], stress.fStress_b[k][3], stress.fStress_b[k][4]);
        COUT << szLine << endl;
    } // for(int k = 0; k < 4; k++)
    COUT << endl;

} // for(int i = 0; i < nShell4s; i++)
COUT << endl;
}

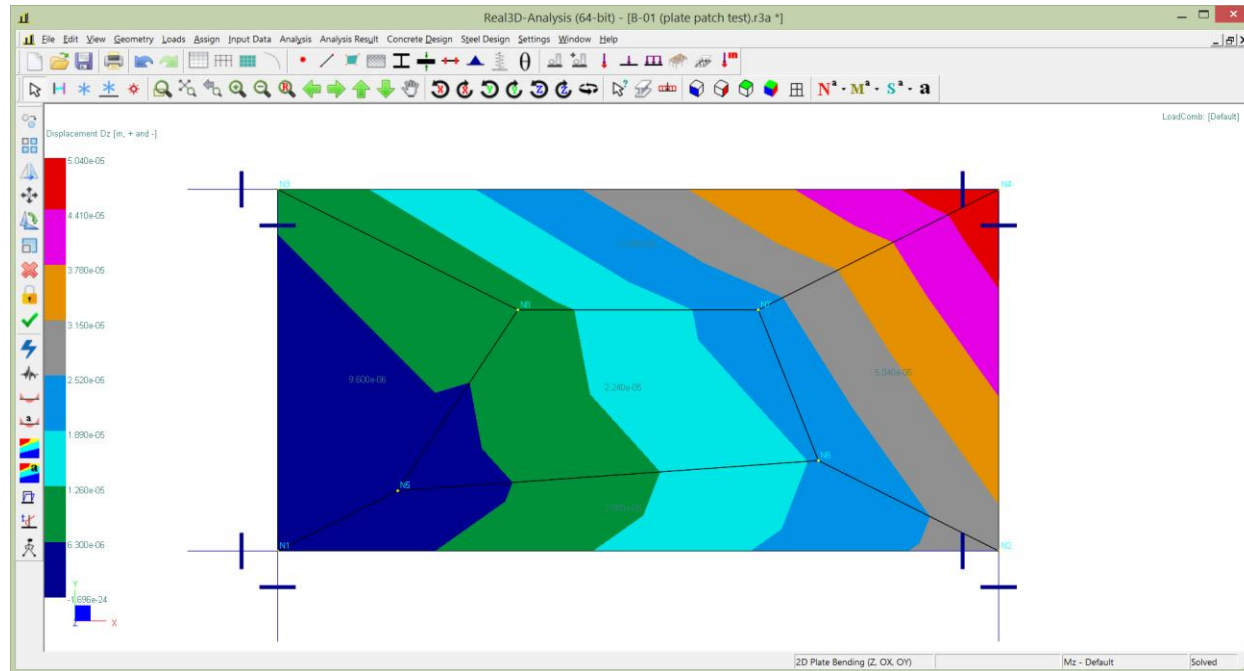
pStructure->deleteMemoryArray(vCombResult);
}

```

The following is the detailed explanations for Example Model Two

Please refer the detailed explanations for Example Model One in the previous section as the procedures involved are similar.

You can use Real3D to view the model, perform analysis and view its results (shown below). You may need to zoom extent from View ->Zoom->Zoom Extent menu and adjust graphics scales from Settings->Graphics Scales.



.NET Interface through C++/CLI

The .NET interface is a class library assembly `cgiSolverBlazeCli.dll`. It is written in C++/CLI and forwards calls to a native Windows DLL called `cgiSolverBlaze.dll`. The actual interface functions are contained in a single class called `cgiSolverBlazeCli` as listed in the following. The .NET Core interface is a class library assembly `cgiSolverBlazeCoreCli.dll`. The interface functions are identical to those in .NET Framework interface. Please note that x64 version of `cgiSolverBlaze.dll` depends on two other Windows native DLLs `libiomp5md.dll` and `double128Proj.dll`. It is important to point out that at this time (January, 2026), C++/CLI is only supported on Windows x64 platform targets and is NOT supported on Windows ARM64 platform target.

Different versions of `cgiSolverBlazeCli.dll` are provided to work with .NET Framework 4.0, 4.5x, 4.6x, 4.7x, and 4.8x on x64 CPUs. Different versions of `cgiSolverBlazeCoreCli.dll` are provided to work with .NET Core 5.0, 6.0, 7.0, 8.0, 9.0, 10.0 on x64 CPUs. Make sure your project has reference to the correct `cgiSolverBlazeCli.dll` or `cgiSolverBlazeCoreCli.dll`. If your project is configured to build for x64 CPUs, make sure to set “Copy Local” to false for the DLL reference and manually copy the correct version of `cgiSolverBlazeCli.dll` or `cgiSolverBlazeCoreCli.dll` to the executable directory. Also make sure a copy of the native dependent DLLS (`cgiSolverBlaze.dll`, `libiomp5md.dll`, and `double128Proj.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.

`cgiSolverBlazeCli` 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 `cgiSolverBlazeTestCSharp`. These examples are taken from the Verification Manual of Real3D, which is a structural and finite element analysis program by CGI. These examples include 2D/3D frame analysis, plate bending analysis, frequency analysis and response spectrum analysis. A full input and output report can be produced after each analysis. You can compare the reports with those produced from within Real3D.

The following lists all the interface functions exposed by `cgiSolverBlazeClass` in `cgiSolverBlazeCli` namespace for .NET and `cgiSolverBlazeCoreCli` namespace for .NET Core:

```
namespace cgiSolverBlazeCli
{
    public class cgiSolverBlazeClass : IDisposable
    {
        public void setListMessageFunction(cgiSolverBlazeClass.ListMessageDelegate fnListMsg);
        public void setStatusMessageFunction(cgiSolverBlazeClass.StatusMessageDelegate fnStatusMsg);
        public void setSparseSolverStatusFunction(cgiSolverBlazeClass.SparseSolverProgressDelegate fnSparseStatus);
    }
}
```

```

public bool createStructure();
public void getExePath(ref StringBuilder exePath);
public void getDefaultTestPath(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 void clearModel();
public void setModelType(int nModelType);
public int getModelType();
public void setSuppressedDOFs(List<bool> bSuppress);
public void getSuppressedDOFs(ref List<bool> bSuppress);
public void setAbortSolutionKey(int key);
public int getAbortSolutionKey();

public void setAnalysisOptions( bool bConsiderBeamShearDeformation, int nMaximumPDeltaIterations,
                                double fPDeltaToleranceInPercentage, int nNumberOfSegmentsForBeamOutput, bool bUseThinPlate,
                                bool bUseCompatibleModes, int nUseAverageStress);
public void getAnalysisOptions(ref bool bConsiderBeamShearDeformation, ref int nMaximumPDeltaIterations,
                                ref double fPDeltaToleranceInPercentage, ref int nNumberOfSegmentsForBeamOutput,
                                ref bool bUseThinPlate, ref bool bUseCompatibleModes, ref int nUseAverageStress);
public void setFrequencyAnalysisOptions(int nEigenNumber, double fEigenVlaueTolerance, int nMaximumSubspaceIterations,
                                        int nIterationVectors, bool bConvertLoadToMass, int nGravityDirection,
                                        int nLoadCombinationForMass);
public void getFrequencyAnalysisOptions(ref int nEigenNumber, ref double fEigenVlaueTolerance,
                                        ref int nMaximumSubspaceIterations, ref int nIterationVectors,
                                        ref bool bConvertLoadToMass, ref int nGravityDirection, ref int nLoadCombinationForMass);
public void setResponseSpectrumAnalysisOptions(List<cgiSpectrumCli> spectrumsXYZ, double fDampingRatio, int combinationMethod,
                                                List<double> fSpectrumDirectionalFactorXYZ, bool bUseDominantModeForSignage);
public void getResponseSpectrumAnalysisOptions(ref List<cgiSpectrumCli> spectrumsXYZ, ref double fDampingRatio,
                                                ref int combinationMethod, ref List<double> fSpectrumDirectionalFactorXYZ,

```

```

        ref bool bUseDominantModeForSignage);

public void setApplyStiffnessModification(bool bApply);
public bool getApplyStiffnessModification()const;

public void setTolerance(double fTolerance);
public double getTolerance();
public void setFictitiousOzFactor(double fFictitiousOzFactor);
public double getFictitiousOzFactor();
public void setUseOoc(bool bUseOoc);    // Ooc = Out of core solver
public bool getUseOoc();
public void setEpsilon(double fEpsilon);
public double getEpsilon();

public void setStressLocation(int nStressLocation);
public int getStressLocation();
public void setGravityFactor(double fGravityFactor);
public double getGravityFactor();
public void setGravityDirection(int nGravityDirection);
public int getGravityDirection();
public void setGravityLoadCase(int nGravityLoadCase);
public int getGravityLoadCase();
public void setUseZAsVerticalAxis(bool useZAsVerticalAxis);
public bool getUseZAsVerticalAxis();
public void setSolver(int iSolver);
public int getSolver();

public void setMaterials(List<cgiMaterialCli> listMat);
public void getMaterials(ref List<cgiMaterialCli> listMat);
public void setSections(List<cgiSectionCli> listSect);
public void getSections(ref List<cgiSectionCli> listSect);
public void setThicknesses(List<cgiThicknessCli> listThick);
public void getThicknesses(ref List<cgiThicknessCli> listThick);
public void setSupports(List<cgiSupportCli> listSupt);
public void getSupports(ref List<cgiSupportCli> listSupt);
public void setNodalSprings(List<cgiSpringCli> listNdSpring);
public void getNodalSprings(ref List<cgiSpringCli> listNdSpring);
public void setCoupledSprings(List<cgiCoupledSpringCli> listCoupledSpring);
public void getCoupledSprings(ref List<cgiCoupledSpringCli> listCoupledSpring);
public void setLineSprings(List<cgiSpringCli> listLnSpring);
public void getLineSprings(ref List<cgiSpringCli> listLnSpring);
public void setSurfaceSprings(List<cgiSpringCli> listShell4Spring);
public void getSurfaceSprings(ref List<cgiSpringCli> listShell4Spring);

```

```

public void setMomentReleases(List<cgiReleaseCli> listRels);
public void getMomentReleases(ref List<cgiReleaseCli> listRels);
public void setDiaphragms(List<cgiDiaphragmCli> listDiaphragms);
public void getDiaphragms(ref List<cgiDiaphragmCli> listDiaphragms);
public void setDiaphragmOptions(double fDiaphragmStiffnessFactor, bool bConsiderDiaphragm);
public void getDiaphragmOptions(ref double fDiaphragmStiffnessFactor, ref bool bConsiderDiaphragm);

public void setMultiDofConstraints(List<cgiMultiDofConstraintCli> listMultiDofConstraints);
public void getMultiDofConstraints(ref List<cgiMultiDofConstraintCli> listMultiDofConstraints);

public void setNodes(List<cgiNodeCli> listNd);
public void getNodes(ref List<cgiNodeCli> listNd);
public void setBeams(List<cgiBeamCli> listBm);
public void getBeams(ref List<cgiBeamCli> listBm);
public void setShell4s(List<cgiShell4Cli> listShell4);
public void getShell4s(ref List<cgiShell4Cli> listShell4);
public void setBricks(List<cgiBrickCli> listBrick);
public void getBricks(ref List<cgiBrickCli> listBrick);
public void getSingleNode(ref cgiNodeCli item, int nId);
public void getSingleBeam(ref cgiBeamCli item, int nId);
public void getSingleShell4(ref cgiShell4Cli item, int nId);
public void getSingleBrick(ref cgiBrickCli item, int nId);

public void getBeamLocalAngleByThirdPoint(cgiPointCli beamStartPt, cgiPointCli beamEndPt, cgiPointCli thirdPt);
public void getBeamLocalAxes(ref cgiPointCli xAxis, ref cgiPointCli yAxis, ref cgiPointCli zAxis, cgiPointCli point1,
    cgiPointCli point2, double fGamma);
public void getShellLocalAxes(ref cgiPointCli xAxis, ref cgiPointCli yAxis, ref cgiPointCli zAxis, cgiPointCli point1,
    cgiPointCli point2, cgiPointCli point3, cgiPointCli point4, double fGamma);
public bool matchShellLocalxAxis(int sourceShellId, int targetShellId);
public bool matchShellLocalzAxis(int sourceShellId, int targetShellId);

public void setLoadCases(List<cgiLoadCaseCli> listLoadCase);
public void getLoadCases(ref List<cgiLoadCaseCli> listLoadCase);
public void setLoadCombinations(List<cgiLoadCombCli> listLoadComb);
public void getLoadCombinations(ref List<cgiLoadCombCli> listLoadComb);
public void setCaseLoads(List<cgiCaseLoadCli> listCaseLoad);
public void getCaseLoads(ref List<cgiCaseLoadCli> listCaseLoad);
public void setNodalMasses(List<cgiNodalLoadCli> listNdMass);
public void getNodalMasses(ref List<cgiNodalLoadCli> listNdMass);
public void getCalculatedNodalMasses(ref List<cgiNodalLoadCli> listNdCombMass);
public void convertLocalLoadsToGlobalLoads();
public void convertAreaLoadsToLineLoads();
public void setReportOptions(cgiReportOptionsCli reportOptions);

```

```

public void getReportOptions(ref cgiReportOptionsCli reportOptions);
public int getEquations();
public void getStaticResults(ref List<cgiCombResultCli> listCombResult);
public int getStaticResultsForSingleLoadCombination(ref List<cgiCombResultCli> listCombResult, int loadCombinationIndex);
public bool getNodalDisplacementsEnvelope(ref List<cgiEnvelopeValue6Cli> result, List<int> envelopeLoadCombinationIndexes);
public bool getNodalDisplacementsEnvelope(ref List<cgiEnvelopeValue6Cli> result, List<int> envelopeLoadCombinationIndexes);
public bool getSupportsEnvelope(ref List<cgiEnvelopeValue6Cli> result, List<int> envelopeLoadCombinationIndexes);
public bool getBeamVmdEnvelope(ref List<cgiEnvelopeBmVMDCli> result, List<int> envelopeLoadCombinationIndexes);
public bool getShell4GroupNodalResultant(ref cgiNodalResultantCli result, int loadCombinationIndex, List<int> selectedNodes,
List<int> selectedShell4s, cgiPointCli resultLocation, int referenceShellId);

public bool getBeamStressesForSingleLoadCombination(ref List<cgiBeamStressCli> listBeamStress, int loadCombinationIndex);

public void getEigenResults(ref List<cgiEigenResultCli> listEigenResult);
public int getEigenResultsForSingleMode(ref List<cgiEigenResultCli> listEigenResult, int modeIndex);
public void getResponseSpectrumResults(ref List<cgiResponseSpectrumResultCli> listResult);
public int getResponseSpectrumResultsForSingleMode(ref List<cgiResponseSpectrumResultCli> listResult, int modeIndex);
public void getModalCombinationResults(ref cgiCombResultCli modalResult);
public void getModalCombinationBaseShears (ref double fBaseShearX, ref double fBaseShearY, ref double fBaseShearZ);

public bool saveDocument(string pathName);
public bool openDocument(string pathName);

public void clearResult();
public void clearStaticResult();
public void clearFrequencyResult();
public void clearResponseSpectrumResult();
public bool checkInputData(int iMode, bool bReport);
public bool hasStaticSolution();
public bool hasEigenSolution();
public bool hasResponseSpectrumSolution();
public bool runStaticAnalysis();
public bool runFrequencyAnalysis( bool bCalcMassOnly);
public bool runResponseSpectrumAnalysis();
public bool runReport();
public void getContourMinMax(ref double fMin, ref double fMax, int iContourIndex, int iComb, int iStressAtShellTopBottom);

public int getLastError();
public void clearLastError();
public void setShowSolverMessageBox( bool bShow);
public bool getShowSolverMessageBox();

public int removeAllOrphanedNodes(ref List<int> removedNodeIds);

```

```

public bool mergeAllNodesAndElements(ref cgiMergedInfoCli info);
public bool insertNodesAtBeamIntersections( ref List<cgiIntersectionNodeCli> intersectionNodes, List<int> beamIds);
public int explodeBeamsAtNodes(ref List<int> affectedBeamIds, List<int> beamIds);

public delegate void ListMessageDelegate(string A_0, string A_1);
public delegate void StatusMessageDelegate(string A_0);
public delegate int SparseSolverProgressDelegate(IntPtr A_0, IntPtr A_1, IntPtr A_2, int A_3);

// clear functions
virtual void clearMaterials();
virtual void clearSections();
virtual void clearThicknesses();
virtual void clearSupports();
virtual void clearNodalSprings();
virtual void clearCoupledSprings();
virtual void clearLineSprings();
virtual void clearSurfaceSprings();
virtual void clearMomentReleases();
virtual void clearDiaphragms();
virtual void clearMultiDofConstraints();
virtual void clearNodes();
virtual void clearBeams();
virtual void clearShell4s();
virtual void clearBricks();
virtual void clearLoadCases();
virtual void clearLoadCombinations();
virtual void clearCaseLoads();
virtual void clearNodalMasses();
}
}

```

The following lists a few useful enums.

```

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,
    }

    public enum cgiModelEnum
    {
        kModel_Frame3D,
        kModel_Frame2D,
        kModel_Truss3D,
        kModel_Truss2D,
        kModel_PlateBending,
        kModel_PlaneStress,
        kModel_Brick,
        kModel_Grillage,
        kModel_End,
    }

    public enum cgiDofEnum
    {
        X,
        Y,
        Z,
        OX,
        OY,
        OZ,
    }

    public enum cgiLoadSystemEnum
    {
        LOCAL,
        GLOBAL,
        PROJECTED,// not supported at this time
    }

    public enum cgiDistanceSpec
    {
        PERCENT,
        DISTANCE,
    }

    public enum cgiMemberNonlinearTypeEnum
    {

```

```

        kMemberLinear,
        kMemberTensionOnly,
        kMemberCompressionOnly,
    }

public enum cgiDiaphragmTypeEnum
{
    kDiaphragm_Generic,
    kDiaphragm_XZ,
    kDiaphragm_YZ,
    kDiaphragm_XY,
}

// units are are in SPRING_TRANS_1D, FORCE/DISPLACEMENT_ROTATE, and SPRING_ROTATE_1D for corresponding terms
public enum cgiCoupledSpringStiffnessTermEnum {
    Kx_Kx, Kx_Ky, Kx_Kz, Kx_Kox, Kx_Koy, Kx_Koz,
    Ky_Ky, Ky_Kz, Ky_Kox, Ky_Koy, Ky_Koz,
    Kz_Kz, Kz_Kox, Kz_Koy, Kz_Koz,
    Kox_Kox, Kox_Koy, Kox_Koz,
    Koy_Koy, Koy_Koz,
    Koz_Koz,
    kCoupledSpringTermsEnd
};

public enum cgiSolverEnum
{
    kSolver64,
    kSolver128,
    kSolverSparse64,
}

public enum cgiIncludeEnum
{
    kInclude,
    kExclude,
}

public enum cgiErrorEnum
{
    kErrorNone,
    kInvalidInput,
    kErrorProcessingElementStiffness,
    kNoMassDefined,
    kErrorConvertingAreaLoadsToLineLoads,
    kTooManyDigitsLostDuringFactorization,
    kSolverError,
    kAbnormalSolverTermination,
    kMustRunResponseSpectrumAnalysis,
    kMustRunFrequencyAnalysisFirst,
    kFileAccessError,
    kLicenseError,
}

```

```

        kInvalidLoadCombinationIndex,
        kInvalidModeIndex,
        kNoResultAvailable,
        kUnknownError
    }

    public enum cgiResponseSpectrumCombinationMethodEnum
    {
        kCombination_Cqc,
        kCombination_Srss,
        kCombination_AbsSum,
    }

    public enum cgiAreaLoadDistributionMethodEnum
    {
        kTwoWay,
        kShortSides,
        kLongSides,
        kAB_CD,
        kBC_AD,
        kCentroid,
        kCircumference,
        kAreaLoad_Distribution_End
    }

    public enum cgiStressAveragingMethodEnum
    {
        kStressAveragingNone,
        kStressAveragingAll,
        kStressAveragingLocal
    }

```

The following lists a few useful constants.

```

const int LABEL_SIZE = 128;
const int MAX_SPECTRUM_DATA_POINTS = 128;

```

The following lists result data structures.

```

public class cgiResult6ValCli
{
    public int iId;           // node or element number
    public List<double> fVal; // values in six degrees of freedom (DOF) directions
}

```

```

public ref class cgiMultiDofConstraintForceCli
{
public:
    int iId;           // constraint id
    int iDof1;        // first constrained degrees of freedom direction
    int iDof2;        // second constrained degrees of freedom direction
    int iNode1;       // first constrained node
    int iNode2;       // second constrained node
    double fVal1;     // constrained force or moment depending on the first constrained DOF
    double fVal2;     // constrained force or moment depending on the second constrained DOF
};

public class cgiVMDCli
{
    public double fDist;           // ratio
    public List<double> fVMD;      // forces and moments in six DOF directions
    public List<double> fDef1;    // deflections in member local y and z directions
}
public class cgiBeamVMDCli
{
    public int iId;               // member id
    public List<cgiVMDCli> vmd;   // forces and moments at segments of a member
}

public class cgiBeamEndVMDCli
{
    public int iId;              // element number
    public List<cgiVMDCli> vmd;  // forces and moments at two ends of a member
}

public class cgiShellStressCli
{
    // shell element number
    public int iId;

    // Fxx, Fyy, Fxy (at the center and four nodes of the shell)
    // size of [5][3]
    public List<List<double>> fStress_m;

    // Mxx, Myy, Mxy, Fv1, Fv2 (at the center and four nodes of the shell)
    // size of [5][3]
    public List<List<double>> fStress_b;

    // Top & Bottom, Sxx, Syy, Szz, Sxy, Syz, Sxz
    // (combined membrane and bending stresses at the center and four nodes of the shell)
    // size of [2][5][6]
    public List<List<List<double>>> fStress;
}

```

```

// nodal force resultants for membrane, in local coordinate
// (Fx, Fy component at four nodes of the shell)
// size of [8]
public List<double> fForce_m;

// nodal moment and force resultants for bending, in local coordinate
// (Mx, My, Fz components at four nodes of the shell)
// size of [12]
public List<double> fForce_b;
}

public class cgiBrickStressCli
{
    public int iId; // brick element element number
    public List<List<double>> fStress; // Fxx, Fyy, Fzz, Fxy, Fyz, Fxz at element center + eight nodes
    public List<double> fForce; // Fx, Fy, Fz nodal resultant at eight nodes
}

// results for a load combination
public class cgiCombResultCli
{
    public List<cgiResult6ValCli> m_listNdDisp; // nodal displacements
    public List<cgiResult6ValCli> m_listSuptReact; // support reactions
    public List<cgiMultiDofConstraintForceCli> m_listMultiDofConstraintForce; // multi-dof constraint reactions
    public List<cgiResult6ValCli> m_listSpringReact_Node; // nodal spring reactions
    public List<cgiResult6ValCli> m_listSpringReact_Coupled; // coupled spring reactions
    public List<cgiResult6ValCli> m_listSpringReact_Beam; // line spring reactions
    public List<cgiResult6ValCli> m_listSpringReact_Shell4; // surface spring reaction
    public List<cgiShellStressCli> m_listShell4Stress; // shell stresses
    public List<cgiBrickStressCli> m_listBrickStress; // brick stresses
    public List<cgiFixedEndMomentCli> m_listFEM; // member fixed end forces
    public List<cgiBeamEndVMDCli> m_listBmEndVMD; // vmd at member ends
    public List<cgiBeamVMDCli> m_listBmVMD; // vmd at segmental points along member
}

public class cgiEnvelopeValue6Cli
{
    public int iId;
    public List<double> fMax; // six values
    public List<double> fMin; // six values
    public List<int> iMaxComb; // six values
    public List<int> iMinComb; // six values
}

public class cgiEnvelopeVMDCli
{
    public double fDist;
    public List<double> fMaxVMD; // six values
    public List<double> fMinVMD; // six values
    public List<int> iMaxComb; // six values
    public List<int> iMinComb; // six values
}

```

```

public class cgiEnvelopeBmVMDcli
{
    public int iId;
    public List<cgiEnvelopeVMDcli> vmds;
}

public class cgiSectionStressCli
{
    public double fDist;
    public List<double> normalStress;
    public double fMaxCompressiveStress;
    public double fMaxTensileStress;
}

public class cgiBeamStressCli
{
    public int iId;
    public List<cgiSectionStressCli> bmSectionStresses;
}

public class cgiNodalResultantCli
{
    public List<double> fVal;          // six values
}

// eigen result for one mode
public class cgiEigenResultCli
{
    public double m_fEigen;           // eigen value ( = circular frequency * circular frequency)
    public double m_fErrorMeasure;   // error measures on the eigen value
    public List<cgiResult6ValCli> m_listEigenVector; // eigen vector
}

// response spectrum result for one mode
public class cgiResponseSpectrumResultCli
{
    public double m_fEigen;           // eigen value ( = circular frequency * circular frequency)
    public List<double> m_fParticipation; // participation factors in global X, Y and Z
    public List<List<cgiResult6ValCli>> m_vModalDisplacement; // modal displacements in global X, Y and Z directions
    public List<List<cgiResult6ValCli>> m_vInertialForce; // nodal inertia forces in global X, Y and Z directions
}

// results of editing commands
public class cgiMergedPairCli
{
    public int oldId;
    public int newId;
}

public class cgiMergedInfoCli
{

```

```

public List<cgiMergedPairCli> mergedNodeIds;
public List<cgiMergedPairCli> mergedBeamIds;
public List<cgiMergedPairCli> mergedShell4Ids;
public List<cgiMergedPairCli> mergedBrickIds;
}

public class cgiIntersectionNodeCli
{
    public int nodeId;
    public int intersectingBeam1;
    public int intersectingBeam2;
}

```

There are a few utility functions within the interface. For example, you can calculate a beam local angle such that the beam local z axis is perpendicular to the plane formed by the two member end points and a 3rd point.

```

cgiPointCli pt1 = new cgiPointCli(-141.53, -372.022, 210.897);
cgiPointCli pt2 = new cgiPointCli(-116.462, -510.254, 152.296);
cgiPointCli pt3 = new cgiPointCli(0, 0, 0);
double fAngleInRadian = solver.getBeamLocalAngleByThirdPoint(pt1, pt2, pt3);

```

We will illustrate the use of the SolverBlaze API by using the following structural models taken from example 4 and B-01 (Plate Patch Test) of Real3D Verification Manual, which is freely available for download from <http://www.cg-inc.com/download/REAL3DVerifications.pdf>

It is highly recommended that you study the “Technical Issues” in the Real3D program manual, which is freely available from here <http://www.cg-inc.com/download/REAL3DManual.pdf>. You will get a good understanding of the theoretical background on which SolverBlaze is based.

Example Model Three (refer to the sample file Example-04.cs included in the library)

The following portal frame has a span of 60 ft and a column height of 24 ft. The beam is vertically loaded with 60 kips placed at 20 ft from the left end of the beam. The right column is vertically loaded with 120 kips. A horizontal load of 6 kips is applied at the joint of the beam and the left column. Each column is modeled with 2 members. The beam is modeled with a single frame element.

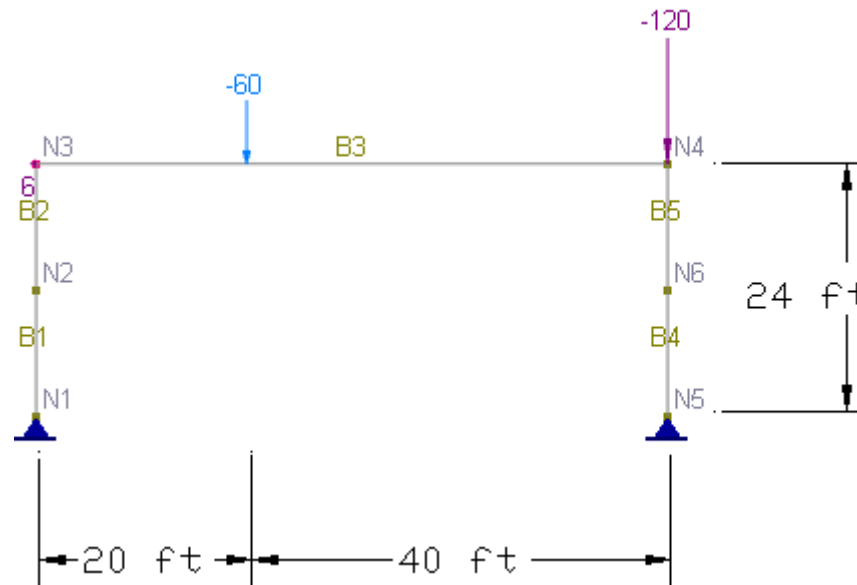
Columns: W10x45, $A = 13.3 \text{ in}^2$, $I_z = 248 \text{ in}^4$

Beam: W27x84, $A = 24.8 \text{ in}^2$, $I_z = 2850 \text{ in}^4$

Material: $E = 2.9e7 \text{ psi}$, $\nu = 0.3$

Perform analysis for the following two cases:

- First order (Linear) elastic analysis
- Second order (P-Delta) elastic analysis



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

```
using System;
using System.Collections.Generic;
using System.Text;
using cgiSolverBlazeCli;

namespace cgiSolverBlazeTestCSharp
{
    public class verify_example4
    {
        // delegate used for the call back.
        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()
        {
            cgiSolverBlazeClass solver = new cgiSolverBlazeClass();
            solver.createStructure();

            StringBuilder sb = new StringBuilder();
            solver.getDefaultTestPath(ref sb);
            string sInputFileName = sb.ToString() + "\\tests\\newModels\\Verify-Example4.r3a";

            // the following should be called after createStructure()
            cgiSolverBlazeClass.ListMessageDelegate ListMsg = new cgiSolverBlazeClass.ListMessageDelegate(Callback);
            cgiSolverBlazeClass.StatusMessageDelegate StatusMsg = new cgiSolverBlazeClass.StatusMessageDelegate(StatusCallback);
            solver.setListMessageFunction(ListMsg);
            solver.setStatusMessageFunction(StatusMsg);

            solver.setModelType((int)cgiModelEnum.kModel_Frame2D);

            // 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();

            // define materials
            List<cgiMaterialCli> listMat = new List<cgiMaterialCli>();
            cgiMaterialCli mat = new cgiMaterialCli();
            mat.setId(1);
            mat.setProperties("Default222", 29000, 0.3, 450);
            listMat.Add(mat);
        }
    }
}
```

```

solver.setMaterials(listMat);

// define sections
List<cgiSectionCli> listSect = new List<cgiSectionCli>();
cgiSectionCli sect1 = new cgiSectionCli();
sect1.setId(1);
sect1.setProperties("W27X84", 24.8, 12.282, 12.7488, 2850, 106, 2.81);
listSect.Add(sect1);
cgiSectionCli sect2 = new cgiSectionCli();
sect2.setId(2);
sect2.setProperties("W27X84", 24.8, 12.282, 12.7488, 2850, 106, 2.81);
listSect.Add(sect2);
cgiSectionCli sect3 = new cgiSectionCli();
sect3.setId(3);
sect3.setProperties("W10X45", 13.3, 3.535, 9.9448, 248, 53.4, 1.51);
listSect.Add(sect3);
solver.setSections(listSect);

// define nodes
List<cgiNodeCli> listNode = new List<cgiNodeCli>();
cgiNodeCli nd1 = new cgiNodeCli();
nd1.setId(1); // nodal id
nd1.setCoordinates(0, 0, 0); // nodal coordinates
listNode.Add(nd1);
cgiNodeCli nd2 = new cgiNodeCli();
nd2.setId(2);
nd2.setCoordinates(0, 12, 0);
listNode.Add(nd2);
cgiNodeCli nd3 = new cgiNodeCli();
nd3.setId(3);
nd3.setCoordinates(0, 24, 0);
listNode.Add(nd3);
cgiNodeCli nd4 = new cgiNodeCli();
nd4.setId(5);
nd4.setCoordinates(60, 0, 0);
listNode.Add(nd4);
cgiNodeCli nd5 = new cgiNodeCli();
nd5.setId(6);
nd5.setCoordinates(60, 12, 0);
listNode.Add(nd5);
cgiNodeCli nd6 = new cgiNodeCli();
nd6.setId(4);
nd6.setCoordinates(60, 24, 0);
listNode.Add(nd6);
solver.setNodes(listNode);

// define beams
List<cgiBeamCli> listBeam = new List<cgiBeamCli>();
cgiBeamCli bm1 = new cgiBeamCli();
bm1.setId(1); // beam id
bm1.setNodes(1, 2); // begin and end node ids of the beam
bm1.setProperties(1, 3, 0); // material id, section id, beam angle in radian

```

```

listBeam.Add(bm1);
cgiBeamCli bm2 = new cgiBeamCli();
bm2.setId(2);
bm2.setNodes(2, 3);
bm2.setProperties(1, 3, 0);
listBeam.Add(bm2);
cgiBeamCli bm3 = new cgiBeamCli();
bm3.setId(3);
bm3.setNodes(3, 4);
bm3.setProperties(1, 2, 0);
listBeam.Add(bm3);
cgiBeamCli bm4 = new cgiBeamCli();
bm4.setId(5);
bm4.setNodes(6, 4);
bm4.setProperties(1, 3, 0);
listBeam.Add(bm4);
cgiBeamCli bm5 = new cgiBeamCli();
bm5.setId(4);
bm5.setNodes(5, 6);
bm5.setProperties(1, 3, 0);
listBeam.Add(bm5);
solver.setBeams(listBeam);

// define supports
List<cgiSupportCli> listSupt = new List<cgiSupportCli>();
cgiSupportCli supt1 = new cgiSupportCli();
supt1.setId(1); // nodal id that this support is on
// six DOFs, 1 for supported, 0 for free; forced settlements and rotations
supt1.setSupportDOFs("111000", 0, 0, 0, 0, 0, 0);
listSupt.Add(supt1);
cgiSupportCli supt2 = new cgiSupportCli();
supt2.setId(5);
supt2.setSupportDOFs("111000", 0, 0, 0, 0, 0, 0);
listSupt.Add(supt2);
solver.setSupports(listSupt);

// define load cases
List<cgiLoadCaseCli> listLoadCase = new List<cgiLoadCaseCli>();
cgiLoadCaseCli loadcase = new cgiLoadCaseCli();
loadcase.setId(1); // load case id
// load case label; type such as DEAD, LIVE etc; report on this load case
loadcase.setLoadCase("Default", "DEAD", true);
listLoadCase.Add(loadcase);
solver.setLoadCases(listLoadCase);

// define load combinations
List<cgiLoadCombCli> listLoadComb = new List<cgiLoadCombCli>();
cgiLoadCombCli loadcomb1 = new cgiLoadCombCli();
loadcomb1.clearLoadCombItem(); // make sure we start clean with this load combination
// load combination label, linear combination (no pdelta effect), want report on this combination
loadcomb1.setLoadComb("Linear", false, true);
loadcomb1.addLoadCombItem("Default", 1.0); // add load case and factor to this load combination

```

```

listLoadComb.Add(loadcomb1);
cgiLoadCombCli loadcomb2 = new cgiLoadCombCli();
loadcomb2.clearLoadCombItem(); // make sure we start clean with this load combination
loadcomb2.setLoadComb("P-Delta", true, true); // load combination label, p-delta, want report on this combination
loadcomb2.addLoadCombItem("Default", 1.0); // add load case and factor to this load combination
listLoadComb.Add(loadcomb2);
solver.setLoadCombinations(listLoadComb);

// each case load corresponds to each load case
// each case load includes nodal loads, point loads, line loads etc in this load case
List<cgiCaseLoadCli> listCaseLoad = new List<cgiCaseLoadCli>();
cgiCaseLoadCli caseload = new cgiCaseLoadCli();
cgiNodalLoadCli ndload1 = new cgiNodalLoadCli();
ndload1.setLoad(4, -120.0, (int)cgiDofEnum.Y);
caseload.addNodalLoad(ndload1);
cgiNodalLoadCli ndload2 = new cgiNodalLoadCli();
ndload2.setLoad(3, 6.0, (int)cgiDofEnum.X); // node id, load magnitude and direction
caseload.addNodalLoad(ndload2);

cgiPointLoadCli ptload = new cgiPointLoadCli();
// member id, load magnitude, distance from member start in percentage/100, load direction
ptload.setLoad(3, (int)cgiLoadSystemEnum.GLOBAL, -60.0, 0.333333, (int)cgiDofEnum.Y);
caseload.addPointLoad(ptload);
listCaseLoad.Add(caseload);
solver.setCaseLoads(listCaseLoad);

// set report options
//solver.setListMessageFunction(0); // do not list message
cgiReportOptionsCli reportOptions = new cgiReportOptionsCli();
reportOptions.selectAll();
reportOptions.bHTML = false;
solver.setReportOptions(reportOptions);

// set analysis options
bool bConsiderBeamShearDeformation = false;
int nMaximumPDeltaIterations = 10;
double fPDeltaToleranceInPercentage = 0.5; // in percentage
int nNumberOfSegmentsForBeamOutput = 20;
bool bUseThinPlate = false;
bool bUseCompatibleModes = true;
int nUseAverageStressMode = 1;
solver.setAnalysisOptions(bConsiderBeamShearDeformation, nMaximumPDeltaIterations, fPDeltaToleranceInPercentage,
    nNumberOfSegmentsForBeamOutput, bUseThinPlate, bUseCompatibleModes, nUseAverageStressMode);

// save the data to a file for possible later use
bool b2 = solver.saveDocument(sInputFileName);

// run static analysis
solver.setSolver((int)cgiSolverEnum.kSolver64);
bool bRun = solver.runStaticAnalysis();
if (!bRun)
{

```

```

    Console.WriteLine("Error running static analysis... " + sInputFileName);
    return;
}

// report
if (!solver.runReport())
{
    Console.WriteLine("Error running report... " + sInputFileName);
    return;
}

// extract results
List<cgiCombResultCli> listCombResult = new List<cgiCombResultCli>();
solver.getStaticResults(ref listCombResult);
// list displacements for all load combinations
StringBuilder sTranslationalDisplacementUnit = new StringBuilder();
StringBuilder sRotationalDisplacementUnit = new StringBuilder();
solver.getUnit(ref sTranslationalDisplacementUnit, (int)cgiUnitEnum.DISPLACEMENT_TRANS);
solver.getUnit(ref sRotationalDisplacementUnit, (int)cgiUnitEnum.DISPLACEMENT_ROTATE);
Console.WriteLine("-----");
Console.WriteLine("----- Verifying Example 4 -----");

Console.WriteLine("Nodal Displacements. Units: {0}, {1}", sRotationalDisplacementUnit, sRotationalDisplacementUnit);
for(int iComb = 0; iComb < listCombResult.Count; iComb++)
{
    Console.WriteLine("Load combination {0}, {1}", iComb + 1, listLoadComb[iComb].szLabel);

    int nDisps = listCombResult[iComb].m_listNdDisp.Count;
    for(int i = 0; i < nDisps; i++) {
        cgiResult6ValCli disp = listCombResult[iComb].m_listNdDisp[i];
        Console.WriteLine("Node- {0}: \t{1:f3}\t{2:f3}\t{3:f3}\t{4:f3}\t{5:f3}\t{6:f3}",
            disp.iId, disp.fVal[(int)cgiDofEnum.X], disp.fVal[(int)cgiDofEnum.Y], disp.fVal[(int)cgiDofEnum.Z],
            disp.fVal[(int)cgiDofEnum.OX], disp.fVal[(int)cgiDofEnum.OY], disp.fVal[(int)cgiDofEnum.OZ]);
    }
    Console.WriteLine("");
}

// list internal forces and moments at beam ends for all load combinations
StringBuilder sForceUnit = new StringBuilder();
StringBuilder sMomentUnit = new StringBuilder();
solver.getUnit(ref sForceUnit, (int)cgiUnitEnum.FORCE);
solver.getUnit(ref sMomentUnit, (int)cgiUnitEnum.MOMENT);
Console.WriteLine("Beam End Forces and Moments. Units: {0}, {1}", sForceUnit, sMomentUnit);
for (int iComb = 0; iComb < listCombResult.Count; iComb++)
{
    Console.WriteLine("Load combination {0}, {1}", iComb + 1, listLoadComb[iComb].szLabel);

    int nBeams = listCombResult[iComb].m_listBmVMD.Count;
    for(int i = 0; i < nBeams; i++) {
        cgiBeamVMDCli bmVMD = listCombResult[iComb].m_listBmVMD[i];

```


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

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

```
cgiSolverBlazeClass solver = new cgiSolverBlazeClass();
solver.createStructure();
```

You can supply three 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);
public delegate int SparseSolverProgressDelegate(IntPtr A_0, IntPtr A_1, IntPtr A_2, int A_3);

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

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

cgiSolverBlazeClass.ListMessageDelegate ListMsg = new cgiSolverBlazeClass.ListMessageDelegate(Callback);
cgiSolverBlazeClass.StatusMessageDelegate StatusMsg = new cgiSolverBlazeClass.StatusMessageDelegate(StatusCallback);
solver.setListMessageFunction(ListMsg);
solver.setStatusMessageFunction(StatusMsg);
```

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 SolverBlaze currently may emit.

```
public enum cgiErrorEnum
{
    kErrorNone,
    kInvalidInput,
    kErrorProcessingElementStiffness,
    kNoMassDefined,
    kErrorConvertingAreaLoadsToLineLoads,
    kTooManyDigitsLostDuringFactorization,
    kSolverError,
    kAbnormalSolverTermination,
    kMustRunResponseSpectrumAnalysis,
    kMustRunFrequencyAnalysisFirst,
    kFileAccessError,
```

```

        kLicenseError,
        kInvalidLoadCombinationIndex,
        kInvalidModeIndex,
        kNoResultAvailable,
        kUnknownError
    }

```

The last error code is cleared (set to `kErrorNone`) at the beginning of each solution. Next you can set the model type. For example:

```

solver.setModelType((int)cgiModelEnum.kModel_Frame2D); // a 2D Frame

```

Other model types can be found by examining the following enumerations

```

public enum cgiModelEnum
{
    kModel_Frame3D,
    kModel_Frame2D,
    kModel_Truss3D,
    kModel_Truss2D,
    kModel_PlateBending,
    kModel_PlaneStress,
    kModel_Brick,
    kModel_Grillage,
    kModel_End,
}

```

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 ();
```

The input includes definitions for materials, sections, nodes, elements, load cases, load combinations, loads (nodal, point, line etc.) in each load case. You can also set different analysis and report options.

Each material is defined by specifying a unique material id (integer number), a material's label, young's modulus, Poisson ratio and weight density. **The material label must be less than 127 characters long and must not contain spaces. This limitation also applies in other labels used in sections, load cases, load combinations etc.** All materials must then be assigned to the structural model by calling `solver.setMaterials()`

```
// define materials
List<cgiMaterialCli> listMat = new List<cgiMaterialCli>();
cgiMaterialCli mat = new cgiMaterialCli();
mat.setId(1);
mat.setProperties("Default222", 29000, 0.3, 450);
listMat.Add(mat);

solver.setMaterials(listMat);
```

Each beam section is defined by specifying a unique section id (integer number), a section label, sectional area, major and minor shear area, major and minor moment of inertia and rotational moment of inertia. **The section label must be less than 127 characters long and must not contain spaces.** All sections must then be assigned to the structural model by calling `solver.setSections ()`. You can set a section to be rigid link by calling `solver.setRigidLink()`.

```
// define sections
List<cgiSectionCli> listSect = new List<cgiSectionCli>();
cgiSectionCli sect1 = new cgiSectionCli();
sect1.setId(1);
sect1.setProperties("W27X84", 24.8, 12.282, 12.7488, 2850, 106, 2.81);
listSect.Add(sect1);
cgiSectionCli sect2 = new cgiSectionCli();
sect2.setId(2);
sect2.setProperties("W27X84", 24.8, 12.282, 12.7488, 2850, 106, 2.81);
listSect.Add(sect2);
cgiSectionCli sect3 = new cgiSectionCli();
sect3.setId(3);
sect3.setProperties("W10X45", 13.3, 3.535, 9.9448, 248, 53.4, 1.51);
listSect.Add(sect3);

solver.setSections(listSect);
```

Note: The beam section properties include area (A), major shear area (A_y), minor shear area (A_z), major inertia (I_z), minor inertia (I_y), and rotational inertia (J) with respect to beam local axes.

Next each node is defined by specifying a unique node id (integer number), x, y and z coordinates. All nodes must then be assigned to the structural model by calling `solver.setNodes()`

```
// define nodes
List<cgiNodeCli> listNode = new List<cgiNodeCli>();
cgiNodeCli nd1 = new cgiNodeCli();
nd1.setId(1); // nodal id
nd1.setCoordinates(0, 0, 0); // nodal coordinates
listNode.Add(nd1);
cgiNodeCli nd2 = new cgiNodeCli();
nd2.setId(2);
nd2.setCoordinates(0, 12, 0);
listNode.Add(nd2);
cgiNodeCli nd3 = new cgiNodeCli();
nd3.setId(3);
nd3.setCoordinates(0, 24, 0);
listNode.Add(nd3);
cgiNodeCli nd4 = new cgiNodeCli();
nd4.setId(5);
nd4.setCoordinates(60, 0, 0);
listNode.Add(nd4);
cgiNodeCli nd5 = new cgiNodeCli();
nd5.setId(6);
nd5.setCoordinates(60, 12, 0);
listNode.Add(nd5);
cgiNodeCli nd6 = new cgiNodeCli();
nd6.setId(4);
nd6.setCoordinates(60, 24, 0);
listNode.Add(nd6);

solver.setNodes(listNode);
```

Next beam (or frame member) is defined by specifying a unique beam id (integer number), start and end node ids, a material id, a section id and a beam local angle in radian. By default, a beam behavior is linear. You can set a beam to be tension only or compression only by calling `solver.setNonlinear()`. You also have the option to modify beam stiffness in I_z , I_y , J , A , A_y , and A_z directions by calling `cgiBeam::setStiffnessModificationFactor()`. All beams must then be assigned to the structural model by calling `solver.setBeams()`.

```
// define beams
List<cgiBeamCli> listBeam = new List<cgiBeamCli>();
cgiBeamCli bm1 = new cgiBeamCli();
bm1.setId(1); // beam id
bm1.setNodes(1, 2); // begin and end node ids of the beam
bm1.setProperties(1, 3, 0); // material id, section id, beam angle in radian
```

```

listBeam.Add(bm1);
cgiBeamCli bm2 = new cgiBeamCli();
bm2.setId(2);
bm2.setNodes(2, 3);
bm2.setProperties(1, 3, 0);
listBeam.Add(bm2);
cgiBeamCli bm3 = new cgiBeamCli();
bm3.setId(3);
bm3.setNodes(3, 4);
bm3.setProperties(1, 2, 0);
listBeam.Add(bm3);
cgiBeamCli bm4 = new cgiBeamCli();
bm4.setId(5);
bm4.setNodes(6, 4);
bm4.setProperties(1, 3, 0);
listBeam.Add(bm4);
cgiBeamCli bm5 = new cgiBeamCli();
bm5.setId(4);
bm5.setNodes(5, 6);
bm5.setProperties(1, 3, 0);
listBeam.Add(bm5);

solver.setBeams(listBeam);

```

Next support (or constraint) is defined by specifying a unique node id (integer number), six constraints for each of the six global degrees of freedom, six forced translations and rotations in each of the six global degrees of freedom. ‘1’ represents a constrained DOF while ‘0’ represents a free DOF. All supports must then be assigned to the structural model by calling `solver.setSupports()`

```

// define supports
List<cgiSupportCli> listSupt = new List<cgiSupportCli>();
cgiSupportCli supt1 = new cgiSupportCli();
supt1.setId(1); // nodal id that this support is on
// six DOFs, 1 for supported, 0 for free; forced settlements and rotations
supt1.setSupportDOFs("111000", 0, 0, 0, 0, 0, 0);
listSupt.Add(supt1);
cgiSupportCli supt2 = new cgiSupportCli();
supt2.setId(5);
supt2.setSupportDOFs("111000", 0, 0, 0, 0, 0, 0);
listSupt.Add(supt2);

solver.setSupports(listSupt);

```

Please note that moment releases and springs follow similar approaches as supports.

Although not needed in this 2D example, SolverBlaze allows you to define rigid diaphragms in a 3D building by calling `solver.setDiaphragms()`.

Next load case is defined by specifying a unique load case id (integer number), a load case label, a load case type and whether to report the load case or not. **The load case label must be less than 127 characters long and must not contain spaces.** All load cases must then be assigned to the structural model by calling `solver.setLoadCases()`. The types of load cases are: "Dead", "Live", "RoofLive", "Snow", "Wind", "Earthquake", "Rain";

```
// define load cases
List<cgiLoadCaseCli> listLoadCase = new List<cgiLoadCaseCli>();
cgiLoadCaseCli loadcase = new cgiLoadCaseCli();
loadcase.setId(1); // load case id
// load case label; type such as DEAD, LIVE etc; report on this load case
loadcase.setLoadCase("Default", "DEAD", true);
listLoadCase.Add(loadcase);

solver.setLoadCases(listLoadCase);
```

Next load combination is defined by specifying a load combination label, p-delta flag and whether to report the load case or not. **The load combination label must be less than 127 characters long and must not contain spaces.** Each load case and its factor is specified by calling `cgiLoadCombCli.addLoadCombItem()`. All load combinations must then be assigned to the structural model by calling `solver.setLoadCombinations()`.

```
// define load combinations
List<cgiLoadCombCli> listLoadComb = new List<cgiLoadCombCli>();
cgiLoadCombCli loadcomb1 = new cgiLoadCombCli();
loadcomb1.clearLoadCombItem(); // make sure we start clean with this load combination
// load combination label, linear combination (no pdelta effect), want report on this combination
loadcomb1.setLoadComb("Linear", false, true);
loadcomb1.addLoadCombItem("Default", 1.0); // add load case and factor to this load combination
listLoadComb.Add(loadcomb1);
cgiLoadCombCli loadcomb2 = new cgiLoadCombCli();
loadcomb2.clearLoadCombItem(); // make sure we start clean with this load combination
loadcomb2.setLoadComb("P-Delta", true, true); // load combination label, p-delta, want report on this combination
loadcomb2.addLoadCombItem("Default", 1.0); // add load case and factor to this load combination
listLoadComb.Add(loadcomb2);

solver.setLoadCombinations(listLoadComb);
```

Each caseload corresponds to all loads for one load case. These loads may be nodal loads, point loads on members, line loads on members etc. A nodal load is defined by specifying a node id, load magnitude and load direction (X, Y, Z, OX, OY or OZ). A nodal load is added to a caseload by calling `cgiCaseLoadCli.addNodalLoad()`. A point load is defined by a beam id, local or global load system, load magnitude, a distance percentage from member start and load direction (X, Y, Z, OX, OY or OZ). A point load is added to a caseload by calling `cgiCaseLoadCli.addPointLoad()`. All caseloads must then be assigned to the structural model by calling `solver.setCaseLoads()`

```

// each case load corresponds to each load case
// each case load includes nodal loads, point loads, line loads etc in this load case
List<cgiCaseLoadCli> listCaseLoad = new List<cgiCaseLoadCli>();
cgiCaseLoadCli caseload = new cgiCaseLoadCli();
cgiNodalLoadCli ndload1 = new cgiNodalLoadCli();
ndload1.setLoad(4, -120.0, (int)cgiDofEnum.Y);
caseload.addNodalLoad(ndload1);
cgiNodalLoadCli ndload2 = new cgiNodalLoadCli();
ndload2.setLoad(3, 6.0, (int)cgiDofEnum.X); // node id, load magnitude and direction
caseload.addNodalLoad(ndload2);

cgiPointLoadCli ptload = new cgiPointLoadCli();
// member id, load magnitude, distance from member start in percentage/100, load direction
ptload.setLoad(3, (int)cgiLoadSystemEnum.GLOBAL, -60.0, 0.333333, (int)cgiDofEnum.Y);
caseload.addPointLoad(ptload);
listCaseLoad.Add(caseload);

solver.setCaseLoads(listCaseLoad);

```

Note: The number of caseloads must equal the number of load cases. In the example, `listLoadCase.Count == listCaseLoad.Count`.

Report options may be specified setting multiple flags in the `cgiReportOptions` defined in `_cgiDefines.h`. The most comprehensive report options can be set by calling `cgiReportOptionsCli.SelectAll()`. Report options must be set to the structural model by calling `solver.setReportOptions()`

```

// set report options
//solver.setListMessageFunction(0); // do not list message
cgiReportOptionsCli reportOptions = new cgiReportOptionsCli();
reportOptions.selectAll();
reportOptions.bHTML = false;
solver.setReportOptions(reportOptions);

```

The last important step is to set the analysis options. These options include whether you want to consider shear deformations from the beam members, p-delta tolerance and iterations, the number of segments to output beam results etc. The analysis options must be assigned the structural model by calling `solver.setAnalysisOptions()`. *If the model contains beam or shell elements with stiffness modifications, then you must call `cgiIStructure::setApplyStiffnessModification()` in order for the stiffness modifications to take effect.*

You can also specify which solver you would like to use by calling `solver.setSolver()`.

```

public enum cgiSolverEnum
{
    kSolver64,
    kSolver128,
    kSolverSparse64,
}

```

```

// set analysis options
bool bConsiderBeamShearDeformation = false;
int nMaximumPDeltaIterations = 10;
double fPDeltaToleranceInPercentage = 0.5; // in percentage
int nNumberOfSegmentsForBeamOutput = 20;
bool bUseThinPlate = false;
bool bUseCompatibleModes = true;
int nUseAverageStressMode = 1;
solver.setAnalysisOptions(bConsiderBeamShearDeformation, nMaximumPDeltaIterations, fPDeltaToleranceInPercentage,
                          nNumberOfSegmentsForBeamOutput, bUseThinPlate, bUseCompatibleModes, nUseAverageStressMode);

// run static analysis
solver.setSolver((int)cgiSolverEnum.kSolver64);

```

Before you perform structural analysis, you can save the input to a file by calling `solver.saveDocument()`. This file can then be opened, visualized or analyzed by Real3D. This can be very useful to verify the correctness of the model input and its results. The actual structural analysis is performed by calling `solver.runStaticAnalysis()`. The program checks for input errors (such as duplicate nodes) prior to performing analysis or report. The error messages are listed in the log file that resides in the same directory as the input file. It is always a good idea to check this file even if no errors are reported. By default, a user can abort the solution process by pressing ESC key. You can customize this behavior by calling `cgiIStructure::setAbortSolutionKey()`.

```

// save the data to a file for possible later use
bool b2 = solver.saveDocument(sInputFileName);

bool bRun = solver.runStaticAnalysis();
if (!bRun)
{
    Console.WriteLine("Error running static analysis... " + sInputFileName);
    return;
}

// report
if (!solver.runReport())
{
    Console.WriteLine("Error running report... " + sInputFileName);
    return;
}

```

The analysis results can be retrieved easily by calling `solver.getStaticResults()` function as illustrated in the following.

```

// extract results
List<cgiCombResultCli> listCombResult = new List<cgiCombResultCli>();
solver.getStaticResults(ref listCombResult);
// list displacements for all load combinations

```

```

StringBuilder sTranslationalDisplacementUnit = new StringBuilder();
StringBuilder sRotationalDisplacementUnit = new StringBuilder();
solver.getUnit(ref sTranslationalDisplacementUnit, (int)cgiUnitEnum.DISPLACEMENT_TRANS);
solver.getUnit(ref sRotationalDisplacementUnit, (int)cgiUnitEnum.DISPLACEMENT_ROTATE);
Console.WriteLine("-----");
Console.WriteLine("----- Verifying Example 4 -----");

Console.WriteLine("Nodal Displacements. Units: {0}, {1}", sRotationalDisplacementUnit, sRotationalDisplacementUnit);
for(int iComb = 0; iComb < listCombResult.Count; iComb++)
{
    Console.WriteLine("Load combination {0}, {1}", iComb + 1, listLoadComb[iComb].szLabel);

    int nDisps = listCombResult[iComb].m_listNdDisp.Count;
    for(int i = 0; i < nDisps; i++) {
        cgiResult6ValCli disp = listCombResult[iComb].m_listNdDisp[i];
        Console.WriteLine("Node- {0}: \t{1:f3}\t{2:f3}\t{3:f3}\t{4:f3}\t{5:f3}\t{6:f3}",
            disp.iId, disp.fVal[(int)cgiDofEnum.X], disp.fVal[(int)cgiDofEnum.Y], disp.fVal[(int)cgiDofEnum.Z],
            disp.fVal[(int)cgiDofEnum.OX], disp.fVal[(int)cgiDofEnum.OY], disp.fVal[(int)cgiDofEnum.OZ]);
    }
    Console.WriteLine("");
}

// list internal forces and moments at beam ends for all load combinations
StringBuilder sForceUnit = new StringBuilder();
StringBuilder sMomentUnit = new StringBuilder();
solver.getUnit(ref sForceUnit, (int)cgiUnitEnum.FORCE);
solver.getUnit(ref sMomentUnit, (int)cgiUnitEnum.MOMENT);
Console.WriteLine("Beam End Forces and Moments. Units: {0}, {1}", sForceUnit, sMomentUnit);
for (int iComb = 0; iComb < listCombResult.Count; iComb++)
{
    Console.WriteLine("Load combination {0}, {1}", iComb + 1, listLoadComb[iComb].szLabel);

    int nBeams = listCombResult[iComb].m_listBmVMD.Count;
    for(int i = 0; i < nBeams; i++) {
        cgiBeamVMDCli bmVMD = listCombResult[iComb].m_listBmVMD[i];

        int[] iSeg = new int[2] { 0, (int)bmVMD.vmd.Count - 1 };
        for(int k = 0; k < 2; k++)
        {
            cgiVMDCli vmd = bmVMD.vmd[iSeg[k]];
            string sEnd = (k==0)? "Start" : "End";
            Console.WriteLine("Beam {0}: {1}, \t{2:f3}\t{3:f3}\t{4:f3}\t{5:f3}\t{6:f3}\t{7:f3}",
                bmVMD.iId, sEnd, vmd.fVMD[(int)cgiDofEnum.X], vmd.fVMD[(int)cgiDofEnum.Y], vmd.fVMD[(int)cgiDofEnum.Z],
                vmd.fVMD[(int)cgiDofEnum.OX], vmd.fVMD[(int)cgiDofEnum.OY], vmd.fVMD[(int)cgiDofEnum.OZ]);
        }
    }
    Console.WriteLine("");
}
}

```

You can also retrieve analysis results for a single load combination as shown in the following.

```

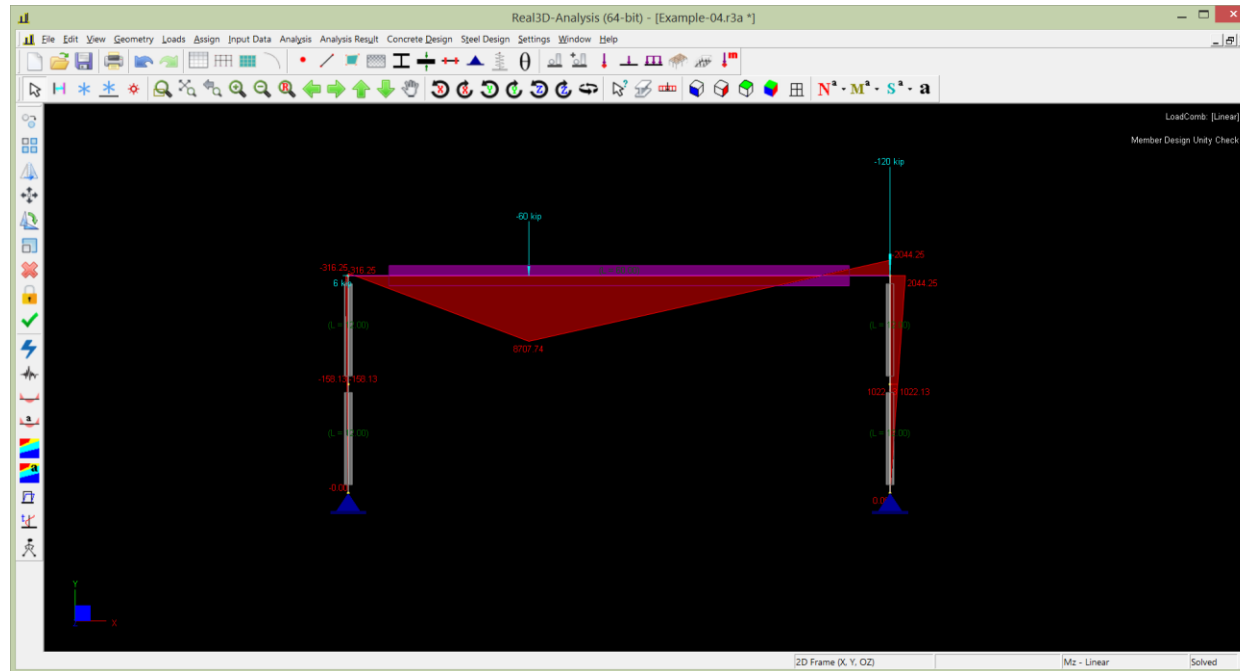
List<cgiCombResultCli> listCombResult2 = new List<cgiCombResultCli>();
int loadCombinationIndex = 1;
bool bSuccess = solver.getStaticResultsForSingleLoadCombination(ref listCombResult2, loadCombinationIndex);
if (bSuccess)
{
    Console.WriteLine("Single Load combination {0}, {1}", loadCombinationIndex + 1,
        listLoadComb[loadCombinationIndex].szLabel);
    int nDisps2 = listCombResult2[0].m_listNdDisp.Count;
    for (int i = 0; i < nDisps2; i++)
    {
        cgiResult6ValCli disp = listCombResult2[0].m_listNdDisp[i];
        Console.WriteLine("Node- {0}: \t{1:f3}\t{2:f3}\t{3:f3}\t{4:f3}\t{5:f3}\t{6:f3}", disp.iId,
            disp.fVal[(int)cgiDofEnum.X], disp.fVal[(int)cgiDofEnum.Y],
            disp.fVal[(int)cgiDofEnum.Z], disp.fVal[(int)cgiDofEnum.OX],
            disp.fVal[(int)cgiDofEnum.OY], disp.fVal[(int)cgiDofEnum.OZ]);
    }
    Console.WriteLine();
}
}

```

For your references, the following table shows the critical nodal displacements and member internal moments.

		Real3D	Reference
Linear	Maximum Displacement (in)	4.387	4.4
	Max + moment in beam (in-kips)	8707.7	8708
	Max – moment in beam (in-kips)	2044.3	2044
P-Delta	Maximum Displacement (in)	8.26	8.1
	Max + moment in beam (in-kips)	9079.4	9078
	Max – moment in beam (in-kips)	2663.3	2661

You can use Real3D to view the model, perform analysis and view its results (shown below). You may need to zoom extent from View ->Zoom->Zoom Extent menu and adjust graphics scales from Settings->Graphics Scales.



Example Model Four (refer to the sample file verify-plate-patch-test.cs sample included in the library)

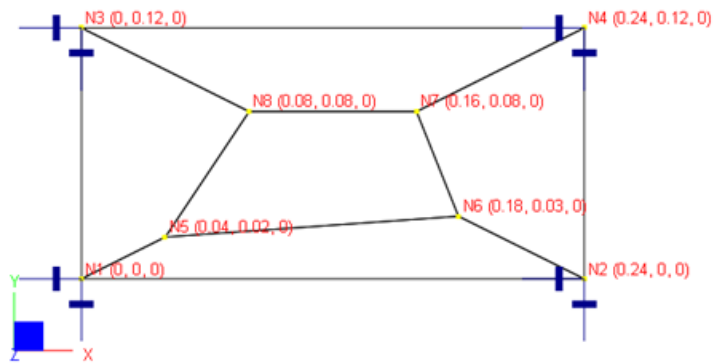
A plate of size 0.12 x 0.24 in is subjected to forced displacements at the four corners as shown below. The boundary conditions are:

$$w = 1.0e^{-3}(x^2 + xy + y^2) / 2$$

$$\theta_x = \frac{\partial w}{\partial y} = 1.0e^{-3}(y + x/2) ; \theta_y = -\frac{\partial w}{\partial x} = 1.0e^{-3}(-x - y/2)$$

Material properties: $E = 1.0e6$ psi, $\nu = 0.25$

Geometry: nodal coordinates are shown in the parenthesis below, thickness $t = 0.001$ in



Forced displacements on boundary nodes (units: displacement – in; rotation – rad)

Boundary Nodes	Displacement Dz	Rotation D _{ox}	Rotation D _{oy}
1	0	0	0
2	2.88e-5	1.20e-4	-2.40e-4
3	7.20e-6	1.20e-4	-6.00e-5
4	5.04e-5	2.40e-4	-3.00e-4

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

```
using System;
using System.Collections.Generic;
using System.Text;
using cgiSolverBlazeCli;

namespace cgiSolverBlazeTestCSharp
{
    public class verify_plate_patch_test
    {
        // delegate used for the call back.
        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()
        {
            cgiSolverBlazeClass solver = new cgiSolverBlazeClass();
            solver.createStructure();

            StringBuilder sb = new StringBuilder();
            solver.getDefaultTestPath(ref sb);
            string sInputFileName = sb.ToString() + "\\tests\\newModels\\Verify-PlatePatchTest.r3a";

            // the following should be called after createStructure()
            cgiSolverBlazeClass.ListMessageDelegate ListMsg = new cgiSolverBlazeClass.ListMessageDelegate(Callback);
            cgiSolverBlazeClass.StatusMessageDelegate StatusMsg = new cgiSolverBlazeClass.StatusMessageDelegate(StatusCallback);
            solver.setListMessageFunction(ListMsg);
            solver.setStatusMessageFunction(StatusMsg);

            solver.setModelType((int)cgiModelEnum.kModel_PlateBending);

            // in, lb and rad
            solver.setConsistentEnglishUnits();

            // define materials
            List<cgiMaterialCli> listMat = new List<cgiMaterialCli>();
            cgiMaterialCli mat = new cgiMaterialCli();
            mat.setId(1);
            mat.setProperties("Default", 1e+006, 0.25, 0.28);
            listMat.Add(mat);
            solver.setMaterials(listMat);
        }
    }
}
```

```

// define sections
List<cgiThicknessCli> listThick = new List<cgiThicknessCli>();
cgiThicknessCli thick = new cgiThicknessCli();
thick.setId(1);
thick.setProperties("Default", 0.001); // thickness label, thickness
listThick.Add(thick);
solver.setThicknesses(listThick);

// define nodes
double[][] coordinates = new double[][]{new double[]{0.00, 0.00},new double[]{0.24, 0.00},
                                         new double[]{0.00, 0.12},new double[]{0.24, 0.12},
                                         new double[]{0.04, 0.02},new double[]{0.18, 0.03},
                                         new double[]{0.16, 0.08},new double[]{0.08, 0.08}};

List<cgiNodeCli> listNode = new List<cgiNodeCli>();
for(int i = 0; i < 8; i++)
{
    cgiNodeCli nd = new cgiNodeCli();
    nd.setId(i+ 1); // nodal id
    nd.setCoordinates(coordinates[i][0], coordinates[i][1], 0); // nodal coordinates
    listNode.Add(nd);
}
solver.setNodes(listNode);

// define shells
int[][] connectivity = {new int[]{1, 2, 6, 5},new int[]{6, 7, 8, 5},new int[]{2, 4, 7, 6},
                       new int[]{4, 3, 8, 7},new int[]{5, 8, 3, 1}};
List<cgiShell4Cli> listShell = new List<cgiShell4Cli>();
for (int i = 0; i < 5; i++)
{
    cgiShell4Cli shell = new cgiShell4Cli() ;
    shell.setId(i + 1);
    shell.setNodes(connectivity[i][0], connectivity[i][1], connectivity[i][2], connectivity[i][3]);
    shell.setProperties(1, 1, 0.0);
    listShell.Add(shell);
}
solver.setShell4s(listShell);

// define supports
double[][] fForcedDisplacement = {new double[]{0, 0, 0, 0, 0, 0},
                                   new double[]{0, 0, 2.88e-005, 0.00012, -0.00024, 0},
                                   new double[]{0, 0, 7.2e-006, 0.00012, -6e-005, 0},
                                   new double[]{0, 0, 5.04e-005, 0.00024, -0.0003, 0}};

List<cgiSupportCli> listSupt = new List<cgiSupportCli>();
for (int i = 0; i < 4; i++)
{
    cgiSupportCli supt = new cgiSupportCli();
    supt.setId(i + 1); // nodal id that this support is on
    // six DOFs, 1 for supported, 0 for free; forced settlements and rotations
    supt.setSupportDOFs("001110", fForcedDisplacement[i][0], fForcedDisplacement[i][1], fForcedDisplacement[i][2],
                       fForcedDisplacement[i][3],fForcedDisplacement[i][4],fForcedDisplacement[i][5]);
    listSupt.Add(supt);
}

```

```

}
solver.setSupports(listSupt);

// define load cases
List<cgiLoadCaseCli> listLoadCase = new List<cgiLoadCaseCli>();
cgiLoadCaseCli loadcase = new cgiLoadCaseCli();
loadcase.setId(1); // load case id
loadcase.setLoadCase("Default", "DEAD", true); // load case label; type such as DEAD, LIVE etc; report on this load case
listLoadCase.Add(loadcase);
solver.setLoadCases(listLoadCase);

// define load combinations
List<cgiLoadCombCli> listLoadComb = new List<cgiLoadCombCli>();
cgiLoadCombCli loadcomb1 = new cgiLoadCombCli();
loadcomb1.clearLoadCombItem(); // make sure we start clean with this load combination
// load combination label, linear combination (no pdelta effect), want report on this combination
loadcomb1.setLoadComb("Patch-Test", false, true);
loadcomb1.addLoadCombItem("Default", 1.0); // add load case and factor to this load combination
listLoadComb.Add(loadcomb1);
solver.setLoadCombinations(listLoadComb);

// note: there is no actual loads for patch test
// or you can say force support displacements are a form of loads

// set report options
cgiReportOptionsCli reportOptions = new cgiReportOptionsCli();
reportOptions.selectAll();
reportOptions.bHTML = false;
solver.setReportOptions(reportOptions);

// set analysis options
bool bConsiderBeamShearDeformation = false;
int nMaximumPDeltaIterations = 10;
double fPDeltaToleranceInPercentage = 0.5; // in percentage
int nNumberOfSegmentsForBeamOutput = 20;
bool bUseThinPlate = false;
bool bUseCompatibleModes = true;
int nUseAverageStressMode = 1;
solver.setAnalysisOptions(bConsiderBeamShearDeformation, nMaximumPDeltaIterations, fPDeltaToleranceInPercentage,
                          nNumberOfSegmentsForBeamOutput, bUseThinPlate, bUseCompatibleModes, nUseAverageStressMode);

// save the data to a file for possible later use
bool b2 = solver.saveDocument(sInputFileName);

// run static analysis
bool bRun = solver.runStaticAnalysis();
if (!bRun)
{
    Console.WriteLine("Error running static analysis... " + sInputFileName);
    return;
}

```

```

// report
if (!solver.runReport())
{
    // report will be saved in ttestt.htm file
    Console.WriteLine("Error running report... " + sInputFileName);
    return;
}

// extract results
List<cgiCombResultCli> listCombResult = new List<cgiCombResultCli>();
solver.getStaticResults(ref listCombResult);
// list displacements for all load combinations
StringBuilder sTranslationalDisplacementUnit = new StringBuilder();
StringBuilder sRotationalDisplacementUnit = new StringBuilder();
solver.getUnit(ref sTranslationalDisplacementUnit, (int)cgiUnitEnum.DISPLACEMENT_TRANS);
solver.getUnit(ref sRotationalDisplacementUnit, (int)cgiUnitEnum.DISPLACEMENT_ROTATE);
Console.WriteLine("-----");
Console.WriteLine("----- Verifying Plate Patch Test -----");

Console.WriteLine("Nodal Displacements. Units: {0}, {1}", sRotationalDisplacementUnit, sRotationalDisplacementUnit);
for (int iComb = 0; iComb < listCombResult.Count; iComb++)
{
    Console.WriteLine("Load combination {0}, {1}", iComb + 1, listLoadComb[iComb].szLabel);

    int nDisps = listCombResult[iComb].m_listNdDisp.Count;
    for (int i = 0; i < nDisps; i++)
    {
        cgiResult6ValCli disp = listCombResult[iComb].m_listNdDisp[i];
        Console.WriteLine("Node- {0}: \t{1:e3}\t{2:e3}\t{3:e3}\t{4:e3}\t{5:e3}\t{6:e3}",
            disp.iId, disp.fVal[(int)cgiDofEnum.X], disp.fVal[(int)cgiDofEnum.Y], disp.fVal[(int)cgiDofEnum.Z],
            disp.fVal[(int)cgiDofEnum.OX], disp.fVal[(int)cgiDofEnum.OY], disp.fVal[(int)cgiDofEnum.OZ]);
    }
    Console.WriteLine("");
}

// list internal forces and moments at beam ends for all load combinations
StringBuilder sLineForceUnit = new StringBuilder();
StringBuilder sLineMomentUnit = new StringBuilder();
solver.getUnit(ref sLineForceUnit, (int)cgiUnitEnum.FORCE_LINE);
solver.getUnit(ref sLineMomentUnit, (int)cgiUnitEnum.MOMENT_LINE);
Console.WriteLine("Plate Internal Moments Mxx, Myy, Mxy and Shears Vxx, Vyy. Units: {0}, {1}", sLineForceUnit, sLineMomentUnit);
for (int iComb = 0; iComb < listCombResult.Count; iComb++)
{
    Console.WriteLine("Load combination {0}, {1}", iComb + 1, listLoadComb[iComb].szLabel);

    int nShell4s = listCombResult[iComb].m_listShell4Stress.Count;
    for (int i = 0; i < nShell4s; i++) {
        cgiShellStressCli stress = listCombResult[iComb].m_listShell4Stress[i];

        cgiShell4Cli shell4 = new cgiShell4Cli();
        solver.getSingleShell4(ref shell4, stress.iId);
        for (int k = 0; k < 5; k++)

```

```

{
    string szNode = "";
    if(k == 0)
    {
        szNode = "Center";
    }
    else
    {
        int iNode = shell4.iNode1;
        if(k == 2)
        {
            iNode = shell4.iNode2;
        }
        else if(k == 3)
        {
            iNode = shell4.iNode3;
        }
        else if(k == 4)
        {
            iNode = shell4.iNode4;
        }
        szNode = String.Format("Node-{0}", iNode);
    }

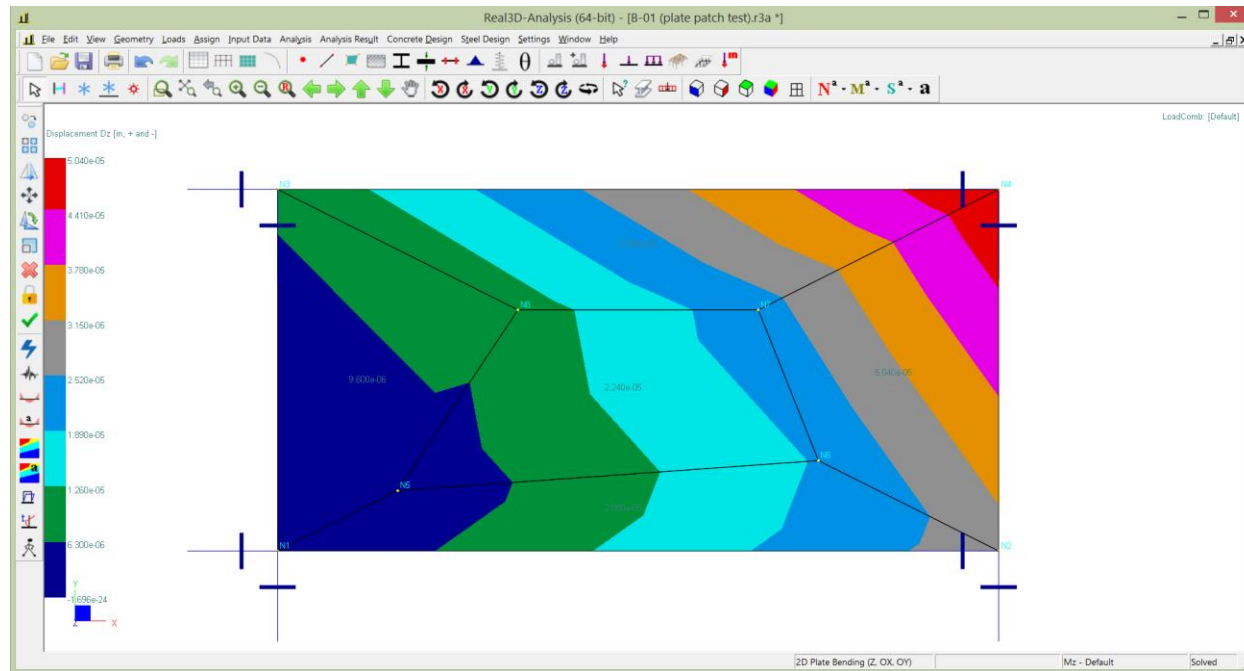
    string szLine = String.Format("Plate-{0}, {1}, {2:e3}, {3:e3}, {4:e3}, {5:e3}, {6:e3}",
        shell4.iId, szNode, stress.fStress_b[k][0], stress.fStress_b[k][1],
        stress.fStress_b[k][2], stress.fStress_b[k][3], stress.fStress_b[k][4]);
    Console.WriteLine(szLine);
} // for(int k = 0; k < 5; k++)
Console.WriteLine("");

} // for(int i = 0; i < nShell4s; i++)
Console.WriteLine("");
}
}
}

```

Please refer the detailed explanations for Example Model Three in the .NET Interface section as the procedures involved are similar.

You can use Real3D to view the model, perform analysis and view its results (shown below). You may need to zoom extent from View ->Zoom->Zoom Extent menu and adjust graphics scales from Settings->Graphics Scales.



.NET Interface through P/Invoke

The .NET interface through P/Invoke is supported by exporting a set of C functions in a native Windows DLL called `cgiSolverBlaze.dll`. The exported interface functions are contained in a header file called `cgiStructure_Interop.h`. It is important to point out that P/Invoke can be used on both x64 and ARM64 target platforms.

```
#include "_cgiIStructure.h"

#include <vector>

extern "C"
{
    //-----
    // Creation / Deletion
    //-----
    CGISOLVERBLAZE_API cgiIStructure* cgiCreateStructure();
    CGISOLVERBLAZE_API void cgiDeleteStructure(cgiIStructure* pStructure);

    //-----
    // Callback setters
    //-----
    CGISOLVERBLAZE_API void cgiSetListMessageFunction(cgiIStructure* pStructure, fnLISTMSG fnListMsg);
    CGISOLVERBLAZE_API void cgiSetStatusMessageFunction(cgiIStructure* pStructure, fnSTATUSMSG fnStatusMsg);
    CGISOLVERBLAZE_API void cgiSetSparseSolverProgressFunction(cgiIStructure* pStructure, fnMKLPROGRESS fnMklProgress);

    //-----
    // Paths, Identifiers, Strings
    //-----
    CGISOLVERBLAZE_API void cgiGetExePath(const cgiIStructure* pStructure, TCHAR exePath[], int size);
    CGISOLVERBLAZE_API void cgiGetDefaultTestPath(const cgiIStructure* pStructure, TCHAR testPath[], int size);
    CGISOLVERBLAZE_API void cgiSetProjPath(cgiIStructure* pStructure, const TCHAR projPath[]);

    CGISOLVERBLAZE_API void cgiGetProjPath(const cgiIStructure* pStructure, TCHAR projPath[]);
    CGISOLVERBLAZE_API void cgiSetModelName(cgiIStructure* pStructure, const TCHAR modelName[]);
    CGISOLVERBLAZE_API void cgiGetModelName(const cgiIStructure* pStructure, TCHAR modelName[]);
    CGISOLVERBLAZE_API void cgiSetDesignCompany(cgiIStructure* pStructure, const TCHAR designCompany[]);
    CGISOLVERBLAZE_API void cgiGetDesignCompany(const cgiIStructure* pStructure, TCHAR designCompany[]);
    CGISOLVERBLAZE_API void cgiSetEngineer(cgiIStructure* pStructure, const TCHAR engineer[]);
    CGISOLVERBLAZE_API void cgiGetEngineer(const cgiIStructure* pStructure, TCHAR engineer[]);
    CGISOLVERBLAZE_API void cgiSetNotes(cgiIStructure* pStructure, const TCHAR notes[]);
    CGISOLVERBLAZE_API void cgiGetNotes(const cgiIStructure* pStructure, TCHAR notes[]);

    //-----
    // Unit handling
    //-----
    CGISOLVERBLAZE_API void cgiSetStandardEnglishUnits(cgiIStructure* pStructure);
    CGISOLVERBLAZE_API void cgiSetStandardMetricUnits(cgiIStructure* pStructure);
}
```

```

CGISOLVERBLAZE_API void cgiSetConsistentEnglishUnits(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiSetConsistentMetricUnits(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiGetUnit(const cgiIStructure* pStructure, TCHAR szUnit[], int nUnit);

//-----
// Clear entire model
//-----
CGISOLVERBLAZE_API void cgiClearModel(cgiIStructure* pStructure);

//-----
// Model type
//-----
CGISOLVERBLAZE_API void cgiSetModelType(cgiIStructure* pStructure, int nModelType);
CGISOLVERBLAZE_API int cgiGetModelType(const cgiIStructure* pStructure);

//-----
// Suppressed DOFs
//-----
CGISOLVERBLAZE_API void cgiSetSuppressedDOFs(cgiIStructure* pStructure, const bool bSuppress[6]);
CGISOLVERBLAZE_API void cgiGetSuppressedDOFs(const cgiIStructure* pStructure, bool bSuppress[6]);

//-----
// Abort solution key
//-----
CGISOLVERBLAZE_API void cgiSetAbortSolutionKey(cgiIStructure* pStructure, int key);
CGISOLVERBLAZE_API int cgiGetAbortSolutionKey(const cgiIStructure* pStructure);

//-----
// Analysis Options
//-----
CGISOLVERBLAZE_API void cgiSetAnalysisOptions(
    cgiIStructure* pStructure,
    bool bConsiderBeamShearDeformation,
    int nMaximumPDeltaIterations,
    double fPDeltaToleranceInPercentage,
    int nNumberOfSegmentsForBeamOutput,
    bool bUseThinPlate,
    bool bUseCompatibleModes,
    int nUseAverageStress);
CGISOLVERBLAZE_API void cgiGetAnalysisOptions(
    const cgiIStructure* pStructure,
    bool* bConsiderBeamShearDeformation,
    int* nMaximumPDeltaIterations,
    double* fPDeltaToleranceInPercentage,
    int* nNumberOfSegmentsForBeamOutput,
    bool* bUseThinPlate,
    bool* bUseCompatibleModes,
    int* nUseAverageStress);
}

//-----
// Frequency Analysis Options

```

```

//-----
CGISOLVERBLAZE_API void cgiSetFrequencyAnalysisOptions(
    cgiIStructure* pStructure,
    int nEigenNumber,
    double fEigenVlaueTolerance,
    int nMaximumSubspaceIterations,
    int nIterationVectors,
    bool bConvertLoadToMass,
    int nGravityDirection,
    int nLoadCombinationForMass);

CGISOLVERBLAZE_API void cgiGetFrequencyAnalysisOptions(
    const cgiIStructure* pStructure,
    int* nEigenNumber,
    double* fEigenVlaueTolerance,
    int* nMaximumSubspaceIterations,
    int* nIterationVectors,
    bool* bConvertLoadToMass,
    int* nGravityDirection,
    int* nLoadCombinationForMass);

//-----
// Response Spectrum Analysis Options
//-----
CGISOLVERBLAZE_API void cgiSetResponseSpectrumAnalysisOptions(
    cgiIStructure* pStructure,
    cgiSolverBlazeNamespace::cgiSpectrum spectrums[3],
    double fDampingRatio,
    int combinationMethod,
    double fSpectrumDirectionalFactor[3],
    bool bUseDominantModeForSignage);
CGISOLVERBLAZE_API void cgiGetResponseSpectrumAnalysisOptions(
    cgiIStructure* pStructure,
    cgiSolverBlazeNamespace::cgiSpectrum spectrums[3],
    double* fDampingRatio,
    int* combinationMethod,
    double fSpectrumDirectionalFactor[3],
    bool* bUseDominantModeForSignage);

//-----
// Various flags and numerical parameters
//-----
CGISOLVERBLAZE_API void cgiSetApplyStiffnessModification(cgiIStructure* pStructure, bool bApply);
CGISOLVERBLAZE_API bool cgiGetApplyStiffnessModification(const cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiSetTolerance(cgiIStructure* pStructure, double fTolerance);
CGISOLVERBLAZE_API double cgiGetTolerance(const cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiSetFictitiousOzFactor(cgiIStructure* pStructure, double fFictitiousOzFactor);
CGISOLVERBLAZE_API double cgiGetFictitiousOzFactor(const cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiSetUseOoc(cgiIStructure* pStructure, bool bUseOoc);
CGISOLVERBLAZE_API bool cgiGetUseOoc(const cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiSetEpsilon(cgiIStructure* pStructure, double fEpsilon);
CGISOLVERBLAZE_API double cgiGetEpsilon(const cgiIStructure* pStructure);

```

```

CGISOLVERBLAZE_API void cgiSetStressLocation(cgiIStructure* pStructure, int nStressLocation);
CGISOLVERBLAZE_API int cgiGetStressLocation(const cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiSetGravityFactor(cgiIStructure* pStructure, double fGravityFactor);
CGISOLVERBLAZE_API double cgiGetGravityFactor(const cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiSetGravityDirection(cgiIStructure* pStructure, int nGravityDirection);
CGISOLVERBLAZE_API int cgiGetGravityDirection(const cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiSetGravityLoadCase(cgiIStructure* pStructure, int nGravityLoadCase);
CGISOLVERBLAZE_API int cgiGetGravityLoadCase(const cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiSetUseZAsVerticalAxis(cgiIStructure* pStructure, bool useZAsVerticalAxis);
CGISOLVERBLAZE_API bool cgiGetUseZAsVerticalAxis(const cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiSetSolver(cgiIStructure* pStructure, int iSolver);
CGISOLVERBLAZE_API int cgiGetSolver(const cgiIStructure* pStructure);

//-----
// deleteMemoryArray Overloads
//-----
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_Int(const cgiIStructure* pStructure, int* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_Material(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiMaterial* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_Section(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiSection* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_Thickness(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiThickness* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_Support(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiSupport* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_Spring(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiSpring* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_CoupledSpring(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiCoupledSpring* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_Release(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiRelease* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_Diaphragm(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiDiaphragm* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_MultiDofConstraint(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiMultiDofConstraint* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_Node(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiNode* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_Beam(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiBeam* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_Shell4(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiShell4* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_Brick(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiBrick* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_LoadCase(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiLoadCase* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_LoadCombination(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiLoadCombination* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_LoadCaseLoad(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiLoadCaseLoad* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_NodalMass(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiNodalMass* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_CombinationResult(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiCombinationResult* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_EigenValueVectorResult(const cgiIStructure* pStructure,
    cgiSolverBlazeNamespace::cgiEigenValueVectorResult* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_ResponseSpectrumResult(const cgiIStructure* pStructure,
    cgiSolverBlazeNamespace::cgiResponseSpectrumResult* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_EnvelopeValue6(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiEnvelopeValue6* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_EnvelopeBmVMD(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiEnvelopeBmVMD* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_IntersectionNode(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiIntersectionNode* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_MergedInfo(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiMergedInfo* p);
CGISOLVERBLAZE_API void cgiDeleteMemoryArray_BeamStress(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiBeamStress* p);

//-----
// Set / Get Materials, Sections, Thicknesses, etc.
//-----
CGISOLVERBLAZE_API void cgiSetMaterials(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiMaterial* vMat, int count);
CGISOLVERBLAZE_API void cgiGetMaterials(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiMaterial** vMat, int* count);
CGISOLVERBLAZE_API void cgiSetSections(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiSection* vSect, int count);
CGISOLVERBLAZE_API void cgiGetSections(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiSection** vSect, int* count);

```

```

CGISOLVERBLAZE_API void cgiSetThicknesses(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiThickness* vThick, int count);
CGISOLVERBLAZE_API void cgiGetThicknesses(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiThickness** vThick, int* count);
CGISOLVERBLAZE_API void cgiSetSupports(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiSupport* vSupt, int count);
CGISOLVERBLAZE_API void cgiGetSupports(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiSupport** vSupt, int* count);
CGISOLVERBLAZE_API void cgiSetNodalSprings(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiSpring* vNdSpring, int count);
CGISOLVERBLAZE_API void cgiGetNodalSprings(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiSpring** vNdSpring, int* count);
CGISOLVERBLAZE_API void cgiSetCoupledSprings(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiCoupledSpring* vCoupledSpring, int count);
CGISOLVERBLAZE_API void cgiGetCoupledSprings(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiCoupledSpring** vCoupledSpring, int* count);
CGISOLVERBLAZE_API void cgiSetLineSprings(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiSpring* vLnSpring, int count);
CGISOLVERBLAZE_API void cgiGetLineSprings(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiSpring** vLnSpring, int* count);
CGISOLVERBLAZE_API void cgiSetSurfaceSprings(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiSpring* vShell4Spring, int count);
CGISOLVERBLAZE_API void cgiGetSurfaceSprings(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiSpring** vShell4Spring, int* count);
CGISOLVERBLAZE_API void cgiSetMomentReleases(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiRelease* vRels, int count);
CGISOLVERBLAZE_API void cgiGetMomentReleases(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiRelease** vRels, int* count);
CGISOLVERBLAZE_API void cgiSetDiaphragms(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiDiaphragm* vDiaphragm, int count);
CGISOLVERBLAZE_API void cgiGetDiaphragms(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiDiaphragm** vDiaphragm, int* count);
CGISOLVERBLAZE_API void cgiSetDiaphragmOptions(cgiIStructure* pStructure, double fDiaphragmStiffnessFactor, bool bConsiderDiaphragm);
CGISOLVERBLAZE_API void cgiGetDiaphragmOptions(const cgiIStructure* pStructure, double* fDiaphragmStiffnessFactor, bool* bConsiderDiaphragm);
CGISOLVERBLAZE_API void cgiSetMultiDofConstraints(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiMultiDofConstraint* vMultiDofConstraint,
int count);
CGISOLVERBLAZE_API void cgiGetMultiDofConstraints(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiMultiDofConstraint** vMultiDofConstraint,
int* count);

//-----
// Nodes, Beams, Shell4s, Bricks
//-----
CGISOLVERBLAZE_API void cgiSetNodes(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiNode* vNd, int count);
CGISOLVERBLAZE_API void cgiGetNodes(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiNode** vNd, int* count);
CGISOLVERBLAZE_API void cgiSetBeams(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiBeam* vBm, int count);
CGISOLVERBLAZE_API void cgiGetBeams(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiBeam** vBm, int* count);
CGISOLVERBLAZE_API void cgiSetShell4s(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiShell4* vShell4, int count);
CGISOLVERBLAZE_API void cgiGetShell4s(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiShell4** vShell4, int* count);
CGISOLVERBLAZE_API void cgiSetBricks(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiBrick* vBrick, int count);
CGISOLVERBLAZE_API void cgiGetBricks(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiBrick** vBrick, int* count);

//-----
// Single-element getters
//-----
CGISOLVERBLAZE_API void cgiGetSingleNode(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiNode* node, int nId);
CGISOLVERBLAZE_API void cgiGetSingleBeam(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiBeam* beam, int nId);
CGISOLVERBLAZE_API void cgiGetSingleShell4(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiShell4* shell4, int nId);
CGISOLVERBLAZE_API void cgiGetSingleBrick(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiBrick* brick, int nId);

//-----
// Local axes calculations
//-----
CGISOLVERBLAZE_API double cgiGetBeamLocalAngleByThirdPoint(
    cgiIStructure* pStructure,
    cgiSolverBlazeNamespace::cgiPoint* beamStartPt,
    cgiSolverBlazeNamespace::cgiPoint* beamEndPt,
    cgiSolverBlazeNamespace::cgiPoint* thirdPt);
CGISOLVERBLAZE_API void cgiGetBeamLocalAxes(

```

```

const cgiIStructure* pStructure,
cgiSolverBlazeNamespace::cgiPoint* xAxis,
cgiSolverBlazeNamespace::cgiPoint* yAxis,
cgiSolverBlazeNamespace::cgiPoint* zAxis,
const cgiSolverBlazeNamespace::cgiPoint* point1,
const cgiSolverBlazeNamespace::cgiPoint* point2,
double fGamma);
CGISOLVERBLAZE_API void cgiGetShellLocalAxes(
const cgiIStructure* pStructure,
cgiSolverBlazeNamespace::cgiPoint* xAxis,
cgiSolverBlazeNamespace::cgiPoint* yAxis,
cgiSolverBlazeNamespace::cgiPoint* zAxis,
const cgiSolverBlazeNamespace::cgiPoint* point1,
const cgiSolverBlazeNamespace::cgiPoint* point2,
const cgiSolverBlazeNamespace::cgiPoint* point3,
const cgiSolverBlazeNamespace::cgiPoint* point4,
double fGamma);
CGISOLVERBLAZE_API bool cgiMatchShellLocalxAxis(cgiIStructure* pStructure, int sourceShellId, int targetShellId);
CGISOLVERBLAZE_API bool cgiMatchShellLocalzAxis(cgiIStructure* pStructure, int sourceShellId, int targetShellId);

//-----
// Load cases, combos, loads, masses
//-----
CGISOLVERBLAZE_API void cgiSetLoadCases(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiLoadCase* vLoadCase, int count);
CGISOLVERBLAZE_API void cgiGetLoadCases(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiLoadCase** vLoadCase, int* count);
CGISOLVERBLAZE_API void cgiSetLoadCombinations(cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiLoadCombination** vLoadCombArray, int count);
CGISOLVERBLAZE_API void cgiGetLoadCombinations(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiLoadCombination** vLoadComb, int* count);
CGISOLVERBLAZE_API void cgiSetCaseLoads(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiLoadCaseLoad** vCaseLoad, int count);
CGISOLVERBLAZE_API void cgiGetCaseLoads(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiLoadCaseLoad** vCaseLoad, int* count);
CGISOLVERBLAZE_API void cgiSetNodalMasses(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiNodalMass* vNdMass, int count);
CGISOLVERBLAZE_API void cgiGetNodalMasses(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiNodalMass** vNdMass, int* count);
CGISOLVERBLAZE_API void cgiGetCalculatedNodalMasses(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiNodalMass** vNdCombMass, int* count);

//-----
// Conversions
//-----
CGISOLVERBLAZE_API void cgiConvertLocalLoadsToGlobalLoads(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiConvertAreaLoadsToLineLoads(cgiIStructure* pStructure);

//-----
// Reports, results, solving
//-----
CGISOLVERBLAZE_API void cgiSetReportOptions(cgiIStructure* pStructure, const cgiSolverBlazeNamespace::cgiReportOptions* reportOptions);
CGISOLVERBLAZE_API void cgiGetReportOptions(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiReportOptions* reportOptions);
CGISOLVERBLAZE_API int cgiGetEquations(const cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiGetStaticResults(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiCombinationResult** result, int* count);
CGISOLVERBLAZE_API bool cgiGetStaticResultsForSingleLoadCombination(const cgiIStructure* pStructure,
cgiSolverBlazeNamespace::cgiCombinationResult** result, int loadCombinationIndex);
CGISOLVERBLAZE_API bool cgiGetBeamStressesForSingleLoadCombination(const cgiIStructure* pStructure,
cgiSolverBlazeNamespace::cgiBeamStress** result, int* count, int loadCombinationIndex);
CGISOLVERBLAZE_API bool cgiGetNodalDisplacementsEnvelope(
const cgiIStructure* pStructure,

```

```

    cgiSolverBlazeNamespace::cgiEnvelopeValue6** result,
    int* valueCount,
    int** envelopeLoadCombinationIndexes,
    int loadCombCount);
CGISOLVERBLAZE_API bool cgiGetSupportsEnvelope(
    const cgiIStructure* pStructure,
    cgiSolverBlazeNamespace::cgiEnvelopeValue6** result,
    int* valueCount,
    int** envelopeLoadCombinationIndexes,
    int loadCombCount);
CGISOLVERBLAZE_API bool cgiGetBeamVmdEnvelope(
    const cgiIStructure* pStructure,
    cgiSolverBlazeNamespace::cgiEnvelopeBmVMD** result,
    int* valueCount,
    int** envelopeLoadCombinationIndexes,
    int loadCombCount);
CGISOLVERBLAZE_API bool cgiGetShell4GroupNodalResultant(
    const cgiIStructure* pStructure,
    cgiSolverBlazeNamespace::cgiNodalResultant* pResult,
    int loadCombinationIndex,
    int** selectedNodes,
    int selectNodeCount,
    int** selectedShell4s,
    int selectedShell4Count,
    const cgiSolverBlazeNamespace::cgiPoint* resultLocation,
    int referenceShellId);
CGISOLVERBLAZE_API void cgiGetEigenResults(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiEigenValueVectorResult** result, int* count);
CGISOLVERBLAZE_API bool cgiGetEigenResultsForSingleMode(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiEigenValueVectorResult** result,
    int modeIndex);
CGISOLVERBLAZE_API void cgiGetResponseSpectrumResults(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiResponseSpectrumResult** result,
    int* count);
CGISOLVERBLAZE_API bool cgiGetResponseSpectrumResultsForSingleMode(const cgiIStructure* pStructure,
    cgiSolverBlazeNamespace::cgiResponseSpectrumResult** result, int modeIndex);
CGISOLVERBLAZE_API void cgiGetModalCombinationResults(const cgiIStructure* pStructure, cgiSolverBlazeNamespace::cgiCombinationResult* result);
CGISOLVERBLAZE_API void cgiGetModalCombinationBaseShears(const cgiIStructure* pStructure, double* fBaseShearX, double* fBaseShearY,
    double* fBaseShearZ);

//-----
// Saving / Loading
//-----
CGISOLVERBLAZE_API bool cgiSaveDocument(cgiIStructure* pStructure, LPCTSTR lpszPathName);
CGISOLVERBLAZE_API bool cgiOpenDocument(cgiIStructure* pStructure, LPCTSTR lpszPathName);

//-----
// Clear results
//-----
CGISOLVERBLAZE_API void cgiClearResult(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearStaticResult(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearFrequencyResult(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearResponseSpectrumResult(cgiIStructure* pStructure);

//-----

```

```

// Checking input, solution existence, running analyses
//-----
CGISOLVERBLAZE_API bool cgiCheckInputData(cgiIStructure* pStructure, int iMode, bool bReport /*= false*/);
CGISOLVERBLAZE_API bool cgiHasStaticSolution(cgiIStructure* pStructure);
CGISOLVERBLAZE_API bool cgiHasEigenSolution(cgiIStructure* pStructure);
CGISOLVERBLAZE_API bool cgiHasResponseSpectrumSolution(cgiIStructure* pStructure);
CGISOLVERBLAZE_API bool cgiRunStaticAnalysis(cgiIStructure* pStructure);
CGISOLVERBLAZE_API bool cgiRunFrequencyAnalysis(cgiIStructure* pStructure, bool bCalcMassOnly);
CGISOLVERBLAZE_API bool cgiRunResponseSpectrumAnalysis(cgiIStructure* pStructure);
CGISOLVERBLAZE_API bool cgiRunReport(cgiIStructure* pStructure);

//-----
// Contour results
//-----
CGISOLVERBLAZE_API void cgiGetContourMinMax(
    cgiIStructure* pStructure,
    double* fMin,
    double* fMax,
    int iContourIndex,
    int iComb,
    int iStressAtShellTopBottom);

//-----
// Error handling
//-----
CGISOLVERBLAZE_API int cgiGetLastError(const cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearLastError(cgiIStructure* pStructure);

//-----
// Show solver message box
//-----
CGISOLVERBLAZE_API void cgiSetShowSolverMessageBox(cgiIStructure* pStructure, bool bShow);
CGISOLVERBLAZE_API bool cgiGetShowSolverMessageBox(const cgiIStructure* pStructure);

//-----
// Edit functions
//-----
CGISOLVERBLAZE_API int cgiRemoveAllOrphanedNodes(
    cgiIStructure* pStructure,
    int** nodeIds, // was int*& in the interface, now pointer-to-pointer
    int* count);
CGISOLVERBLAZE_API bool cgiMergeAllNodesAndElements(
    cgiIStructure* pStructure,
    cgiSolverBlazeNamespace::cgiMergedInfo* mergedInfo);
CGISOLVERBLAZE_API bool cgiInsertNodesAtBeamIntersections(
    cgiIStructure* pStructure,
    cgiSolverBlazeNamespace::cgiIntersectionNode** intersectionNodes, // pointer-to-pointer
    int* intersectionNodeCount, // pointer
    int* beamIds,
    int beamCount);
CGISOLVERBLAZE_API int cgiExplodeBeamsAtNodes(
    cgiIStructure* pStructure,

```

```

int** affectedBeamIds, // pointer-to-pointer
int* affectedBeamCount, // pointer
int* beamIds,
int beamCount);

//-----
// Clear xxxx data
//-----
CGISOLVERBLAZE_API void cgiClearMaterials(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearSections(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearThickneses(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearSupports(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearNodalSprings(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearCoupledSprings(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearLineSprings(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearSurfaceSprings(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearMomentReleases(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearDiaphragms(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearMultiDofConstraints(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearNodes(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearBeams(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearShell4s(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearBricks(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearLoadCases(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearLoadCombinations(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearCaseLoads(cgiIStructure* pStructure);
CGISOLVERBLAZE_API void cgiClearNodalMasses(cgiIStructure* pStructure);

////////////////////////////////////
// C-API Wrapper for cgiLoadCombination
CGISOLVERBLAZE_API cgiSolverBlazeNamespace::cgiLoadCombination* cgiCreateLoadCombination(const TCHAR* szLabel);
CGISOLVERBLAZE_API void cgiDeleteLoadCombination(cgiSolverBlazeNamespace::cgiLoadCombination* pComb);
CGISOLVERBLAZE_API void cgiSetLoadComb(cgiSolverBlazeNamespace::cgiLoadCombination* pComb, const TCHAR* szLoadCombLabel, BOOL bConsiderPDelta,
                                      BOOL bEnableReport);
CGISOLVERBLAZE_API void cgiSetResponseSpectrumLoadFactor(cgiSolverBlazeNamespace::cgiLoadCombination* pComb, double factor);
CGISOLVERBLAZE_API double cgiGetResponseSpectrumLoadFactor(const cgiSolverBlazeNamespace::cgiLoadCombination* pComb);
CGISOLVERBLAZE_API void cgiAddLoadCombItem(cgiSolverBlazeNamespace::cgiLoadCombination* pComb, const TCHAR* szLoadCaseLabel, double fLoadFactor);
CGISOLVERBLAZE_API void cgiClearLoadCombItem(cgiSolverBlazeNamespace::cgiLoadCombination* pComb);
CGISOLVERBLAZE_API int cgiGetLoadCombItemCount(const cgiSolverBlazeNamespace::cgiLoadCombination* pComb);

// We copy the returned cgiLoadCombItem into *pOutItem so the C# side can read it.
CGISOLVERBLAZE_API bool cgiGetLoadCombItem(const cgiSolverBlazeNamespace::cgiLoadCombination* pComb, int index,
                                           cgiSolverBlazeNamespace::cgiLoadCombItem* pOutItem);
CGISOLVERBLAZE_API const TCHAR* cgiGetLoadCombinationLabel(const cgiSolverBlazeNamespace::cgiLoadCombination* pComb);
CGISOLVERBLAZE_API BOOL cgiGetLoadCombinationReport(const cgiSolverBlazeNamespace::cgiLoadCombination* pComb);
CGISOLVERBLAZE_API BOOL cgiGetLoadCombinationPDelta(const cgiSolverBlazeNamespace::cgiLoadCombination* pComb);

////////////////////////////////////
// load case load
CGISOLVERBLAZE_API cgiSolverBlazeNamespace::cgiLoadCaseLoad* cgiCreateLoadCaseLoad();
CGISOLVERBLAZE_API void cgiDeleteLoadCaseLoad(cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad);

```

```

CGISOLVERBLAZE_API void cgiAddNodalLoad(cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad, const cgiSolverBlazeNamespace::cgiNodalLoad* load);
CGISOLVERBLAZE_API void cgiAddPointLoad(cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad, const cgiSolverBlazeNamespace::cgiPointLoad* load);
CGISOLVERBLAZE_API void cgiAddLineLoad(cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad, const cgiSolverBlazeNamespace::cgiLineLoad* load);
CGISOLVERBLAZE_API void cgiAddMemberThermalLoad(cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad,
const cgiSolverBlazeNamespace::cgiThermalLoad* load);
CGISOLVERBLAZE_API void cgiAddShellThermalLoad(cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad,
const cgiSolverBlazeNamespace::cgiThermalLoad* load);
CGISOLVERBLAZE_API void cgiAddBrickThermalLoad(cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad,
const cgiSolverBlazeNamespace::cgiThermalLoad* load);
CGISOLVERBLAZE_API void cgiAddShellLoad(cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad, const cgiSolverBlazeNamespace::cgiShellLoad* load);
CGISOLVERBLAZE_API void cgiAddAreaLoad(cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad, const cgiSolverBlazeNamespace::cgiAreaLoad* load);
CGISOLVERBLAZE_API int cgiGetNodalLoadCount(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad);
CGISOLVERBLAZE_API int cgiGetPointLoadCount(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad);
CGISOLVERBLAZE_API int cgiGetLineLoadCount(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad);
CGISOLVERBLAZE_API int cgiGetMemberThermalLoadCount(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad);
CGISOLVERBLAZE_API int cgiGetShellThermalLoadCount(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad);
CGISOLVERBLAZE_API int cgiGetBrickThermalLoadCount(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad);
CGISOLVERBLAZE_API int cgiGetShellLoadCount(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad);
CGISOLVERBLAZE_API int cgiGetAreaLoadCount(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad);

// For each "getXXXLoad(index)" method, we copy the result into *pOut if valid.
CGISOLVERBLAZE_API bool cgiGetNodalLoad(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad, int index,
cgiSolverBlazeNamespace::cgiNodalLoad* pOut);
CGISOLVERBLAZE_API bool cgiGetPointLoad(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad, int index,
cgiSolverBlazeNamespace::cgiPointLoad* pOut);
CGISOLVERBLAZE_API bool cgiGetLineLoad(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad, int index,
cgiSolverBlazeNamespace::cgiLineLoad* pOut);
CGISOLVERBLAZE_API bool cgiGetMemberThermalLoad(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad, int index,
cgiSolverBlazeNamespace::cgiThermalLoad* pOut);
CGISOLVERBLAZE_API bool cgiGetShellThermalLoad(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad, int index,
cgiSolverBlazeNamespace::cgiThermalLoad* pOut);
CGISOLVERBLAZE_API bool cgiGetBrickThermalLoad(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad, int index,
cgiSolverBlazeNamespace::cgiThermalLoad* pOut);
CGISOLVERBLAZE_API bool cgiGetShellLoad(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad, int index,
cgiSolverBlazeNamespace::cgiShellLoad* pOut);
CGISOLVERBLAZE_API bool cgiGetAreaLoad(const cgiSolverBlazeNamespace::cgiLoadCaseLoad* pCaseLoad, int index,
cgiSolverBlazeNamespace::cgiAreaLoad* pOut);

////////////////////////////////////
//
CGISOLVERBLAZE_API cgiSolverBlazeNamespace::cgiBeamShearMomentDeflection* cgiCreateBeamShearMomentDeflection(int id);
CGISOLVERBLAZE_API void cgiDeleteBeamShearMomentDeflection(cgiSolverBlazeNamespace::cgiBeamShearMomentDeflection* pObj);
CGISOLVERBLAZE_API void cgiSetBeamShearMomentDeflectionId(cgiSolverBlazeNamespace::cgiBeamShearMomentDeflection* pObj, int id);
CGISOLVERBLAZE_API int cgiGetBeamShearMomentDeflectionId(const cgiSolverBlazeNamespace::cgiBeamShearMomentDeflection* pObj);
CGISOLVERBLAZE_API bool cgiGetBeamShearMomentDeflectionVMDValues(const cgiSolverBlazeNamespace::cgiBeamShearMomentDeflection* pObj,
cgiSolverBlazeNamespace::cgiVMD** pOutArray, int& outCount);

CGISOLVERBLAZE_API cgiSolverBlazeNamespace::cgiBeamStress* cgiCreateBeamStress(int id);
CGISOLVERBLAZE_API void cgiDeleteBeamStress(cgiSolverBlazeNamespace::cgiBeamStress* pObj);
CGISOLVERBLAZE_API void cgiSetBeamStressId(cgiSolverBlazeNamespace::cgiBeamStress* pObj, int id);
CGISOLVERBLAZE_API int cgiGetBeamStressId(const cgiSolverBlazeNamespace::cgiBeamStress* pObj);
CGISOLVERBLAZE_API bool cgiGetBeamStressValues(const cgiSolverBlazeNamespace::cgiBeamStress* pObj,

```

```

        cgiSolverBlazeNamespace::cgiSectionStress** pOutArray, int& outCount);

//////////
CGISOLVERBLAZE_API cgiSolverBlazeNamespace::cgiEnvelopeBmVMD* cgiCreateEnvelopeBmVMD(int id);
CGISOLVERBLAZE_API void cgiDeleteEnvelopeBmVMD(cgiSolverBlazeNamespace::cgiEnvelopeBmVMD* pObj);
CGISOLVERBLAZE_API void cgiSetEnvelopeBmVMDId(cgiSolverBlazeNamespace::cgiEnvelopeBmVMD* pObj, int id);
CGISOLVERBLAZE_API int cgiGetEnvelopeBmVMDId(const cgiSolverBlazeNamespace::cgiEnvelopeBmVMD* pObj);

// Retrieve the beam envelope VMD values.
// The function sets 'buffer' to point to an allocated array of cgiEnvelopeVMD
// and sets 'count' to the number of items.
CGISOLVERBLAZE_API void cgiGetEnvelopeBmVMDValues(const cgiSolverBlazeNamespace::cgiEnvelopeBmVMD* pObj,
        cgiSolverBlazeNamespace::cgiEnvelopeVMD*& buffer, int& count);

} // extern "C"

```

You can call the “C” interface functions above through P/Invoke mechanism. For convenience, SolverBlaze library provides a full set of P/Invoke C# interface function declarations and related C# structs in a file called `cgiStructureInterop.cs`. The library also provides a file called `cgiStructureInteropUtils.cs` that includes some convenient utility functions for such tasks as retrieving static analysis results, eigen analysis results, and response spectrum analysis through P/Invoke. The following `_display_static_results()` function is an example:

```

public static void _display_static_results(IntPtr pStructure, IntPtr combResultsPtr, int combCount)
{
    //IntPtr combResultsPtr = IntPtr.Zero;
    //int combCount = 0;
    //cgiStructureInterop.cgiGetStaticResults(pStructure, ref combResultsPtr, ref combCount);
    Console.WriteLine($"We have {combCount} combination results returned..");

    StringBuilder sTranslationDisplacementUnit = new StringBuilder(50);
    StringBuilder sRotationDisplacementUnit = new StringBuilder(50);
    StringBuilder sForceUnit = new StringBuilder(50);
    StringBuilder sMomentUnit = new StringBuilder(50);
    cgiStructureInterop.cgiGetUnit(pStructure, sTranslationDisplacementUnit, (int)cgiInteropUnitEnum.DISPLACEMENT_TRANS);
    cgiStructureInterop.cgiGetUnit(pStructure, sRotationDisplacementUnit, (int)cgiInteropUnitEnum.DISPLACEMENT_ROTATE);
    cgiStructureInterop.cgiGetUnit(pStructure, sForceUnit, (int)cgiInteropUnitEnum.FORCE);
    cgiStructureInterop.cgiGetUnit(pStructure, sMomentUnit, (int)cgiInteropUnitEnum.MOMENT);
    Console.WriteLine("Displacement Units: {0}, {1}", sTranslationDisplacementUnit, sRotationDisplacementUnit);
    Console.WriteLine("Forces and Moments Units: {0}, {1}", sForceUnit, sMomentUnit);

    if (combCount > 0 && combResultsPtr != IntPtr.Zero)
    {
        int combination_result_ptr_size = Marshal.SizeOf(typeof(cgiInteropCombinationResult));
        for (int iComb = 0; iComb < combCount; iComb++)
        {
            // for each load combination
            Console.WriteLine("***** Load combination {0} *****", iComb + 1);

            IntPtr combResultPtr = new IntPtr(combResultsPtr.ToInt64() + iComb * combination_result_ptr_size);
            cgiInteropCombinationResult result = Marshal.PtrToStructure<cgiInteropCombinationResult>(combResultPtr);

```

```

Console.WriteLine("Nodal displacements...");
int result6Val_ptr_size = Marshal.SizeOf(typeof(cgiInteropResult6Val));
for (int j = 0; j < result.m_nNdDispCount; j++)
{ // for each node
    IntPtr ptr = IntPtr.Add(result.m_vNdDisp, j * result6Val_ptr_size);
    cgiInteropResult6Val disp = Marshal.PtrToStructure<cgiInteropResult6Val>(ptr);
    Console.WriteLine("Node- {0}: \t{1:e3}\t{2:e3}\t{3:e3}\t{4:e3}\t{5:e3}\t{6:e3}", disp.iId,
        disp.fVal[(int)cgiInteropDofEnum.X], disp.fVal[(int)cgiInteropDofEnum.Y], disp.fVal[(int)cgiInteropDofEnum.Z],
        disp.fVal[(int)cgiInteropDofEnum.OX], disp.fVal[(int)cgiInteropDofEnum.OY], disp.fVal[(int)cgiInteropDofEnum.OZ]);
}
Console.WriteLine();

Console.WriteLine("Support reactions...");
for (int j = 0; j < result.m_nSuptReactCount; j++)
{ // for each support
    IntPtr ptr = IntPtr.Add(result.m_vSuptReact, j * result6Val_ptr_size);
    cgiInteropResult6Val supportReaction = Marshal.PtrToStructure<cgiInteropResult6Val>(ptr);
    Console.WriteLine("Support {0}: {1:e3}\t{2:e3}\t{3:e3}\t{4:e3}\t{5:e3}\t{6:e3}", supportReaction.iId,
        supportReaction.fVal[(int)cgiInteropDofEnum.X], supportReaction.fVal[(int)cgiInteropDofEnum.Y],
        supportReaction.fVal[(int)cgiInteropDofEnum.Z],
        supportReaction.fVal[(int)cgiInteropDofEnum.OX], supportReaction.fVal[(int)cgiInteropDofEnum.OY],
        supportReaction.fVal[(int)cgiInteropDofEnum.OZ]);
}
Console.WriteLine();

// and so on...
}

cgiStructureInterop.cgiDeleteMemoryArray_CombinationResult(pStructure, combResultsPtr);
}

```

Example Model Five (refer to the sample file verify-example4-interop.cs included in the library)

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Runtime.InteropServices;
using System.Text;

namespace cgiSolverBlazeTestInterop
{
    public class verify_example4_interop
    {
        public static void verify()
        {
            Console.WriteLine(Environment.CurrentDirectory);
            Console.WriteLine(AppDomain.CurrentDomain.BaseDirectory);

            IntPtr pStructure = cgiStructureInterop.cgiCreateStructure();
            if (pStructure == IntPtr.Zero)
            {
                Console.WriteLine("Error: cgiCreateStructure() failed!");
                return;
            }

            cgiStructureInteropUtils.setup_callbacks(pStructure);

            string testPath = cgiStructureInteropUtils.GetDefaultTestModelPath(pStructure, false);
            Console.WriteLine(testPath);
            if (!Directory.Exists(testPath))
            {
                Directory.CreateDirectory(testPath);
            }
            string sInputFileName = testPath + "\\Verify-Example4.r3a";
            Console.WriteLine(sInputFileName);

            cgiStructureInterop.cgiSetModelType(pStructure, (int)cgiInteropModelEnum.kModel_Frame2D);

            // 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
            cgiStructureInterop.cgiSetStandardEnglishUnits(pStructure);

            cgiInteropMaterial mat1 = new cgiInteropMaterial(1);
            mat1.setProperties("Default222", 29000, 0.3, 450);
            cgiInteropMaterial[] mats = new cgiInteropMaterial[] { mat1 };

            IntPtr matPtr = cgiStructureInteropUtils.ToUnmanagedArray(mats);
            cgiStructureInterop.cgiSetMaterials(pStructure, matPtr, mats.Length);
            cgiStructureInteropUtils.FreeUnmanagedArray(matPtr);
        }
    }
}
```

```

cgiInteropSection s1 = new cgiInteropSection(1);
s1.setProperties("W27X84", 24.8, 12.282, 12.7488, 2850, 106, 2.81);
cgiInteropSection s2 = new cgiInteropSection(2);
s2.setProperties("W27X84", 24.8, 12.282, 12.7488, 2850, 106, 2.81);
cgiInteropSection s3 = new cgiInteropSection(3);
s3.setProperties("W10X45", 13.3, 3.535, 9.9448, 248, 53.4, 1.51);

cgiInteropSection[] sects = new cgiInteropSection[] { s1, s2, s3 };
IntPtr sectPtr = cgiStructureInteropUtils.ToUnmanagedArray(sects);
cgiStructureInterop.cgiSetSections(pStructure, sectPtr, sects.Length);
cgiStructureInteropUtils.FreeUnmanagedArray(sectPtr);

List<cgiInteropNode> nodeList = new List<cgiInteropNode>();
cgiInteropNode nd1 = new cgiInteropNode(); nd1.setId(1); nd1.setCoordinates(0, 0, 0);
cgiInteropNode nd2 = new cgiInteropNode(); nd2.setId(2); nd2.setCoordinates(0, 12, 0);
cgiInteropNode nd3 = new cgiInteropNode(); nd3.setId(3); nd3.setCoordinates(0, 24, 0);
cgiInteropNode nd4 = new cgiInteropNode(); nd4.setId(5); nd4.setCoordinates(60, 0, 0);
cgiInteropNode nd5 = new cgiInteropNode(); nd5.setId(6); nd5.setCoordinates(60, 12, 0);
cgiInteropNode nd6 = new cgiInteropNode(); nd6.setId(4); nd6.setCoordinates(60, 24, 0);
nodeList.Add(nd1); nodeList.Add(nd2); nodeList.Add(nd3);
nodeList.Add(nd4); nodeList.Add(nd5); nodeList.Add(nd6);

IntPtr ndPtr = cgiStructureInteropUtils.ToUnmanagedArray(nodeList.ToArray());
cgiStructureInterop.cgiSetNodes(pStructure, ndPtr, nodeList.Count);
cgiStructureInteropUtils.FreeUnmanagedArray(ndPtr);

List<cgiInteropBeam> beamList = new List<cgiInteropBeam>();
cgiInteropBeam bm1 = new cgiInteropBeam(1); bm1.setId(1); bm1.setNodes(1, 2); bm1.setProperties(1, 3, 0);
cgiInteropBeam bm2 = new cgiInteropBeam(2); bm2.setId(2); bm2.setNodes(2, 3); bm2.setProperties(1, 3, 0);
cgiInteropBeam bm3 = new cgiInteropBeam(3); bm3.setId(3); bm3.setNodes(3, 4); bm3.setProperties(1, 2, 0);
cgiInteropBeam bm4 = new cgiInteropBeam(5); bm4.setId(5); bm4.setNodes(6, 4); bm4.setProperties(1, 3, 0);
cgiInteropBeam bm5 = new cgiInteropBeam(4); bm5.setId(4); bm5.setNodes(5, 6); bm5.setProperties(1, 3, 0);
beamList.Add(bm1); beamList.Add(bm2); beamList.Add(bm3); beamList.Add(bm4); beamList.Add(bm5);

IntPtr bmPtr = cgiStructureInteropUtils.ToUnmanagedArray(beamList.ToArray());
cgiStructureInterop.cgiSetBeams(pStructure, bmPtr, beamList.Count);
cgiStructureInteropUtils.FreeUnmanagedArray(bmPtr);

List<cgiInteropSupport> supList = new List<cgiInteropSupport>();
cgiInteropSupport sup1 = new cgiInteropSupport(); sup1.setId(1); sup1.setSupportDOFs("111000");
cgiInteropSupport sup2 = new cgiInteropSupport(); sup2.setId(5); sup2.setSupportDOFs("111000");
supList.Add(sup1); supList.Add(sup2);

IntPtr supPtr = cgiStructureInteropUtils.ToUnmanagedArray(supList.ToArray());
cgiStructureInterop.cgiSetSupports(pStructure, supPtr, supList.Count);
cgiStructureInteropUtils.FreeUnmanagedArray(supPtr);

cgiInteropLoadCase lc1 = new cgiInteropLoadCase();
lc1.setId(1);

```

```

lc1.setLoadCase("Default", "DEAD", true);
cgiInteropLoadCase[] loadCases = new cgiInteropLoadCase[] { lc1 };
IntPtr lcPtr = cgiStructureInteropUtils.ToUnmanagedArray(loadCases);
cgiStructureInterop.cgiSetLoadCases(pStructure, lcPtr, loadCases.Length);
cgiStructureInteropUtils.FreeUnmanagedArray(lcPtr);

// a little tricky with setting load combinations
IntPtr comb1 = cgiStructureInterop.cgiCreateLoadCombination("Linear");
cgiStructureInterop.cgiSetLoadComb(comb1, "Linear", false, true);
cgiStructureInterop.cgiAddLoadCombItem(comb1, "Default", 1.0);

IntPtr comb2 = cgiStructureInterop.cgiCreateLoadCombination("P-Delta");
cgiStructureInterop.cgiSetLoadComb(comb2, "P-Delta", true, true);
cgiStructureInterop.cgiAddLoadCombItem(comb2, "Default", 1.0);

IntPtr[] combos = new IntPtr[2];
combos[0] = comb1;
combos[1] = comb2;
IntPtr ptrCombArray = Marshal.AllocHGlobal(IntPtr.Size * combos.Length);
Marshal.Copy(combos, 0, ptrCombArray, combos.Length);
cgiStructureInterop.cgiSetLoadCombinations(pStructure, ptrCombArray, combos.Length);
Marshal.FreeHGlobal(ptrCombArray);

cgiStructureInterop.cgiDeleteLoadCombination(comb1);
cgiStructureInterop.cgiDeleteLoadCombination(comb2);

IntPtr caseload = cgiStructureInterop.cgiCreateLoadCaseLoad();

cgiInteropNodalLoad ndload1 = new cgiInteropNodalLoad();
ndload1.setLoad(4, -120.0, (int)cgiInteropDofEnum.Y);
cgiStructureInterop.cgiAddNodalLoad(caseload, ref ndload1);
cgiInteropNodalLoad ndload2 = new cgiInteropNodalLoad();
ndload2.setLoad(3, 6.0, (int)cgiInteropDofEnum.X);
cgiStructureInterop.cgiAddNodalLoad(caseload, ref ndload2);

cgiInteropPointLoad ptload = new cgiInteropPointLoad();
ptload.setLoad(3, (int)cgiInteropLoadSystemEnum.GLOBAL, -60.0, 0.333333, (int)cgiInteropDofEnum.Y);
cgiStructureInterop.cgiAddPointLoad(caseload, ref ptload);

IntPtr[] listCaseLoad = new IntPtr[1];
listCaseLoad[0] = caseload;
IntPtr ptrCaseLoadArray = Marshal.AllocHGlobal(IntPtr.Size * listCaseLoad.Length);
Marshal.Copy(listCaseLoad, 0, ptrCaseLoadArray, listCaseLoad.Length);
cgiStructureInterop.cgiSetCaseLoads(pStructure, ptrCaseLoadArray, listCaseLoad.Length);
Marshal.FreeHGlobal(ptrCaseLoadArray);

cgiStructureInterop.cgiDeleteLoadCaseLoad(caseload);

cgiInteropReportOptions rpt = new cgiInteropReportOptions();
rpt.selectAll();
rpt.bHTML = false;
cgiStructureInterop.cgiSetReportOptions(pStructure, ref rpt);

```

```

cgiInteropReportOptions rpt2 = new cgiInteropReportOptions();
cgiStructureInterop.cgiGetReportOptions(pStructure, ref rpt2);

// set analysis options
bool bConsiderBeamShearDeformation = false;
int nMaximumPDeltaIterations = 10;
double fPDeltaToleranceInPercentage = 0.5; // in percentage
int nNumberOfSegmentsForBeamOutput = 20;
bool bUseThinPlate = false;
bool bUseCompatibleModes = true;
int nUseAverageStressMode = 1;
cgiStructureInterop.cgiSetAnalysisOptions(pStructure, bConsiderBeamShearDeformation, nMaximumPDeltaIterations, fPDeltaToleranceInPercentage,
nNumberOfSegmentsForBeamOutput, bUseThinPlate, bUseCompatibleModes, nUseAverageStressMode);

cgiStructureInterop.cgiSetSolver(pStructure, (int)cgiInteropSolverEnum.kSolverSparse64);

bool saveOk = cgiStructureInterop.cgiSaveDocument(pStructure, sInputFileName);
Console.WriteLine("Saved Document? " + saveOk);

bool runOk = cgiStructureInterop.cgiRunStaticAnalysis(pStructure);
if (!runOk)
{
    Console.WriteLine("Error running static analysis... " + sInputFileName);
    goto Cleanup;
}

bool repOk = cgiStructureInterop.cgiRunReport(pStructure);
if (!repOk)
{
    Console.WriteLine("Error running report... " + sInputFileName);
    goto Cleanup;
}

// display results for all load combinations
cgiStructureInteropUtils.display_static_results(pStructure);

// display results for a single load combination
cgiStructureInteropUtils.display_static_results_for_single_load_combination(pStructure, 0);

Cleanup:
cgiStructureInterop.cgiDeleteStructure(pStructure);
}
}
}

```