SolverBlaze Finite Element Library (SDK)

Quick Guide

Computations & Graphics, Inc.
5290 Windflower Lane
Highlands Ranch, CO 80130, USA
Email: info@cg-inc.com
Web: www.cg-inc.com

Introduction

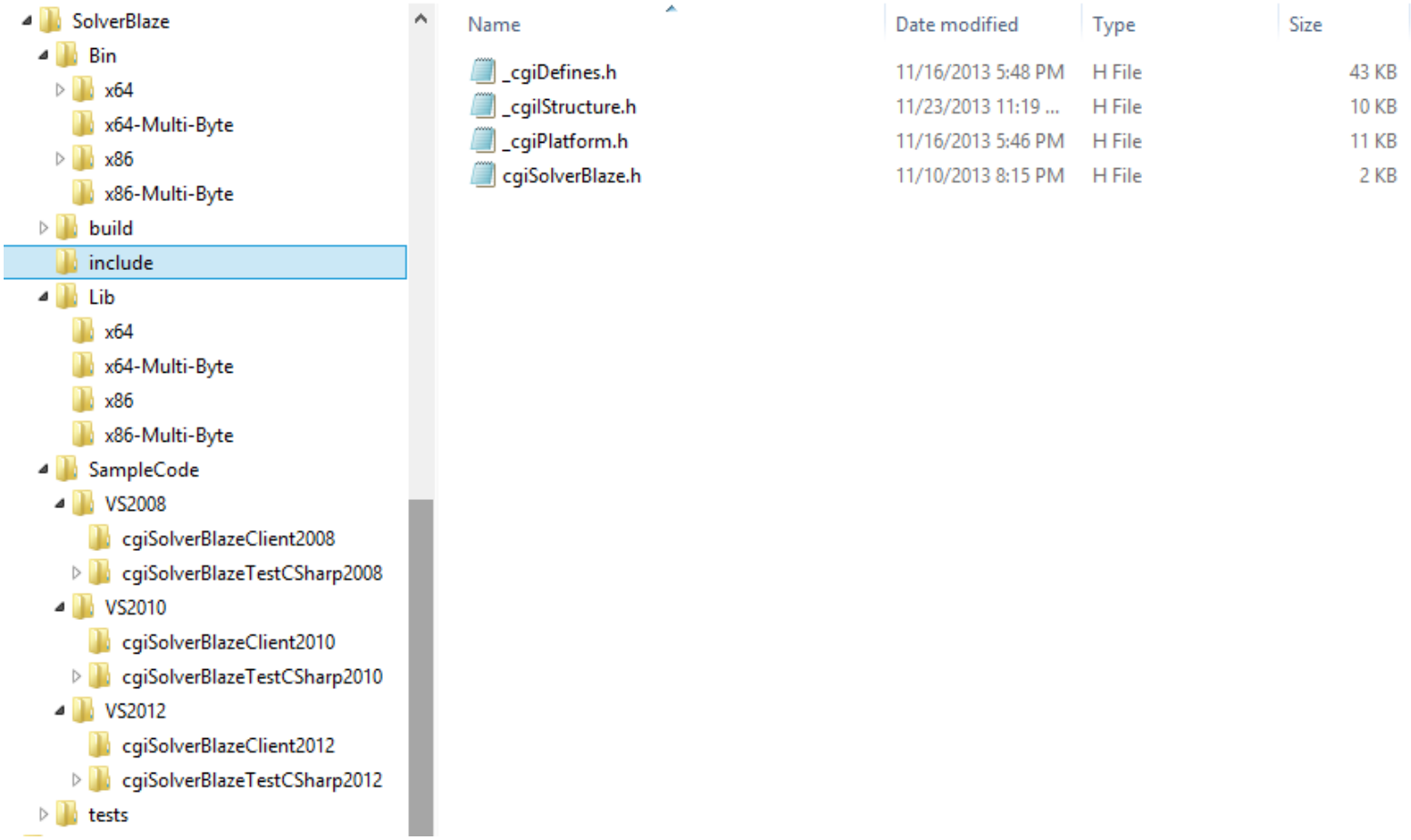
Computations & Graphics, Inc. (CGI) SolverBlaze Finite Element Library is a powerful structural and finite element analysis API (Application Programming Interface). It is based on the time-tested finite element solver engine in Real3D-Analysis, which has been sublicensed to over five hundred civil and 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) and brick elements.
- Supports nodal, point, line, surface and area loads.
- Supports conversion from area loads to line loads, local loads to global loads.
- Supports linear and nonlinear nodal, line and surface springs.
- Supports tension only/compression only members.
- Supports moment releases, rigid elements and rigid diaphragms.
- Support forced displacements.
- Supports both English and Metric units.
- Supports static linear and P-Delta analysis
- Supports frequency analysis.
- Supports response spectrum analysis.
- Supports inactive elements.
- Support both skyline and extremely fast sparse solver
- Support unique 128-bit floating point solver to handle numerically touch problems such as rigid diaphragms.
- Automatically generate input and result report in html format
- Bidirectional communication with Real3D-Analysis. For example, you can use Real3D-Analysis to open and graphically view model file generated by SolverBlaze. You can use Real3D-Analysis to automatically generate SolverBlaze source code that corresponds to the current model.
- Supports both native C++ language and managed .NET languages such as C# and VB.NET.
- Supports both 32-bit and 64-bit CPUs.
- Supports Unicode and Non-Unicode in Visual Studio 2008, 2010, 2012 and 2015.
- Available in both binary library and source code forms.
- A free copy of Professional Real3D-Analysis Program (source code customers only).
- Greate number of source code samples in C++ and C# are freely available.
- Technical support by SolverBlaze author.

- Reasonably priced.

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 2010, 2012 and 2015, a **Tests** folder which contains example files. The sample code will output to various folders under **Build** folder.



C++ Interface

The C++ library contains both a 32-bit and 64-bit 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. Intel PARDISO sparse solver (extremely fast, suitable for huge model with above one million DOFs) libguide40.dll and 128-bit floating point math library double128Proj.dll are also included.

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-Analysis, 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-Analysis.

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 getEngineerh(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
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
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 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 setTolerance(double fTolerance)=0;
virtual double getTolerance()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 setSolver(int iSolver)=0;
virtual int getSolver()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::cgiRelease* p)const=0;
virtual void deleteMemoryArray(cgiSolverBlazeNamespace::cgiDiaphragm* 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 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 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 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 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 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 void getEigenResults(cgiSolverBlazeNamespace::cgiEigenValueVectorResult*& result, int& count)const=0;

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

};
```

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};

// 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};
```

The following lists a few common constants

```
const int LABEL_SIZE = 128;

const int MAX_SPECTRUM_DATA_POINTS = 128;
```

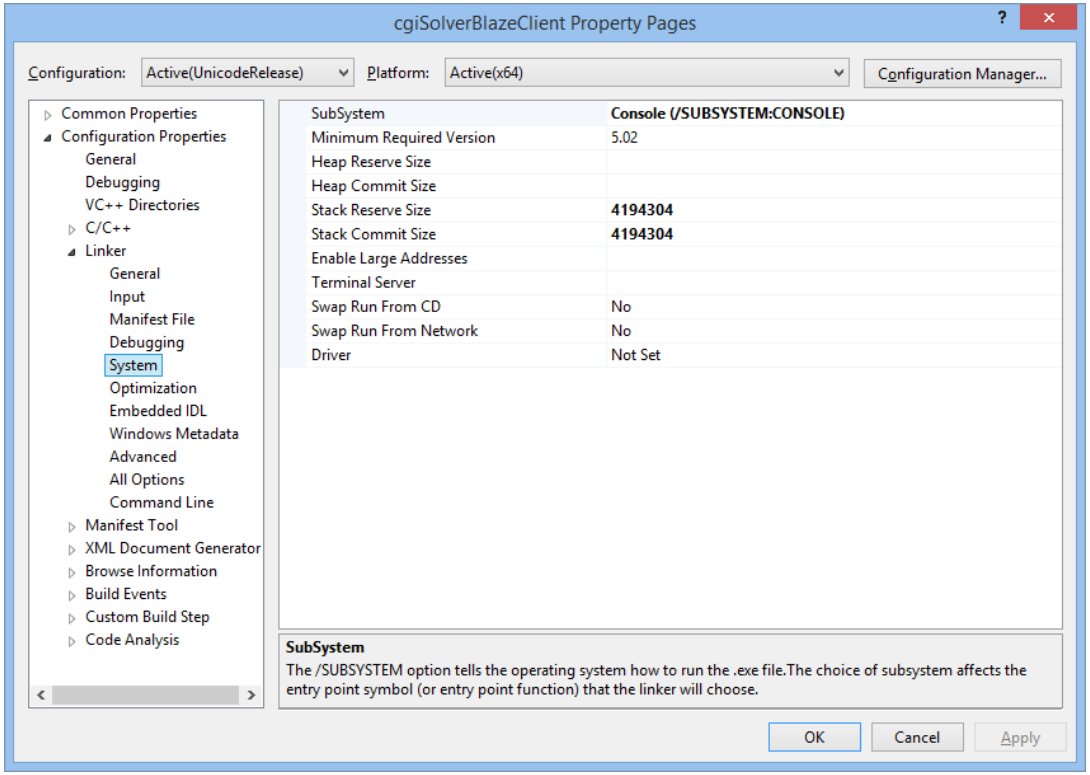
The following lists a few result data structures

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

struct CGISOLVERBLAZE_API 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 nIdentidy);
        int getId()const;
        void getBeamVMDValues(cgiVMD*& buffer, int& count)const;
};

// 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;
        cgiResult6Val* m_vSpringReact_Node;     // nodal spring reactions
        int m_nSpringReact_NodeCount;
        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
        int m_nFEMCount;
        cgiBeamEndVMD* m_vBmEndVMD;             // vmd at member ends
        int m_nBmEndVMDCount;
        cgiBeamShearMomentDeflection* m_vBmVMD; // vmd 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];
};
```

Four versions of cgiSolverBlaze.lib and cgiSolverBlaze.dll are provided for your configuration needs: Unicode or Multi-Bytes, 86x or 64x CPUs. You should link your projects to the correct version of the lib file. Make sure you also copy the correct version of the DLLs (libguide40.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.

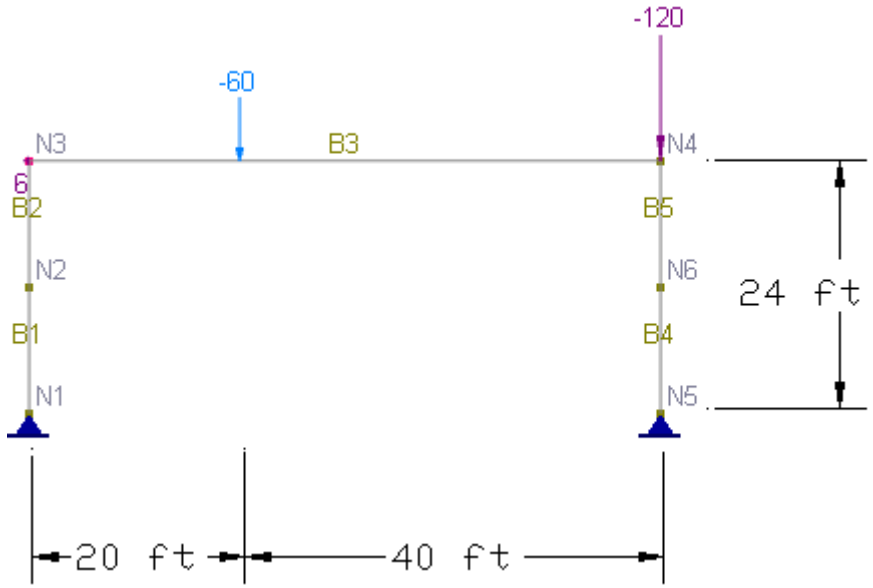


We will illustrate the use of the SolverBlaze API by using the following structural models taken from example 4 and B-01 (Plate Patch Test) of Real3D-Analysis Verification Manual, which is freely available for download from http://www.cg-inc.com/download/REAL3DVerifications.pdf

It is highly recommended that you study the "Technical Issues" in the Real3D-Analysis 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 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 in^2, I_z = 248 in^4
Beam: W27x84, A = 24.8 in^2, I_z = 2850 in^4
Material: E = 2.9e7 psi, ν = 0.3
Perform analysis for the following two cases:
a). First order (Linear) elastic analysis
b). 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
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;
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());

// set report options
cgiReportOptions 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};
```

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
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
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 cgiIStructure::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 cgiIStructure::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.

```
   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::setNonliear(). 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 cgiIStructure:: `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 cgiIStructure:: `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 cgiIStructure:: 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());
```

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()

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 cgiIStructure:: saveDocument(). This file can then be opened, visualized or analyzed by Real3D-Analysis. This can be very useful to verify the correctness of the model input and its results. The actual structural analysis is performed by calling cgiIStructure:: 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.

```
  // 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);
```

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

		Real3D-Analysis	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

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

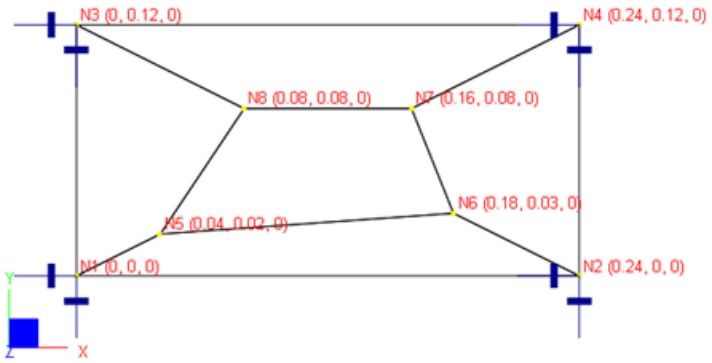
A plate of size 0.12 x 0.24 in is subjected to forced displacements at the four corners as shown below. The boundary conditions are:

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

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

Material properties: E = 1.0e6 psi, ν = 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 Dox	Rotation Doy
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.

.NET Interface

The .NET interface is a Class Library DLL (cgiSolverBlazeCli.dll). It is written in C++/CLI and makes calls to the native Windows DLL (cgiSolverBlaze.dll). The actual procedure to use this .NET interface is identical to C++ interface, except syntax differences.

`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-Analysis, 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-Analysis.

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

```
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 modelNamep);
    public void setDesignCompany(string designCompany);
    public void getDesignCompany(ref StringBuilder designCompany);
    public void setEngineer(string engineer);
    public void getEngineerh(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 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 setTolerance(double fTolerance);
public double getTolerance();
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 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 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 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 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 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 void getEigenResults(ref List<cgiEigenResultCli> listEigenResult);
    public void getResponseSpectrumResults(ref List<cgiResponseSpectrumResultCli> listResult);
    public void getModalCombinationResults(ref cgiCombResultCli modalResult);
    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 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);
  }
}
```

The following lists a 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,
}

public enum cgiSolverEnum
{
        kSolver64,
        kSolver128,
        kSolverSparse64,
}

public enum cgiIncludeEnum
{
        kInclude,
        kExclude,
}

public enum cgiErrorEnum
{
        kErrorNone,
        kInvalidInput,
        kErrorProcessingElementStiffness,
        kNoMassDefined,
        kErrorConvertingAreaLoadsToLineLoads,
        kTooManyDigitsLostDuringFactorization,
        kSolverError,
        kAbnormalSolverTermination,
        kMustRunResponseSpetrumAnalysis,
        kMustRunFrequencyAnalysisFirst,
}

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 class cgiVMDCli
{
        public double fDist;            // ratio
        public List<double> fVMD;       // forces and moments in six DOF directions
        public List<double> fDefl;      // 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
}

// results for a load combination
public class cgiCombResultCli
{
        public List<cgiResult6ValCli> m_listNdDisp;             // nodal displacements
        public List<cgiResult6ValCli> m_listSuptReact;          // support reactions
        public List<cgiResult6ValCli> m_listSpringReact_Node;   // nodal 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
}

// 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
}
```

Different versions of cgiSolverBlazeCli.dll are provided to work with .NET Framework 3.5, 4.0, 4.5 and 4.6 and x86 and x64 CPUs. Make sure your project has reference to the correct cgiSolverBlazeCli.dll. If your project is configured to build for both x86 and x64 CPUs, make sure to set "Copy Local" to false for the reference and manually copy the correct version of cgiSolverBlazeCli.dll to the executable directory. Also make sure a copy of libguide40.dll, double128Proj.dll, cgiSolverBlaze.dll and cgiSolverBlazeCli.dll is placed in the executable directory.

For distribution, you may need to install Visual C++ x86 or x64 redistributables corresponding to Visual Studio 2015 as cgiSolverBlaze.dll and cgiSolverBlazeCli.dll links dynamically with runtime library.

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-Analysis 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-Analysis 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.

For more detailed explanation, please refer to the detailed explanation sections of Example Model One and Two for C++ Interface. Other than the syntax differences between C++ and C# (VB.NET), the procedures involved are exactly the same. The source code for these examples are included in the library (verify-example4.cs and verify-plate-patch-test.cs).

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

```
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
            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())
{// 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 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("");
            }

        }
    }
}
```

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

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

```
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 One in the C++ Interface section as the procedures involved are similar.