

**Documentation of the BEMCmpMgr (CEC Compliance Manager) DLL
For Analysis of Commercial Buildings
SAC 3/24/2014 (v7) (latest changes in Red font)**

The purpose of this document is to provide information needed to develop software interfaces to the CEC Compliance engine DLL(s). The primary library third party tools will interface with is BEMCmpMgr_os.dll. This DLL manages the compliance analysis processing, including:

- Evaluation of compliance rules on user input building models (via BEMProc.dll),
- Simulation of the proposed and standard building models (via OpenStudio/EnergyPlus (and soon to be T24DHW.dll for recirculating hot water systems)), and
- Generation of compliance reports (informal reports internal to compliance manager, certified reports via web-based report generator).

Ultimately, the analysis engine DLLs will have only open source dependencies (primarily boost, Qt C++, OpenSSL, cURL and OpenStudio libraries), but for the time being there are still dependencies on Microsoft (Windows) libraries. We anticipate some, but not substantial, changes to the software interfacing routines documented here as we migrate the compliance manager toward open source dependencies in the coming months.

There are only a handful of exports needed to perform compliance analysis and retrieve analysis results/reports. One initializes the DLLs, one performs the analysis (and potentially generates a report), another to generate a draft report from an existing analysis results file, three more are designed to retrieve data (inputs and/or results) from the building model following analysis, a couple more to retrieve error messages and the last is used to clean-up following exit.

The overall analysis data model is broken into a few subsets, one of which is the “Input Data Model”. This is the version of most interest to third party tools wishing to generate files that can be analyzed for compliance using this tool. The Input Data Model is documented in a text file contained in each release of CBECC-Com, residing in the “Data” directory of the install, at:

<CBECC-Com xxx Data>\Documents\RulesetSource\CEC 2013 NonRes - Input Data Model.txt

There is one variation associated with the input data model that is important to understand, and that has to do with the ability to describe buildings using either “Detailed” or “Simplified” geometry. There are certain limitations on building and analysis features when using the simplified approach, while any/all features and analysis can be performed on detailed geometry projects. For more information on this topic, refer to the CBECC-Com User’s Manual.

A new feature of the Input Data Model text file is the ability to specify whether or not certain building object properties should or should not be included in detailed vs. simplified building model files submitted for analysis. This is denoted in the highlighted excerpt from the Input Data Model text:

```
Spc                Space                #Props:109/340  MaxDefinable: 1000
Parent(s):  Story
Children:  IntLtgSys / DayltgCtrl / Ceiling / ExtFlr...
...
Area  ...  Float  ...  Units: ft2  ...  (see next line)  Error if not: Value >= 0.01
...  ...  ...  ...  Compulsory when Proj:GeometryInpType = 'Simplified', else NotInput
```

One data model variation that is NOT documented in the Input Data Model text file (described above) is the inclusion of PolyLp (PolyLoop) objects in project files. Detailed geometry projects should include PolyLp children for Spc (Space) objects and all types of surfaces & shading devices (Ceiling, ExtFlr, ExtWall, IntFlr, IntWall, Roof,

UndgrFlr, UndgrWall, Win, Skylt, Dr & ExtShdgObj). Simplified geometry projects should include NO PolyLp objects. If PolyLp objects are included in simplified geometry projects, they will be removed and not considered during analysis.

Recommended sequence of events (assuming use of Visual Studio & C++) for a scenario where direct linking to the compliance manager DLLs is not required:

1. Load the pertinent DLLs via LoadLibrary() -->> "libeay32.dll", "ssleay32.dll", "~~QtCore4.dll", "QtXml4.dll", "QtGui4.dll", "BEMProc.dll",~~ and "BEMCmpMgr_os.dll"
This step MAY not be required if the third party application and these referenced DLLs are located in the same directory.
2. Call GetProcAddress() to retrieve function pointers to each DLL function you plan to call.
3. Call InitBEMProcAndCmpMgrDLLs () to initialize the BEMProc & BEMCmpMgr_os DLLs with the desired building data model.
4. For each building model to be simulated:
 - Call CMX_PerformAnalysis_CECNonRes () to perform the compliance analysis.
 - Call CMX_GenerateReport_CEC () to generate a PDF compliance report based on the results contained in an XML file identified by the calling application.
 - Call CMX_GetDataString (), CMX_GetDataInteger (), &/or CMX_GetDataFloat () any number of times to retrieve building model inputs, calculated defaults or simulation results (anything stored in the building model).
 - Call CMX_GetRulesetErrorCount () to retrieve the number of errors encountered during the analysis and CMX_GetRulesetErrorMessage () to retrieve individual error message character strings.
 - ~~- Call CMX_PopulateCSVHourlyResults_CECRes () to export a CSV file containing hourly simulation results and TDV multipliers for a single analysis run (proposed vs. standard...). (only available for residential analysis for the time being)~~
5. Call ExitBEMProcAndCmpMgrDLLs () following all simulations to clean-up BEMProc & BEMCmpMgr_os data.
6. Unload the loaded DLLs (in reverse order) via FreeLibrary() -->> "BEMCmpMgr_os.dll", "~~BEMProc.dll", "QtGui4.dll", "QtXml4.dll", "QtCore4.dll",~~ "ssleay32.dll", and "libeay32.dll"
This step not required if the third party DLLs are not explicitly loaded in step 1.

Function Reference

The following are names and documentation pertaining to the compliance manager routines referenced above

```
void InitBEMProcAndCmpMgrDLLs( const char* psBEMProcFileName,
                              int iBEMType,
                              const char* psInitLogFileName );

// typedef void ( __cdecl *PInitBEMProcAndCmpMgrDLLs)( const char*, int,
//                                                    const char* );
// _InitBEMProcAndCmpMgrDLLs
```

(note: No longer using mangled names as of version 1c (500) / v2 of this document)

where:

- `psBEMProcFileName` is a null terminated string containing the path and filename of the data model definitions file. In the CBECC-Com installer, this would be: '`<Data directory>\Rulesets\CEC 2013 Nonres\CEC 2013 NonRes BEMBase.bin`'.
- `iBEMType` is an enumeration describing the data model type. For CBECC-Com (or -Res) processing this should be set to 0.
- `psInitLogFileName` is a null terminated string containing the path and filename of a file used to log messages. This is something that gets re-set with each project file read, so we tend to ignore this – passing in NULL for this argument.

Call this routine once following the loading of the BEMProc & BEMCmpMgr_os DLLs.

note – all path/filename arguments can be either complete or relative to the path in which the calling executable resides.

```
int CMX_PerformAnalysis_CECNonRes (
    const char* pszRulesetPathFile,
    const char* pszDHWDDLPath,
    const char* pszProcessingPath,
    const char* pszLogPathFile,
    bool bLoadModelFile,
    char* pszErrorMessage,
    bool bDisplayProgress,
    const char* pszBEMBasePathFile,
    const char* pszSimWeatherPath,
    const char* pszDHWeatherPath,
    const char* pszModelPathFile,
    const char* pszUIVersionString,
    const char* pszAnalysisOptionsCSV,
    int iErrorMsgLength,
    HWND hWnd );

// typedef int ( __cdecl *PCMX_PerformAnalysis_CECNonRes)( const char*,
//                                                         const char*, const char*, const char*,
//                                                         const char*, const char*, const char*,
//                                                         const char*, const char*, bool,
//                                                         const char*, char*, int, bool, HWND );
// _CMX_PerformAnalysis_CECNonRes
```

where:

- `pszBEMBasePathFile` is a null terminated string containing the path and filename of the data model definitions file. This should not be specified (pass in NULL) unless the run being performed uses a different data model as the one previously used to initialize the DLLs or perform a simulation.
- `pszRulesetPathFile` is a null terminated string containing the path and filename of the ruleset file used to process the building model. This should be specified for each run when switching building models, as it re-initializes both the building model and ruleset data. In the CBECC-Com installer, this would be: '`<Data directory>\Rulesets\CEC 2013 NonRes.bin`'.
If a simulation is to be performed on a building model already loaded into memory, then you can specify NULL for this argument.
- `pszSimWeatherPath` is a null terminated string containing path (including trailing '\') of the directory where the EnergyPlus CEC weather files reside. The default CBECC-Com installer location for these files is: '`<Data directory >\EPW\`'.

- **pszCompMgrDLLPath** is a null terminated string containing path (including trailing '\') of the directory where the CEC compliance manager DLLs reside. If NULL or a zero-length path is specified for this argument, then the path to the executable file which loaded & called this analysis routine will be used. The analysis mechanism assumes that other required executables are located as follows in relation to the directory identified by this argument:
 - EnergyPlus simulation executable & DLLs located in 'EPlus\' subdirectory, and
 - CEC DHW engine DLLs located in 'T24DHW\' subdirectory.
- **pszDHWWeatherPath** is a null terminated string containing path (including trailing '\') of the directory where the CEC DHW simulation weather files reside. In the CBECC-Res installer, this would be: '<Program directory>\CSE\'.
This argument is NOT YET IMPLEMENTED and should always be NULL (for the time being). The DHW simulation engine IS currently compatible with TMY(2) files, but the BEMCmpMgr_os interface to the DHW engine is currently available only for the California Climate Zones.
- **pszProcessingPath** is a null terminated string containing the path (including trailing '\') of the directory where the simulation inputs and outputs will be written.
- **pszModelPathFile** is a null terminated string containing the path and filename of the building model input (.ibid or .xml) file. The model input file identified by this argument may or may not be loaded into memory during the analysis processing depending on a subsequent argument.
- **pszLogPathFile** is a null terminated string containing the path and filename of a file to write processing messages to. If specified as NULL (which it typically is), then the log file will be equivalent to the pszModelPathFile specified in the previous argument, but with the file extension replaced with '.log'.
- **pszUIVersionString** is a null terminated string containing the name and version/ID of the application that is calling this function. This information is stored in the data model for later reporting in results export and final compliance reports.
- **bLoadModelFile** is a boolean indicating whether or not to read the building model (pszModelPathFile) into memory prior to analysis. This should be specified as 'true' (non-zero) unless the calling application has already loaded the building model into memory via a previous call.
- **pszAnalysisOptionsCSV** is a null terminated, comma separated value (CSV) formatted string defining any/all of the following analysis options:
 - **StoreBEMDetails:** boolean (0/1 – default 0): whether or not to store detailed building energy model ('.ibid-detail') files during the course of the model defaulting and analysis. Typically set to 1 in testing/debugging phases and turned off (value of 0, the default) for distribution to users (reducing processing time and file I/O).
 - **Verbose:** boolean (0/1 – default 0): whether or not to write messages to the log file for each rule evaluated on the model and identifying each step in the analysis sequence. Typically set to 1 in certain testing/debugging phases and turned off (value of 0, the default) for distribution to users (reducing processing time and file I/O – can result in very large analysis log files).
 - **Silent:** boolean (0/1 – default 0): whether or not dialog boxes are permitted to be presented during the analysis. One example is a dialog indicating that a file needing to be written during/following the analysis cannot be written to, prompting the user to close the file in another application it is opened in so that the file can be re-written. A value of '1' will prevent user prompts and issues such as files unable to be written may cause the analysis to be aborted.
 - **AnalysisThruStep:** integer (0-100 – default 100): an indication of the last analysis step to be executed before aborting the analysis. Steps of the analysis (and their corresponding integer values) include:

1. Analysis initialization
 2. Generate Proposed building model
 3. Generate Proposed model OSM (OpenStudio Model) and IDF (EnergyPlus Input) files
 4. Simulate Proposed model, retrieve results and perform UMLH (unmet load hours) check
 5. Generate BaselineSizing model
 6. Generate BaselineSizing model OSM and IDF files
 7. Simulate BaselineSizing model, retrieve HVAC capacities/flows (possible iteration)
 8. Generate Baseline model
 9. Generate Baseline model OSM and IDF files
 10. Simulate Baseline model, retrieve results and perform UMLH check
 11. Generation of compliance report
- DontAbortOnErrorsThruStep: integer (0-100 – default 0): an indication of how far the analysis sequence (based on the steps listed above) should be executed, regardless of the number or types of errors encountered through that point.
 - BypassInputChecks: boolean (0/1 – default 0): whether or not numeric range and required input checks should be bypassed during the course of performing the analysis. This is typically activated (set to 1) only when performing software testing/debugging and a certified compliance report cannot be generated when set to 1.
 - BypassUMLHChecks: boolean (0/1 – default 0): whether or not UMLH (unmet load hour) checks should be bypassed during the course of performing the analysis. This is typically activated (set to 1) only when performing software testing/debugging and a certified compliance report cannot be generated when set to 1.
 - BypassCheckSimRules: boolean (0/1 – default 0): whether or not CheckSim rules (designed to ensure that the building model as input can be simulated) should be bypassed during the course of performing the analysis. This is typically activated (set to 1) only when performing software testing/debugging and a certified compliance report cannot be generated when set to 1.
 - BypassCheckCodeRules: boolean (0/1 – default 0): whether or not CheckCode rules (designed to ensure that mandatory code requirements are present in the building model) should be bypassed during the course of performing the analysis. This is typically activated (set to 1) only when performing software testing/debugging and a certified compliance report cannot be generated when set to 1.
 - BypassOpenStudio_p / BypassOpenStudio_bz / BypassOpenStudio_b: boolean (0/1 – default 0): whether or not OpenStudio (and the EnergyPlus simulation) is to be bypassed for the proposed ('_p'), baseline sizing ('_bz') and/or baseline ('_b') models. This is typically activated (set to 1) only when performing software testing/debugging and a certified compliance report cannot be generated when set to 1.
 - BypassSimulation_p / BypassSimulation_bz / BypassSimulation_b: boolean (0/1 – default 0): whether or not the EnergyPlus simulation is to be bypassed for the proposed ('_p'), baseline sizing ('_bz') and/or baseline ('_b') models. This is typically activated (set to 1) only when performing software testing/debugging and a certified compliance report cannot be generated when set to 1.
 - OverrideAutosize_p / OverrideAutosize_bz / OverrideAutosize_b: integer (-1/0/1 – default -1): whether to force the simulation to automatically size (autosize) the HVAC equipment ('1'), to not autosize the equipment ('0') or to simulate using the default autosizing setting ('-1' => off for Proposed & Baseline, on for BaselineSizing). This is typically activated (set to 0 or 1) only when

performing software testing/debugging and a certified compliance report cannot be generated when set to a value other than -1.

- IgnoreFileReadErrors: boolean (0/1 – default 0): whether or not the analysis should ignore (continue analysis following) errors encountered when reading and parsing the input building model.
- PurgeUnreferencedObjects: boolean (0/1 – default 1): whether or not purge (delete) building objects from the model that have no effect on the analysis during the early stages of rule processing.
- ComplianceReportPDF: boolean (0/1 – default 0): whether or not a PDF compliance report will be generated at the conclusion of the analysis, assuming no errors have occurred and no checks or simulation runs were bypassed. In the event a compliance PDF is generated, the file will be located in the project directory and named:
 <project file name> - AnalysisResults-BEES.pdf
This option can also be toggled on by setting Proj:CompReportPDF to '1' in the input building model.
- ComplianceReportXML: boolean (0/1 – default 0): whether or not a full (XML) compliance report will be generated at the conclusion of the analysis, assuming no errors have occurred and no checks or simulation runs were bypassed. Full XML compliance reports include all analysis inputs and results as well as an imbedded PDF report. In the event a XML compliance report is generated, the file will be located in the project directory and named:
 <project file name> - AnalysisResults-BEES.xml
This option can also be toggled on by setting Proj:CompReportXML to '1' in the input building model.
- SimulationStorage: integer (0-7 – default 1): determines which, if any, simulation input and/or output files are to be retained following each building model simulation. Valid options include:
 - 0. ALL simulation sub-directories and files deleted
 - 1. Only the input (.idf) files are retained
 - 2. (#1 above) + summary output (.htm) retained
 - 3. (#2 above) + limited additional output (.csv|.eio|.err|.rdd) retained
 - 4. (#3 above) + SQL output (.sql) retained
 - 5. (#3 (not 4) above) + both SQL & other standard output files retained
 - 6. (#4 & 5 above) +
 - 7. ALL simulation input and output files retained
- AnalysisStorage: integer (0-3 – default 2): determines which, if any, (non-simulation) files generated during analysis are to be retained following each round of analysis. Valid options include:
 - 0. ALL analysis files deleted
 - 1. Only simulation SDD XML (.xml) files are retained
 - 2. (#1 above) + OpenStudio model (.osm) files are retained
 - 3. ALL files produced during analysis are retained
- ProxyServerAddress: character string: The address (i.e. "site.site:port") of the proxy server to be used in accessing the report generator (via the internet).
- ProxyServerCredentials: character string: Username and password credentials (i.e. "username:password") needed to access the internet via the proxy server (only needed for report generation).

- ModelRpt_ALL (or specific model report options): boolean (0/1 – default 0): whether or not certain CSV report files are to be written by the ruleset for each model generated during the analysis. Such CSV files are used primarily for testing/debugging, but can be informative in comparing certain aspects of the analysis models. The ModelRpt_ALL can be specified to activate all defined reports, or individual report options can be specified as:
 ModelRpt_Space_InteriorLoadsElec (self explanatory)
 ModelRpt_Space_InteriorLoadsFuel

Multiple analysis options can be concatenated into the pszAnalysisOptionsCSV string, for instance, if you wanted to perform a run with the Verbose & BypassOpenStudio_p flags activated (and all other defaults used), then this function argument would include the string “Verbose,1,BypassOpenStudio_p,1,”.

Searches for options in this string argument are case insensitive and if a single flag is repeated multiple times, the value associated with its first occurrence will drive the analysis.

- pszErrorMessage is a pointer to a character string that can be populated by this function to describe error(s) encountered during the analysis. Pass in a value of 0/NULL if no return of error messaging is desired.
- iErrorMsgLength is an integer describing the number of characters present in the error message character string (previous argument). Pass in a value of 0 if no return of error messaging is desired.
- bDisplayProgress is a boolean indicating whether or not to display the analysis progress dialog as the processing is executed.
- hWnd is a HANDLE to a MS Windows application window. This argument is not currently used.

Return value is 0 if simulation successful and > 0 if errors occurred (listed below).

Call this routine once for each compliance analysis to be performed.

All path/filename arguments can be either complete or relative to the path in which the calling executable resides.

Error return value mapping:

- 1 : pszBEMBasePathFile doesn't exist
- 2 : pszRulesetPathFile doesn't exist
- 3 : pszSimWeatherPath doesn't exist
- 4 : pszCompMgrDLLPath specified, but doesn't exist
- 5 : Invalid project log file name (too long)
- 6 : Error writing to project log file
- 7 : Building model input/project file not found
- 8 : Error reading/initializing model input/project file
- 9 : Errors encountered evaluating input model defaulting rules
- 10 : Errors encountered evaluating input model defaulting rules (multiple times)
- 11 : Error(s) encountered performing required data & numeric range checks
- 12 : Error(s) encountered checking input model for simulation compatibility
- 13 : Error(s) encountered checking input model for code requirements
- 14 : Error encountered initializing weather file locations and/or names
- 15 : Error creating or accessing the analysis processing directory
- 16 : Error generating Proposed Sizing model

- 17 : Error generating Proposed (final) model
- 18 : Error generating Standard Sizing model
- 19 : Error generating Standard (final) model
- 20 : Error initializing Standard Sizing model
- 21 : Error initializing Standard (final) model
- 22 : Analysis aborted - user chose not to overwrite SDD XML file
- 23 : Error: Unable to write SDD XML file
- 24 : Error(s) encountered simulating Proposed model
- 25 : Error(s) encountered simulating Standard Sizing model
- 26 : Error(s) encountered simulating Standard (final) model
- 27 : Error(s) encountered retrieving Proposed model simulation results
- 28 : Error(s) encountered retrieving Standard Sizing model simulation results
- 29 : Error(s) encountered retrieving Standard (final) model simulation results
- 30 : Proposed model zone(s) exceed unmet load hours limits
- 31 : Error initializing building model database
- 32 : Error loading analysis ruleset
- 33 : User aborted analysis via progress dialog 'Cancel' button
- 34 : Invalid results object types
- 35 : Error copying results objects from a previous model
- 36 : Error copying equipment sizes/flows from source model
- 37 : Error(s) encountered reading building model (input/project) file
- 38 : Error: EnergyPlus simulation engine not found.
- 39 : Error: Version of EnergyPlus installed not compatible with analysis.
- 40 : Error setting up check of weather & design day file hashes
- 41 : DHW simulation not successful

(return values in the range 101-200 describe issues encountered during/by simulation)

- 101 : SDD XML simulation input file not found
- 102 : Simulation weather file not found
- 103 : Simulation processing path not valid
- 104 : Simulation executable path not valid
- 105 : Simulation error output path/file not valid
- 106 : User aborted analysis
- 131 : Error encountered in OpenStudio loading SDD XML file
- 132 : Error encountered in OpenStudio saving model to OSM file
- 133 : Unable to locate EnergyPlus simulation SQL output file
- 134 : OpenStudio Model not valid following simulation
- 135 : OpenStudio Facility not valid following simulation
- 136 : Error creating OpenStudio Model object
- 161 : Fatal error(s) occurred in EnergyPlus simulation
- 162 : EnergyPlus simulation did not complete successfully
- 181 : User aborted analysis during building model simulation

```
void CMX_GenerateReport_Proxy_CEC( const char* pszXMLResultsPathFile,
                                   const char* pszCACertPath,      const char* pszReportName,
                                   const char* pszAuthToken1,       const char* pszAuthToken2,
                                   const char* pszSignature,         const char* pszPublicKey,
```

```

const char* pszProxyAddress, const char* pszProxyCredentials,
const char* pszDebugBool, bool bVerbose, bool bSilent,
bool bUnused );

// typedef int (__cdecl *PCMX_GenerateReport_Proxy_CEC( const char*,
// const char*, const char*, const char*, const char*,
// const char*, const char*, const char*, const char*,
// const char*, bool, bool, bool );
// _CMX_GenerateReport_Proxy_CEC

```

where:

- `pszXMLResultsPathFile` is a null terminated string containing the path and filename of the analysis results XML file. This is typically: <Project file directory>\<project file name> - AnalysisResults.xml
- `pszCACertPath` is a null terminated string containing the path (only) of the CA cert bundle (curl-ca-bundle.crt) used to verify server certificates. For CBECC-Res, this file is distributed in the main program directory.
- `pszReportName` is a null terminated string containing the name of the report to be generated. Valid report names currently include:
 - “NRCC_PRF_01” (Certificate of Compliance - Nonresidential Performance Compliance Method)
- `pszAuthToken1` is a null terminated string that, combined with `AuthToken2`, identifies the calling application. For the time being, the only accepted string for this argument is “CBECC”.
- `pszAuthToken2` is a null terminated string that, combined with `AuthToken1`, identifies the calling application. For the time being, the only accepted string for this argument is “Com”.
- `pszSignature` is a null terminated string containing the base-64 ASCII-encoded signature used to sign request data. Signature processing is not yet implemented, so for the time being the string “none” should be supplied for this argument.
- `pszPublicKey` is a null terminated string containing the base-64 ASCII-encoded public key used to sign request data. Signature processing is not yet implemented, so for the time being the string “none” should be supplied for this argument.
- `pszProxyAddress` is a null terminated string containing the address (i.e. “site.site:port”) of the proxy server to be used in accessing the report generator (via the internet). If no proxy server is needed/used for internet access, then pass NULL for this argument.
- `pszProxyCredentials` is a null terminated string containing the username and password credentials (i.e. “username:password”) needed to access the internet via the proxy server. If no proxy server is needed/used for internet access or if no username/password credentials are needed in conjunction with the proxy server, then pass NULL for this argument.
- `pszDebugBool` is a null terminated string containing either the word “true” or “false”. Passing “true” (the current default) causes the report generator to activate debugging features.
- `bVerbose` is a boolean flag (not in the form of a character string) indicating whether verbose information should be written to the project processing log file.
- `bSilent` is a boolean flag (not in the form of a character string) indicating whether or not dialog boxes are permitted to be presented during the report generation. One example is a dialog indicating that the file needing to be written to contain the compliance report cannot be written to, prompting the user to close the file in another application it is opened in or change the file permissions so that the file can be re-written. A value of ‘1’ will prevent user prompts and issues such as files unable to be written may cause the report generation to be aborted.

- `bUnused` is a boolean flag – pass either true or false – will have no effect on processing.

For the time being, the report file written will be located in the same directory as the analysis results XML file and will be named: “<analysis results filename>-BEES.PDF” (or .XML, depending on the ... argument).

```
int CMX_GetDataString( char* sReturnStr, int iRetStrLen, const char* pszCompParam,
                    const char* pszCompName, BOOL bAddCommas,
                    int iPrecision, const char* pszDefault );

// typedef int (__cdecl *PCMX_GetDataString)( char*, int, const char*,
//                                           const char*, BOOL, int, const char* );
// _CMX_GetDataString
```

where:

- `sReturnStr` is a character string buffer to be populated with data from the building model.
- `iRetStrLen` is the number of characters allocated in the `sReturnStr` buffer. The returned string will be null-terminated, so the maximum number of characters written to `sReturnStr` will be `(iRetStrLen-1)`.
- `pszCompParam` is a null-terminated string identifying the model object and property to return a string representation of. This argument can reference properties of any type in the data model (float, integer, object reference, enumeration or string). The object name and property name are separated by a colon, so this argument would contain 'Proj: CondFloorArea' to return a string-representation of the CondFloorArea property of the Proj (Project) object.
- `pszCompName` is a null terminated string containing the name of the object for which the object/property is to be returned. If there is only a single object of this type in the project (as is the case for Proj, EUseSummary and some other object types), then this argument can/should be set to NULL. If this argument is NULL for objects where multiple are present in the building model, then the property data for the “currently active” object of that type will be returned (not advised).
- `bAddCommas` is a Boolean indicating whether or not commas (or related Windows Locale-based numeric separators) are to be included in the returned string. This argument is only referenced when returning string representations of integer or float object properties.
- `iPrecision` is an integer indicating the decimal precision to be used when populating the return string of float properties. Ignored for non-float properties.
- `pszDefault` is a null-terminated string containing the desired return string in the event the Object:Property being retrieved is not defined in the building model.

Return value is 0 if data retrieval is successful and > 0 if errors occurred.

<error return value info to be supplied at a later date>

Call this routine each time string data is to be retrieved from the building model.

```
int CMX_GetDataInteger( long* pReturnInt, const char* pszCompParam,
                    const char* pszCompName, long lDefault );

// typedef int (__cdecl *PCMX_GetDataInteger)( long*, const char*,
//                                           const char*, long );
// _CMX_GetDataInteger
```

where:

- `pReturnInt` is a pointer to a (32-bit) integer to be populated with data from the building model.
- `pszCompParam` is a null-terminated string identifying the model object and property to return the value of. This argument can only reference properties of integer or enumeration types in the data model. The object name and property name are separated by a colon in this string argument.
- `pszCompName` is a null terminated string containing the name of the object for which the object/property is to be returned. If there is only a single object of this type in the project (as is the case for Proj, EUseSummary and some other object types), then this argument can/should be set to NULL. If this argument is NULL for objects where multiple are present in the building model, then the property data for the “currently active” object of that type will be returned (not advised).
- `IDefault` is a 32-bit integer value which will be returned (via the `pReturnInt` argument) in the event the Object:Property being retrieved is not defined in the building model.

Return value is 0 if data retrieval is successful and > 0 if errors occurred.

<error return value info to be supplied at a later date>

Call this routine each time integer data is to be retrieved from the building model.

```
int CMX_GetDataFloat( float* pReturnFlt, const char* pszCompParam,
                    const char* pszCompName, float fDefault );

// typedef int ( __cdecl *PCMX_GetDataFloat)( float*, const char*,
//                                           const char*, float );
// _CMX_GetDataFloat
```

where:

- `pReturnFlt` is a pointer to a (32-bit) float to be populated with data from the building model.
- `pszCompParam` is a null-terminated string identifying the model object and property to return the value of. This argument can reference properties of float, integer, enumeration or object reference types in the data model (returning a float value for an object reference property returns a 1-based index of the referenced object (among all objects of that type)). The object name and property name are separated by a colon in this string argument.
- `pszCompName` is a null terminated string containing the name of the object for which the object/property is to be returned. If there is only a single object of this type in the project (as is the case for Proj, EUseSummary and some other object types), then this argument can/should be set to NULL. If this argument is NULL for objects where multiple are present in the building model, then the property data for the “currently active” object of that type will be returned (not advised).
- `fDefault` is a 32-bit float value which will be returned (via the `pReturnFlt` argument) in the event the Object:Property being retrieved is not defined in the building model.

Return value is 0 if data retrieval is successful and > 0 if errors occurred.

<error return value info to be supplied at a later date>

Call this routine each time float data is to be retrieved from the building model.

```
int CMX_XXX(      <type> argument, ... );  
  
    // typedef int  (__cdecl *PCMX_XXX)(  <type>, ... );  
    // ?CMX_XXX@@YAH PAMPBD1M@Z
```

where:

- pXXX is a xxx.

```
void ExitBEMProcAndCmpMgrDLLs ();  
  
    // typedef void (__cdecl *PExitBEMProcAndCmpMgrDLLs) ();  
    // _ExitBEMProcAndCmpMgrDLLs
```

Call this routine only once (following all simulations) immediately prior to unloading the DLLs.