# Residential Ruleset Development – Training Webinars – Oct-2014

This document will partially serve as an agenda for the meeting but will also be added to during/following the webinars to provide more details & Q/A.

Recommended webinar prep –

Review of the following sections of a functional requirements document created early in the CBECC-\* tool design/development projects. It is due for some updates, but still has some very useful information. The document has been added to the source repo @:  
 <repository root>\trunk\Documents\CEC ACM Compliance Engine Func Req\_PAC Review dsp.docx  
Ordered from most to least applicable to our webinar:

* Sections 4 thru 4.3 – These sections are a reasonably good overview of the information used to define and manage the data model and ruleset processing.  
  Note: sections referring to the “Data Model Approach ruleset source” do NOT apply to the residential ruleset, so you can ignore sections 4.3.2 - 4.3.2.11. The residential ruleset uses the “Procedural” approach, which organizes the ruleset source quite differently.
* Sections 5 thru 5.2.1.3 – Covers some fundamental rule syntax and evaluation topics. We are not likely to get too deep into rule syntax, but this is still very important to be familiar with.
* Section 2 – Software System Description. Re: 2013 ACM Standards Compliance Engine Software diagram on pg 2-2 -> the same basic modules and structure apply to the residential sector, except there is no Component Library (purple box) and OpenStudio & EnergyPlus groups in the bottom right are replaced by the CSE (California Simulation Engine).

Not included in this document will be a brief webinar opening describing our objectives and allowing each attendee to **introduce themselves**.

Touch base on **screen/webinar resolution** – make sure clear picture for all attendees

Abbreviations used in this document:  
- frs – functional requirements spec document listed above in the webinar prep section  
- idm – input data model – documentation of all valid objects and properties that can be user-specified

**Tools to use** when performing ruleset authoring/testing/debugging:

* **Text file editor** geared toward programming. EditPadPro & UltraEdit common among ruleset authors. Some useful features: line number labeling & navigation / auto-indentation / syntax coloring schemes / search/replace / etc.
* **GREP** tool – searches for text across multiple files & subdirectories. Some popular options are grepWin (free, usable) & PowerGrep (very powerful, but expensive). Enables you to search for object & property names, error messages, etc. across all ruleset source.
* **Text file comparison/merging** tool. There are some decent free tools available here, but a favorite is BeyondCompare ($30). There is a good comparison tool integrated directly into the TortoiseSVN Windows shell, but that only works on files in the source repo, and it will be important to be able to compare other files, such as project analysis log files and files in the source repo vs. versions of the same file outside the repo.

Topics to be covered:

## **CBECC-Res Source Code Repository**

Separate document circulated prior to the webinar describes what the repository is and how to connect/interface with it.  
This document is stored/maintained in the repository @:  
 <trunk>\Documents\CBECC-Res Code Repository.docx

For now, this repository stores program releases and source code that compiles into the residential data model and ruleset. In the near future, this will be expanded to include a greater number of test/sample runs and batch files to facilitate their analysis in groups of runs. And later still this or another repository will be used to post the C++ source code that compiles into the CBECC-Res and ruleset compiler executables (but this is not yet posted to open source).

* Checking out the repository – questions
* One-time installation of CBECC-Res may be needed to run CBECC-Res directly from the repo directory (on your machine), since not all system dependencies are included in the repo.

## **Data Model & Ruleset Structure**

All data model and ruleset source files are text and many have headers that are reasonably well self-documented.

**Object classifications & relationships**:

* **Global** objects (frs 5.2.1.1) – object types for which only one can be created. Examples include Project, INISettings, Garage, Attic, Crawlspace (one or more of these last three are likely to become multiple-occurrence objects, in which case they will no longer be global.
* **Parent/Child** (frs 4.1.1) – objects that are displayed hierarchically in the CBECC-Res object tree, such as Zone / ExtWall / Window. A child object can only have a single parent object.
* **Referenced** objects (frs 4.1.2) – objects that can be referenced by many other objects, for instance Construction objects that can be referenced to describe a large number of individual ExtWall objects.

Data model and ruleset source use common conventions for including **comments** within these files. Valid comment markers are ‘;’ and ‘//’. Any text following either of these characters (except when included in a quoted string) will be characterized as a comment and ignored in the compiled data model or ruleset.

The Residential Data Model source is made up of two files that compile into <Data path>\Rulesets\CA Res 2013\CAR13 BEMBase.bin

* **CAR13 BEMBase.txt** (frs 4.2.1) – main data model definition file. Two primary types of records describe Object Types & Properties – syntax documented in the file header.
* **CAR13BEMEnums.txt** (frs 4.2.2) – definition of all enumeration (list selection) strings – syntax also documented in header. Enumeration lists can be conditional, in part to provide context-sensitive selections (dependent on other model data) and also to facilitate backward compatibility when selection string change.

Most objects and properties can be specified by users or supplied as inputs to the compliance analysis by third party tools and other properties are either used exclusively for simulation, reporting or to facilitate other compliance analysis functions. Those that can be specified by the user are documented in the **Input Data Model** (idm) text file which third party software vendors use (and compare between versions) to manage the translation of their own native program data to valid compliance inputs. This idm file is automatically produced during ruleset compilation and is written to:  
 <Data path>\Rulesets\CA Res 2013 - Input Data Model.txt

The compliance ruleset source spans over a dozen files which compile into <Data path>\Rulesets\CA Res 2013.bin Some of the ruleset source files include:

* **Datatype**.txt (frs 4.3.1.4) – contains property classifications, such as Required, Prescribed, NotInput and others
* **Ranges**.txt (frs 4.3.1.5) – defines valid numeric ranges and associated error messages for some numeric properties
* **Resets**.txt (frs 4.3.1.7) – identifies lists of properties that will be re-defaulted when certain other building data is modified by the user (only applicable to CBECC-Res or other apps that may interact with the compliance engine during user input)
* **Library**.txt (frs 4.3.1.8) – contains various objects that can be pulled into standard building models during analysis
* **Look-up tables** (txt or CSV files) (frs 4.3.1.3) – contain tables of data that can be referenced by rules. Many are included in the ruleset, some use an old, numeric-only syntax (i.e. ClimateZone.txt – values referenced as: “ClimateZoneTable( Proj:ClimateZone, 1 )”) and some use a new syntax that allow for string or numeric look-ups and return values, blank fields & error/warning messages (i.e. CA13HeatingEquipment.csv – values referenced as:  
   CA13HeatingEquipment:AFUE( "StdVer", Proj:StandardsVersion, "SysType", TypeAbbrevStr )
* **Subordinate Rulelist files (\*.rule)** – contain one or more lists of rules, each designed to manipulate the building model in a certain way. Current .rule files are organized by step/phase of analysis and building topic (construction, HVAC, etc.)
* **Main Rules text file** (frs 4.3.1.1) – this is the master file that references each individual file to be included in the ruleset (from the list above). This file can also contain lists of rules, like the .rule files.

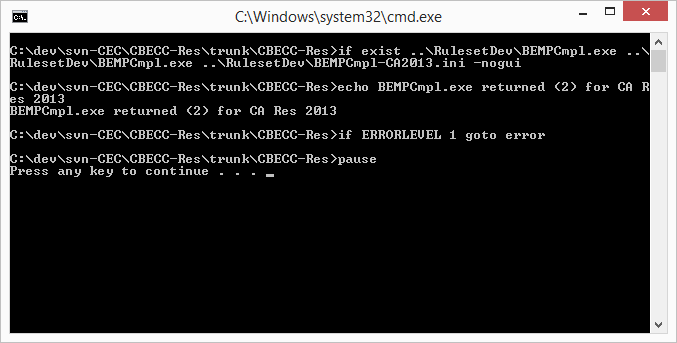
## **Compiling Data Model and Ruleset Source Files**

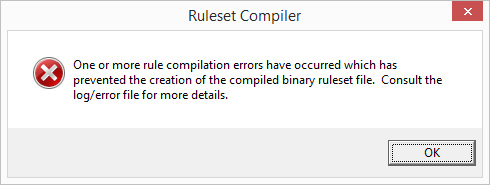
Data model and ruleset source (text/CSV) files are compiled into binary files using the ruleset compiler:  
 <repository root>\trunk\RulesetDev\BEMPCmpl.exe

This program is a simple two-function interface with inputs described in the above sections. Whenever changes are made to the data model source files, then both the data model and ruleset binaries should be re-compiled. If ruleset source is modified but data model source is not, then on the ruleset binary need be re-compiled.

HOWEVER – what we now recommend is to simply compile both binaries using a batch file that is setup to reference ruleset source in the repository and install the compiled binaries in the directory where the repository CBECC-Res program references. The 2013 ruleset compiler batch file is:  
 <repository root>\trunk\CBECC-Res\**CompileRules\_CA2013\_noUI.bat**

Each time the ruleset is compiled, a .log file is generated that provides some useful information about the compiled rules and whether errors were encountered. At the bottom of this file is a list of all RuleLists compiled into the ruleset, their source filename, line number, number of rules and rule evaluation flag values.

If errors are encountered during rule compilation using the .bat file, then the DOS command window remains displayed and requires you to press a key to close it (shown to the right). This is your indication that an error occurred and you should review the log for more information. If the ruleset compiles successfully, then the command window disappears without requiring a key to be pressed.

If an error occurs when running the BEMPCmpl.exe program directly, then you are informed of this failure by the messagebox shown to the right.

An example of a ruleset compilation error is shown below. The filename and line number should enable you to quickly locate the errant rule, but sometimes it is challenging to understand exactly what the problem is – in which case you should seek assistance from fellow rule authors.

Rule 348, 3, **'Rules.txt' Line 213**: Set Proj:RptGenUIVer

Proj:RptGenUIVer

Error: Syntax error near 'j:RptGenCompReport" Proj:RptGenCompRepo'

Full expression:

"Set Proj:RptGenCompReport" Proj:RptGenCompReport = { if (RunScope == 1) then  
 "CF1R\_NCB\_PRF" else "CF1R\_ALT\_PRF" endif

## **CBECC-Res Screen Definitions**

Unlike most Windows programs, the majority of the CBECC-Res dialog interface are defined in text files that are maintained with ruleset source files. These files are commonly modified by ruleset authors and can be altered and referenced by CBECC-Res without having to re-compile or even close the application. These files are maintained in the <repository root>\trunk\RulesetDev\Rulesets\CA Res\ directory, they are copied as part of the ruleset compilation batch file into a corresponding directory under \trunk\CBECC-Res\Data\ and include:

* **CAR13 Screens.txt** – defines the input fields, labels and most other aspects of the dialog interface, including conditions that determine when a field may be hidden vs. displayed and editable by the user. The dialog size is one of the few screen-related changes that require the CBECC-Res C++ source files to be recompiled. Review the screens.txt file header for information about the syntax of the screen definitions.
* **CAR13 ToolTips.txt** – includes tooltip messages for many of the input properties in the data model. Tooltip messages defined in this file are only used for screen fields for which a message string is not defined (Screens.txt messages override these Tooltip messages).

## **CBECC-Res Project & Detailed Input Files**

There are two different compatible formats for files that can be read into and written by CBECC-Res or the compliance engine - .ribd files and .xml files. RIBD (residential input building description) files are the original/native format of the COMcheck-Plus application (the starting point for CBECC-Res & -Com). Both the **RIBD & XML formats** are fundamentally text files, but the XML format includes starting/ending tags for every object and property definition and are more consistent/compatible with modern software apps.

Normal project input RIBD or XML files include only those properties that can be specified by a user. There are also **“detailed” versions of each file type** that echo the value of each and every property defined in the database at the time the file was written, and the IBD-detail file also includes tags that indicate the status of each properties data (user defined, ruleset-defined, program default and simulation result are the most common). Writing of these detailed files can be toggled on in the program INI file (as described below) and are frequently referenced when debugging ruleset issues.

## **Compliance Analysis Processing**

Steps performed during analysis are detailed in a spreadsheet posted in the source repository:  
 <repository root>\trunk\Documents\**BEMCmpMgr - CA Res 2013 Compliance Analysis.xlsx**

The majority of the files produced during analysis are stored in a folder under the main project folder by the name “<project filename> - Comp”. INI file options (described below) exhibit some control over which files are created and whether they are retained following completion of each analysis run. Files whose names end in **“- Prop”** describe the proposed design model and files ending **“- Std”** describe the standard design. The most pertinent files from a ruleset authoring/debugging perspectives are:

* **.ibd-Detail** files – are “detailed” building database files that echo data describing every object and property defined in the database along with the status of each property. This is contrary to typical project files that include only user input data and do not specify each property’s status.
* **.ibd-b4Evals** files – similar to the detailed model echoes above, but written prior to evaluating rules that finalize the corresponding models.
* **.cse** files – CSE simulation input files
* **.rep** files – output (non-hourly) report files from the CSE simulation
* **.csv** files – hourly CSE simulation energy use results
* **.err** files – CSE simulation error messages

Once the proposed and standard models are produced, detailed versions of these building databases are written to

## **CBECC-Res Application**

Three main executables:

* **CBECC-Res** – the end-user application used to test compliance of a residential building design. This tool is available for the public to use, but many use other proprietary products to perform this analysis.
* **CEC Compliance Engine/Manager** – the program library (collection of DLLs) used by CBECC-Res and other third party tools to perform CA Title-24 compliance analysis for residential buildings. All tools certified for Title-24 residential analysis MUST use this compliance engine (and ruleset) to test building compliance.
* **Compliance Ruleset Compiler** – a small application used to convert the ruleset source (text files) into binary files distributed to end users. These binary files contain all data model and “rules” needed to test energy code compliance of residential building designs.

Refer to the Quick Start Guide and User Manual for documentation on use and features of CBECC-Res.

To can toggle certain program features on/off by modifying one of the **two CBECC-Res13.INI files**. The INI file in the main program directory only includes a single entry pointing to the program’s <Data> path and should not be changed unless you wish to manually relocate the data folder. The **CBECC-Res13.INI located in the Data folder** is the one that contains a variety of UI and analysis options, many of which are useful when authoring/debugging rules. Many of the options present in the Data\CBECC-Res13.INI file checked into the repository are set to values that assist in program testing and ruleset development and are not consistent with the version distributed with the end-user application. Some of these options include:

* **DeveloperMenu** (repo setting 1, user setting 0) – controls access to a variety of Tools menu bar options designed to help facilitate program and building model debugging.
* **EnableResearchMode** (repo & user commented out, default 0) – enables access to a “Research” option specified for a project’s Analysis Type selection. This option is designed to allow a user to simply simulate the model described in the project, without performing compliance analysis – BUT this option is not yet fully functional.
* **EnableRptIncFile** (repo setting 1, user setting 0) – controls access to a Report Include File field on the Project / Analysis dialog tab. This feature allows someone to include CSE inputs specified in an existing CSE input file/snippet in each CSE input generated during analysis. This is mainly used to specify more detailed output reporting and is not generally used in ruleset testing/development.
* EnableVarFlowOAV & EnableFixedFlowOAV – settings used to toggle on certain HVAC system options, which now are toggled on by default, so not likely to be used in the near future.
* **EnableRulesetSwitching** (repo setting 1, user setting 0) – controls access to the ability to switch rulesets associated with a project file. For now, this switching is restricted to rulesets known to be compatible with each other – namely the 2013 and 2016 (test) rulesets.
* **BypassRuleLimits** (repo & user commented out, default 0) – turns off many restrictions and range limits defined in the analysis rules. This feature is used mainly by development team members that are performing analysis to test analysis options not currently allowed in 2013 compliance analysis.
* **StoreBEMDetails** (repo setting 1, user setting 0) – controls writing of “detailed” model files both after a project is first read into the CBECC-Res UI and also during analysis (as described above in the Compliance Analysis Processing section – writing of .ibd-Detail files). When toggled on, files written to the project folder by the names “<project name> - post open.ibd-Detail” and “<project name> - post open.xml” and echo all database objects and properties defined following normal (rule-based) defaulting.
* AnalysisRangeChecks (repo & user commented out, default 0) – turns off numeric range checks during analysis.
* **LogWritingMode** (repo & user commented out, default 2) – toggles project log writing mode between ‘0’ (opens, writes data, flushes file and closes with each entry written to log file) and ‘1’ (log file left open as long as project remains open). The default value of ‘2’ (or unspecified) will result in a mode of ‘0’ if no detailed logging is specified in other INI settings, or ‘1’ if other INI settings do specify detailed logging. Performing analysis w/ detailed logging is significantly slower with this setting @ ‘0’, but the default for no detailed logging of ‘1’ helps to ensure that ALL data is written to the log file before the software move on to other tasks.
* **VerboseInputLogging** (repo & user commented out, default 0) – turns on verbose project logging only during evaluation of the project defaulting rules. See below for more information about verbose logging.
* **LogRuleEvaluation** (repo & user commented out, default 0) – turns on verbose project logging during compliance analysis. See below for more information about verbose logging.
* **DebugRuleEvalCSV** (repo commented out, not present in release version) – activates selective rule evaluation logging which can provide maximal detail on only the properties and/or rulelists of interest. Logging of properties using this option are more detailed, listing occurrences not only of setting the property but also each time the property is referenced while evaluating any rule. LogRuleEvaluation is typically set to 0 or commented out when activating this feature, so that only the specified properties/rulelists are logged  
  The expected setting here is the (relative) path and filename of a CSV file listing the rulelists and properties to be logged. Examples are present in the SVN repository @ trunk\CBECC-Res\Data\RuleDebug\\*.csv
* **BypassCSE** and **BypassDHW** (repo commented out, not present in release version, default 0) – prevents each type of simulation from being performed. Useful when writing of detailed building model or simulation input files is desired but no simulation or analysis results are needed. Analysis fails due to a variety of errors caused by the lack of simulation results, but only after proposed and standard models are generated.

Text files by the name **“<project name>.log”** are written to during UI open/defaulting and analysis. Some messages logged here document when the project is opened or saved and any errors that occur, whether or not they are reported to the user, also get written to this file. These error messages are the first place to look when debugging a defaulting or analysis problem. For example:  
 Error: Undefined data: left side of '<=' evaluating rule: Rule 139, 16, Line 12463: Create  
 Zone:CeilAirNetIZXFERs[2] – CC

All error messages generated through the evaluation of rules should identify the ruleset source filename and line number, enabling you to identify exactly which rule triggered the error message. In some cases however, it is unexpected/unintended evaluation of a previous rule that lead to the logged error, so more investigation is often needed.

Some errors are routine/expected when running analysis using executables that are not part of a certified public release, for example:  
 File hash check error - failed consistency check (4) (analysis continuing w/ report signature  
 disabled) - for file: E:\Apps\CBECC-Res13-NEW\BEMCmpMgr\_noui.dll

This error is reporting that the BEMCmpMgr\_noui.dll is not one of the executables from the last certified release – which is expected when testing/developing rules in between public release versions.

When **verbose project logging** is toggled on, the project log file can quickly grow from < 100 kB to tens of megabytes. The evaluation and return value of each and every rule is logged during project defaulting within the UI (when INI flag VerboseInputLogging > 0) and/or analysis (when INI flag LogRuleEvaluation > 0). An example of such logging is shown here:

2014-Sep-18 08:53:49 - Evaluating Compliance Rulelist 'ProposedInput'...

2014-Sep-18 08:53:49 - EVALUATING RULELIST: ProposedInput (on all components)

2014-Sep-18 08:53:49 - Rule 1, 1, Line 247: Set Proj:CompMgr -- List: 'ProposedInput'

2014-Sep-18 08:53:49 - 'Multifamily Apartments' Proj:CompMgrVersion -> BEMCmpMgr 2013-3 (651) (rulelist: 'ProposedInput')

2014-Sep-18 08:53:49 - Rule 1, 2, Line 249: Set Proj:RptGenUIApp -- List: 'ProposedInput'

2014-Sep-18 08:53:49 - 'Multifamily Apartments' Proj:RptGenUIApp -> CBECC-Res (rulelist: 'ProposedInput')

2014-Sep-18 08:53:49 - Rule 1, 3, Line 250: Set Proj:RptGenUIVer -- List: 'ProposedInput'

2014-Sep-18 08:53:49 - 'Multifamily Apartments' Proj:RptGenUIVer -> 3 (rulelist: 'ProposedInput')

Setting these verbose logging INI settings to a value of 1 causes logging identical to that listed above and setting the value to **2 provides the same logging but without the timestamps**, which is useful in facilitating comparison of rule evaluation logs before/after rule edits.

Other CBECC-Res features particularly useful for ruleset developers include:

* Hovering over fields in the dialog interface present **tool tips** that can include a message, numeric range information, the status of the data installed to that property and the Object:Property descriptor (some of which can be toggled on/off via INI file options).
* **Analysis results** from each and every run performed by a CBECC-Res installation are appended to two separate CSV files, one in the project file folder by the name <project filename>.log.csv and one in the main program <Data> folder (stores results across all projects).

## **Rule Authoring**

Critical reading for rule authors – as mentioned near the beginning of this document: Sections 5 thru 5.2.1.3 of trunk\Documents\CEC ACM Compliance Engine Func Req\_PAC Review dsp.docx – Covers fundamental rule syntax and evaluation topics

* **Procedural** vs. Data Model are two different approaches to organizing a ruleset. As you review the functional requirements document, keep in mind that the Title-24 residential rules use the Procedural approach.
* Rule target (left side) – typically just ‘Object:Property’, but can also include multiple referenced objects.
* Referencing building model data – Global, Parent, Local, Referenced & Child data access (sections under 5.2.1).
* Executable statements (section 5.2.3) – if/then/else, switch, nesting statements.
* Operators and Syntax (section 5.2.4).
* String formatting (section 5.2.5).
* Function reference (section 5.2.6) – mostly complete listing of expression functions (in need of some updates).

And some topics/functions to highlight:

* **Rulelist definition flags** (4 settings (0/1) following each rulelist name):  
  - #1 - whether or not the rules in the list are to be evaluated regardless of the datatype flags. A value of 1 means each rule is evaluated no matter what the datatype is set to. A value of 0 means that each rule is evaluated only if its datatype is "Prescribed" or "NotInput" or if the value currently installed in the database was NOT either Library or User defined.  
  - #2 - rulelist evaluation iteration flag – IGNORED in residential ruleset.  
  - #3 - whether or not the values set by the rules in the rulelist are to always be classified as User-Defined (1) vs. the normal ruleset-defined flags (0).  
  - #4 whether or not SetBDData RESETS are to occur (0, the default) or are to be bypassed (if 1).
* Special/reserved property names:  
  - **Name** can be used to reference or change the name of an object.  
  - **Parent** can be used to set the parent of an object, such as setting the parent ExtWall of a Win.  
  - **Action** can be used as a rule target when the purpose of a rule is to execute some action rather than set data back to the building model, for example posting an error or warning message to the project log or evaluating a subordinate rulelist.
* **EvalRulelist**( “RulelistName”, <0/1> ) function causes another rulelist to be evaluated. If called with a second (optional) argument of ‘1’, then the rules included in that subordinate rulelist will only be evaluated on the LOCAL object (of the rule that the EvalRulelist() function was called).
* Ruleset functions commonly used when debugging rule evaluation:  
  **- MessageBox**( <same variable arguments as Format()> ) – causes a dialog to pop-up during rule evaluation containing the message defined by the function argument(s).  
  **- StoreBEMProc**( “string added to file extension”, 0/1 ) – causes a building model (.ribd) file to be exported containing the model at that point in the analysis. Specify a ‘0’ as the second argument to export the “detailed” building model, or ‘1’ to output a file consistent w/ a project input file.  
  Make sure to remove or comment out rules containing these features prior to committing rules to repository.
* **Undefined Data** – directly referencing properties that are undefined at the time of rule evaluation will cause errors unless those properties are referenced in function calls that are compatible with undefined data, such as:  
  - IfValidAnd(), ValidOr()  
  - GlobalValid() / Parent\*Valid() / LocalValid()
* Unique to CBECC-Res – upon each File-Save action, all ruleset default data that can be defined by the user (Compulsory, Required, Optional or Default (as specified in DataType.txt)) will be toggled to User Defined and included in the saved project file.