DEER2017 and DEER2018 Update Summary Appendix

Contents

Appendix – Simplified DEER Processing Utilized for DEER2017 and DEER2018 Residential Update Value Development

1 Overall Process

The eQUEST wizard based process used in previous DEER version to build simulations has been replaced. The old process was difficult for DEER users to examine and discover all the assumptions underlying building vintage prototypes and measure definitions. A new process has been developed that uses standard DOE-2 building description language (BDL) rules in conjunction with an SQLite database to create simulation input files.

A diagram illustrating the new process is shown in Figure 13. The basic building blocks for the process are defined as follows:

- Bare Bones input files are DOE-2 input files that are limited primarily to definitions of geometry and configuration of HVAC systems. For the residential sector, there is one bare-bones file for each building type. While these are complete DOE-2 models, they do not contain the vintage and climate zone specific data that make up the DOE-2 models used in the analysis.
- SQLite data tables contain building parameter data that are organized by building type, climate zone, vintage and HVAC system type. These tables contain the building parameters that are used to convert the bare-bones input files into the climate zone and vintage specific models, called the Initialized Prototypes.
- Initialization Rule Lists are executable routines that use standard DOE-2 rules processing to convert the bare-bones input files to Initialized Prototypes. The initialization rule lists are stored in the SQLite database. The initialization process includes baseline initializations that are not related to specific technologies, and also technology initializations. The technology initializations are listed in the TechInit table in the SQLite database.
- Initialized Prototypes are baseline building models. There is one Initialized Prototype model for each valid combination of building type, climate zone, vintage, and HVAC system type.
- Technology Rule Lists are executable routines that use BDL rules to convert the Initialized Prototypes to Technology Runs. The technology rule lists are stored in the SQLite database. These rules access technology data tables that are stored in the SQLite database.
- Technology Runs are simulations that represent the application of specific technologies. A measure is determined by comparing a measure technology with a pre-existing baseline technology and/or a standard technology. Each technology variation is run for each valid combination of building type, climate zone, vintage and HVAC system type. The results of the technology runs are stored in a separate output database file.

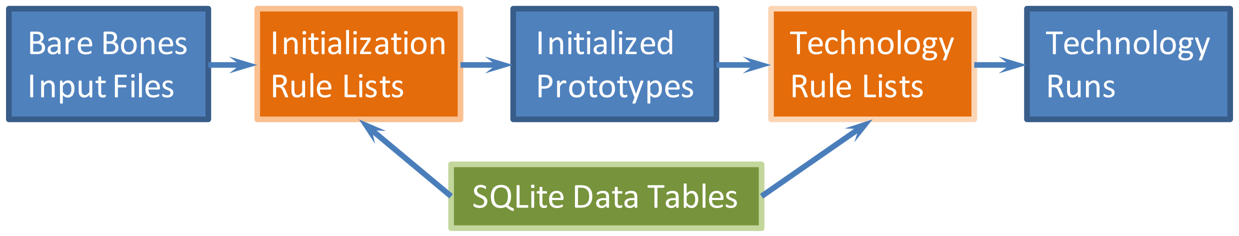

Figure 1. DEER2017 Process schematic

The simulation process for DEER2017 and DER2018 is managed by an update to the MASControl software used for the previous DEER versions. The interface allows the user to select a range of parameters to include in a set of simulations. The interface also includes a general setup page to enter details such as locations for the SQLite databases, locations of simulation files, and options for logging information about the simulations. MASControl2 utilizes the latest version of DOE-2, which includes improved capabilities for modeling whole house fans and radiant barriers. The installation software for MASControl2 can be found on the DEER2017 page of the DEEResources.com web site.

2 DEER2017 Process for Residential Simulations

This section provides detailed listings to describe the key components of the updated process that was used to perform the residential DEER simulations.

2.1 Applicability Tables

When selections are made in the MASControl2 interface, simulations are performed only for cases that are determined to be applicable. The following tables in the input SQLite database define the levels of applicability.

- MeasureApplicability: determines applicability of measures based on building type, vintage, climate and HVAC type; also determines applicability of each run case: pre-existing, standard and measure.
- VintageApplicability: determines applicability of vintages based on building type.
- DEER_fBldgTypeHvacType: determines applicability of an HVAC Type to a given building type.

2.2 Prototype Initialization Rule Lists

Before technologies can be simulated to create a measure, the prototype must be initialized. The residential prototype initialization is started by running the rule list ResProtoInit. This in turn calls a number of other rule lists. The complete set of rule lists involved in the residential prototype initialization is provided below.

- ResSpace: Set internal loads for space.
    - GetLivingBedroomArea: calculate conditioned area of residences.
    - GetResIntLoadData: given the type of load, lookup the load data from the database.
    - SpacePlug: initialize miscellaneous plug loads.
    - ResCookElec: initialize cooking electric loads.
    - ResCookGas: initialize cooking gas loads.
    - ResHouseSpace: set occupant loads for residential spaces.
    - ResInfAirChange: set up infiltration for spaces that use the air change method.
    - ResInfAshraeEnhanced: set up infiltration for spaces that use the ASHRAE enhanced method.
    - ResInfResidential: set up infiltration for spaces that use the RESIDENTIAL method.
- MFmInit: do initializations that are unique to multifamily.
    - MFmPubSpace: set internal loads for multifamily public spaces.
- CalcUFloors: calculate fictitious U-value for slab floor.
    - CalcLayersR: calculate combined R-value of construction layers.
    - CalcUFloorX: calculate U-value of slab floor.
- ResTstat: set thermostats based on building type, vintage and climate.
- ResSystem: set baseline system properties.
    - SetNumStories: set number of stories for this system.
    - CalcHouseFloorArea: calc occupied floor area for this system.
    - ResSysPVVT
        - ResSysHP set heating and cooling capacities for HP system.

    - ResSysAC: set heating and cooling capacities for AC system.
- ResZone: assign thermostat set points.
- CreatOnPkKWReport: set up hourly report for calculating peak period kW.

2.3 Input Data Tables

The initialization rules make extensive use of data stored in the SQLite database, as described by the following list of tables.

- DEER_fActArea: list of properties that vary with activity area.
- DEER_fBldgType: list of building types. The field DEERGen is used to enable specific building types in the MASControl2 interface.
- DEER_fBldgTypeHvacType: list of properties that depend on both building type and HVAC type. The name of the Bare Bones file that is used as the starting point for the initialized prototype is listed in this table.
- DEER_fClimate: list of properties that vary with climate zone. PkPerFirstHr is the first hour of the year for peak period calculations in that climate zone.
- DEER_fHVAC: list of valid HVAC system types. Entries that are valid for residential have HVACType codes that start with the letter "r".
- DEER_fVintage: list of vintage ID codes and descriptions.
- DEER_TStat: valid thermostat index selections based on building sector.
- DesDay: lists design day weather for each climate zones.
- HVAC_fActArea: HVAC properties that vary with activity area.
- HVACDefault: lists default HVAC type for each building type.
- Parameters: list of parameters that are set in the DOE-2 input file during prototype initialization.
- PreTechInit: Provides the name of a rule list that is run before technology rule lists during technology initialization.
- ResInfData: Definition of infiltration parameters based on activity area.
- ResIntLoadData: Parameters to define baseline lighting and equipment loads for residential building types.
- ResTstat: Thermostat setpoints developed in calibration of the residential prototypes.
- SchID_fActAreaBldgType: Provides ID used in schedule names based on building type and activity area.
- SchNames: Alternate list of schedule names based on building type and activity area.
- TechInit: List of technologies that need to be initialized when creating the initialized prototype.

2.4 Measure Tables

A measure is defined by comparing technology runs. The typical measure is comprised of three run cases: the pre-existing baseline technology (Pre), a standard technology (Std, based on T-24) and a measure level technology (Msr). It is also possible for a measure to be defined by combinations of measures. For example, a dishwasher technology is combined with a water heating technology to create the dishwasher measure. The following tables describe how technologies are assigned to measures.

- Measures: this is the complete list of measures. For simple measures that have a single technology for each run case, the technology ID codes are provided in this table. More complicated measures have reference ID codes that connect to the VariTech and TechList tables described below. An additional field that is used for some measure cases is the SimQualifier, which defines the manner in which the technology is applied. For example, refrigerators can have a SimQualifier of "Living" to indicate that the refrigerator is in the living space or "Uncond" to indicate that the refrigerator is in the unconditioned garage.
- VariTech: this table provides information about lists of technologies that are applied to measures according to the key parameters of building type, vintage, climate and HVAC type. The LookupID field in this table connects to the VeriTech fields in the Measures table or the TechLists table.
- TechLists: this table allows the application of multiple technologies to a single measure case. The LookupID field in this table connects to the TechList fields in the Measures table. The VariTechLookupID field in this table allows the combination of multiple technologies with the concept of technologies that depend on the key parameters.

2.5 Technology Types and Rule Lists

Once the MASControl2 program has initialized the prototype and identified the appropriate technologies to simulate for a measure, the technologies must be initialized for each measure run case. The technology initialization is started by running a single rule list, which may in turn call other rule lists. The SQLite data tables and rule lists that are used for technology initialization are described below.

- Technology Table: this is a complete list of valid technology identification codes (TechIDs). The table is used by the process to determine the TechTypeID for a given technology.
- TechType Table: the following fields are used by the MASControl2 process:
    - TechRuleList: the name of the rule list that initializes the technology.
    - DOE2ParamName: the name of a DOE-2 PARAMETER entry that will store the TechID.
    - MeasAreaRuleList: calculates the floor area to which the measure is applied.
    - NormUnitsRuleList: calculates the normalizing units for results processing.
- TechID2Param Table: for a given TechID, this table lists the names of all the relevant parameters and their values.
- Technology Rule Lists (stored in the SQLite table BDLRules):
    - ResPreTechInit: rules to be run before technology initialization.
    - CathRoofR: Insulation level for cathedral roof.
        - CalcCombinedR: determine combined R-value of framing and fill insulation; may also include an additional layer of continuous insulation.
    - FlrAboveCrawlR: Insulation level for floor above crawl space.
        - CalcCombinedR: see above.
    - RefgChg: Refrigerant charge for air conditioning system; can be overcharged, undercharged, or correctly charged.
    - res_AtticFloorR: insulation level in attic floor.

    - CalcCombinedR: see above; for attics, this also accounts for an edge effect due to constriction by the sloped roof.
- ResCloWash: clothes washer efficiency technology.
    - ResCloWaLiv: applies clothes washer loads to the ResLiving space.
    - ResCloDrier: set up clothes dryer loads.
- ResDishWash: dishwasher efficiency technology.
    - ResDWLiv: set SPACE keywords for dishwasher loads in ResLiving.
- ResDuctAirLoss: duct air loss technology.
- ResDuctIns: duct insulation technology,
- ResEHNC: technology for HVAC system with electric heat and no cooling.
    - ResEHNC_Zone: set ZONE keywords for electric heat/no cooling.
- ResGasFurnace: technology for residential gas furnace.
- ResGFNC: technology for HVAC system with electric heat and no cooling.
- ResGlass: set glass and window technologies.
    - ResGlassGtcMethod: set GLASS-TYPE keywords for GLASS-TYPE-CODE method.
    - ResWinGtcMethod: set WINDOW keywords for GLASS-TYPE-CODE method.
    - ResGlassScMethod: set GLASS-TYPE keywords for SHADING-COEF method.
    - ResWinScMethodset: WINDOW keywords for SHADING-COEF method.
- ResLtg: implement lighting technology.
    - SetResLtg: set SPACE keywords for lighting technology.
- ResNatVent: handles natural ventilation and whole house fan technologies.
    - ResNatVent1: sub-rulelist applied only to systems that are not public.
- ResRadBarrier: implement attic radiant barrier technology; baseline technology is no barrier.
- ResRefg_Living: refrigerator technology with "Living" sim qualifier to limit to units that are installed in the kitchen.
    - ResRefgLiving2: sub-rulelist for refrigerator in kitchen.
- ResRefg_Uncond: refrigerator technology with "Uncond" sim qualifier to limit to units that are installed in the garage.
    - ResRefgGarage2: sub-rulelist for refrigerator in garage.
- ResWallIns: wall insulation technology.
    - CalcCombinedR: calculate insulation effective R-value from fill insulation properties and continuous insulation properties.
    - ResWallUo: calculate insulation effective R-value from wall overall U-value.
        - ResWallUo1: sub-rulelist to calculate R from Uo.
            - SetInsLayerIdx: identify the material that represents the insulation in the LAYERS command.
            - SetInsLayerR: calculate insulation R-value based on difference between overall U and R-values of all the layers.
- ResWtrHtInit_InUnit: setup water heater that is located in the living unit.

- ResWtrHtInit: setup water heater as part of an appliance measure.
    - PopulateResDhwCircLoops: set keyword values for DHW loops.
        - GetResDhwData: lookup data for DHW model.
    - ResDwHeater: set keyword values for DHW heaters.
    - SetSFmWtrHtMeters: assign meters to water heaters for SFm.
- SEERDxPerfMap: AC or HP efficiency technology.
    - SEERDxPerfMap1: set keywords for SEER rated AC and HP systems.

2.6 Output Data Tables

Simulation results are written to a separate SQLite database (default name is DEER_Results). There are several tables in the database with different types of results. For residential buildings, the key tables are as follows.

- ip_results: contains annual simulation results for each initialized prototype run.
- ip_techids: contains a list of TechIDs that is included in each initialized prototype.
- tech_results: contains annual simulation results for each technology run.
- measure_runs: one record provides references for each run case (Pre, Std, Msr) to indicate whether the results are to be found in the ip_results table or the tech_results table. If the results are in the tech_results table, then the TechRefID is listed in the appropriate RefID table of the measure_runs record.

3 Measure Energy Impact Processing

The MASControl2 processing produces two tables of results: a table of Initialized Prototype results and a table of Technology results. All of the DEER measure energy impact results are based on the data in these two tables. To complete the processing, these two tables are copied to a PostgresQL database. For residential buildings, the two results tables include the 5 thermostat scenarios that need to be weighted based on calibration weights. The first function listed below weights the individual thermostat results for a given set of results and creates the weighted results records. A similar function is used to weight the Initialized Prototype results.

The second function listed below calculates the energy impacts for measures based on measure definitions and individual technology results.

3.1 Function to weight technology results

```
-- Function to weight the residential tech_results records for a given techid based on the tstat
--  results are added to tech_results_wtd table,
--  existing results for the specified techid are deleted if they exist before new results are added
--  if tstat 1,2,3,4,5 don't exist for given primary keys, the weighted record is written with tstat = -9
indicating an error in processing
--  if weighting is successful, the weighted record has tstat = -1
-- Note: must run "Delete Duplicated Records in tech_results" query prior to running this query.
DECLARE
    TechResRec support."tech_matrix_ip"%ROWTYPE;
    WtdResRec  simresults."tech_results_wtd"%ROWTYPE;
    NumTstats INTEGER;
    NumRecs INTEGER;
    TotProc INTEGER;
    Wt FLOAT;
    sumWt FLOAT;
    measarea FLOAT;
    numunits FLOAT;
    kwh_tot FLOAT;
    kwh_ltg FLOAT;
    kwh_task FLOAT;
    kwh_equip FLOAT;
    kwh_htg FLOAT;
    kwh_clg FLOAT;
    kwh_twr FLOAT;
    kwh_aux FLOAT;
    kwh_vent FLOAT;
    kwh_venthtg FLOAT;
    kwh_ventclg FLOAT;
    kwh_refg FLOAT;
    kwh_hpsup FLOAT;
    kwh_ext FLOAT;
    kwh_shw FLOAT;
    thm_tot FLOAT;
    thm_equip FLOAT;
    thm_htg FLOAT;
    thm_shw FLOAT;
    kwpp_tot FLOAT;
    kwpp_ltg FLOAT;
    kwpp_equip FLOAT;
BEGIN
 -- Delete all weighted results for the specified tech:
 DELETE FROM tech_results_wtd WHERE  tech_results_wtd.techrefid = $1;
 TotProc = 0;
 FOR TechResRec IN SELECT * FROM support.tech_matrix_ip
  WHERE techrefid = $1 and tstat = 1
  LOOP
    -- verify that there are 5 stats (1 - 5) to work with
    SELECT COUNT(*) Into NumTstats FROM tech_results
        WHERE  tech_results.techrefid = TechResRec.techrefid
        and  tech_results.simqual  = TechResRec.simqual
        and  tech_results.bldgtype = TechResRec.bldgtype
        and  tech_results.bldgvint = TechResRec.bldgvint
        and  tech_results.bldgloc  = TechResRec.bldgloc
        and  tech_results.bldghvac = TechResRec.bldghvac
        and  tech_results.tstat > 0 and tech_results.tstat < 6;
    IF NumTstats = 5 THEN
       TotProc = TotProc + 1;
       sumWt = 0;
       measarea = 0;
       numunits = 0;
       kwh_tot = 0;
       kwh_ltg = 0;
       kwh_task = 0;
       kwh_equip = 0;
       kwh_htg = 0;
       kwh_clg = 0;
       kwh_twr = 0;
       kwh_aux = 0;
       kwh_vent = 0;
       kwh_venthtg = 0;
       kwh_ventclg = 0;
       kwh_refg = 0;
       kwh_hpsup = 0;
       kwh_ext = 0;
       kwh_shw = 0;
       thm_tot = 0;
       thm_equip = 0;
       thm_htg = 0;
       thm_shw = 0;
       kwpp_tot = 0;
       kwpp_ltg = 0;
       kwpp_equip = 0;
       FOR WtdResRec IN SELECT * FROM tech_results
        WHERE  tech_results.techrefid = TechResRec.techrefid
        and  tech_results.simqual  = TechResRec.simqual
        and  tech_results.bldgtype = TechResRec.bldgtype
        and  tech_results.bldgvint = TechResRec.bldgvint
        and  tech_results.bldgloc  = TechResRec.bldgloc
        and  tech_results.bldghvac = TechResRec.bldghvac
        and  tech_results.tstat > 0 and tech_results.tstat < 6
        LOOP
         SELECT tstatwt INTO Wt FROM reststatwt WHERE
            reststatwt.bldgtype = WtdResRec.bldgtype AND
            reststatwt.bldgvint = WtdResRec.bldgvint AND
            reststatwt.bldgloc  = WtdResRec.bldgloc AND
            reststatwt.tstat    = WtdResRec.tstat;
         sumWt = sumWt + Wt;
         IF Wt IS NULL THEN
           RAISE EXCEPTION 'no Wt found for %',
WtdResRec.bldgtype||':'||WtdResRec.bldgvint||':'||WtdResRec.bldgloc||':'||WtdResRec.tstat;
           END IF;
         measarea    = measarea    + WtdResRec.measarea    * Wt;
         numunits    = numunits    + WtdResRec.numunits    * Wt;
         kwh_tot     = kwh_tot     + WtdResRec.kwh_tot     * Wt;
         kwh_ltg     = kwh_ltg     + WtdResRec.kwh_ltg     * Wt;
         kwh_task    = kwh_task    + WtdResRec.kwh_task    * Wt;
         kwh_equip   = kwh_equip   + WtdResRec.kwh_equip   * Wt;
         kwh_htg     = kwh_htg     + WtdResRec.kwh_htg     * Wt;
         kwh_clg     = kwh_clg     + WtdResRec.kwh_clg     * Wt;
         kwh_twr     = kwh_twr     + WtdResRec.kwh_twr     * Wt;
         kwh_aux     = kwh_aux     + WtdResRec.kwh_aux     * Wt;
         kwh_vent    = kwh_vent    + WtdResRec.kwh_vent    * Wt;
         kwh_venthtg = kwh_venthtg + WtdResRec.kwh_venthtg * Wt;
         kwh_ventclg = kwh_ventclg + WtdResRec.kwh_ventclg * Wt;
         kwh_refg    = kwh_refg    + WtdResRec.kwh_refg    * Wt;
         kwh_hpsup   = kwh_hpsup   + WtdResRec.kwh_hpsup   * Wt;
         kwh_shw     = kwh_shw     + WtdResRec.kwh_shw     * Wt;
         kwh_ext     = kwh_ext     + WtdResRec.kwh_ext     * Wt;
         thm_tot     = thm_tot     + WtdResRec.thm_tot     * Wt;
         thm_equip   = thm_equip   + WtdResRec.thm_equip   * Wt;
         thm_htg     = thm_htg     + WtdResRec.thm_htg     * Wt;
         thm_shw     = thm_shw     + WtdResRec.thm_shw     * Wt;
         kwpp_tot    = kwpp_tot    + WtdResRec.kwpp_tot    * Wt;
         kwpp_ltg    = kwpp_ltg    + WtdResRec.kwpp_ltg    * Wt;
         kwpp_equip  = kwpp_equip  + WtdResRec.kwpp_equip  * Wt;
        END LOOP;
      INSERT INTO tech_results_wtd VALUES

(DEFAULT,TechResRec.techrefid,TechResRec.simqual,TechResRec.bldgtype,TechResRec.bldgvint,TechResRec.bldgloc,Tech
ResRec.bldghvac,-1,TechResRec.normunit,
          round((numunits   /sumWt)::NUMERIC(15,3),2),
          round((measarea   /sumWt)::NUMERIC(15,1),2),
          round((kwh_tot    /sumWt)::NUMERIC(15,1),1),
          round((kwh_ltg    /sumWt)::NUMERIC(15,1),1),
          round((kwh_task   /sumWt)::NUMERIC(15,1),1),
          round((kwh_equip  /sumWt)::NUMERIC(15,1),1),
          round((kwh_htg    /sumWt)::NUMERIC(15,1),1),
          round((kwh_clg    /sumWt)::NUMERIC(15,1),1),
          round((kwh_twr    /sumWt)::NUMERIC(15,1),1),
          round((kwh_aux    /sumWt)::NUMERIC(15,1),1),
          round((kwh_vent   /sumWt)::NUMERIC(15,1),1),
          round((kwh_venthtg/sumWt)::NUMERIC(15,1),1),
          round((kwh_ventclg/sumWt)::NUMERIC(15,1),1),
          round((kwh_refg   /sumWt)::NUMERIC(15,1),1),
          round((kwh_hpsup  /sumWt)::NUMERIC(15,1),1),
          round((kwh_shw    /sumWt)::NUMERIC(15,1),1),
          round((kwh_ext    /sumWt)::NUMERIC(15,1),1),
          round((thm_tot    /sumWt)::NUMERIC(15,2),2),
          round((thm_equip  /sumWt)::NUMERIC(15,2),2),
          round((thm_htg    /sumWt)::NUMERIC(15,2),2),
          round((thm_shw    /sumWt)::NUMERIC(15,2),2),
          round((kwpp_tot   /sumWt)::NUMERIC(15,6),3),
          round((kwpp_ltg   /sumWt)::NUMERIC(15,6),3),
          round((kwpp_equip /sumWt)::NUMERIC(15,6),3)
          );
    ELSE
     -- Add record into wtd table indicating that weighted values could not be calculated
    INSERT INTO tech_results_wtd VALUES
(DEFAULT,TechResRec.techrefid,TechResRec.simqual,TechResRec.bldgtype,TechResRec.bldgvint,TechResRec.bldgloc,TechResRec.bldghvac,-9,'num_tstats',NumTstats);
  END IF;
END LOOP;
 RETURN TotProc;
END;
```

3.2 Function to calculate Measure impacts

```
-- -- Function to determine the energy impacts for a measure run entry using WEIGHTED results data
--
-- the measure_id (as it exists in the measure_runs table) is passed to this function
-- results are first deleted (from meas_impacts_wtd) for the measure_id
-- missing data are reported in the missing_sim table; which should be emptied to see only latest missing data
-- impacts are rounded to three significant figures and are inserted into the meas_impacts_wtd table
DECLARE
    MeasRunsRec "support"."measure_matrix_ip"%ROWTYPE;
    MsrResRec "simresults"."tech_results_wtd"%ROWTYPE;
    StdResRec "simresults"."tech_results_wtd"%ROWTYPE;
    PreResRec "simresults"."tech_results_wtd"%ROWTYPE;
    IPResRec  "simresults"."tech_results_wtd"%ROWTYPE;
    MeasImpRec  "simresults"."meas_impacts_wtd"%ROWTYPE;
    NumRecs INTEGER;
    TotProc INTEGER;
    numunits FLOAT;
    apre_kwh_tot FLOAT;
    apre_kwh_ltg FLOAT;
    apre_kwh_task FLOAT;
    apre_kwh_equip FLOAT;
    apre_kwh_htg FLOAT;
    apre_kwh_clg FLOAT;
    apre_kwh_twr FLOAT;
    apre_kwh_aux FLOAT;
    apre_kwh_vent FLOAT;
    apre_kwh_venthtg FLOAT;
    apre_kwh_ventclg FLOAT;
    apre_kwh_refg FLOAT;
    apre_kwh_hpsup FLOAT;
    apre_kwh_shw FLOAT;
    apre_kwh_ext FLOAT;
    apre_thm_tot FLOAT;
    apre_thm_equip FLOAT;
    apre_thm_htg FLOAT;
    apre_thm_shw FLOAT;
    apre_kwpp_tot FLOAT;
    apre_kwpp_ltg FLOAT;
    apre_kwpp_equip FLOAT;
    astd_kwh_tot FLOAT;
    astd_kwh_ltg FLOAT;
    astd_kwh_task FLOAT;
    astd_kwh_equip FLOAT;
    astd_kwh_htg FLOAT;
    astd_kwh_clg FLOAT;
    astd_kwh_twr FLOAT;
    astd_kwh_aux FLOAT;
    astd_kwh_vent FLOAT;
    astd_kwh_venthtg FLOAT;
    astd_kwh_ventclg FLOAT;
    astd_kwh_refg FLOAT;
    astd_kwh_hpsup FLOAT;
    astd_kwh_shw FLOAT;
    astd_kwh_ext FLOAT;
    astd_thm_tot FLOAT;
    astd_thm_equip FLOAT;
    astd_thm_htg FLOAT;
    astd_thm_shw FLOAT;
    astd_kwpp_tot FLOAT;
    astd_kwpp_ltg FLOAT;
    astd_kwpp_equip FLOAT;
    base_techid TEXT;

BEGIN
 TotProc = 0;
 -- Delete any existing result for the qualifying records
 --DELETE FROM simresults.meas_impacts_wtd WHERE measure_id = $1;
 --DELETE FROM simresults.missing_sim WHERE measure_id = $1;
 -- Loop thru every intended applicability of the measure:
 FOR MeasRunsRec IN SELECT * FROM support.measure_matrix_ip WHERE tstat = 1
  AND "MeasureID" = $1
  LOOP
```

```
TotProc = TotProc + 1;
-- clear previous results
 apre_kwh_tot = 0;
 apre_kwh_ltg = 0;
 apre_kwh_task = 0;
 apre_kwh_equip = 0;
 apre_kwh_htg = 0;
 apre_kwh_clg = 0;
 apre_kwh_twr = 0;
 apre_kwh_aux = 0;
 apre_kwh_vent = 0;
 apre_kwh_venthtg = 0;
 apre_kwh_ventclg = 0;
 apre_kwh_refg = 0;
 apre_kwh_hpsup = 0;
 apre_kwh_shw = 0;
 apre_kwh_ext = 0;
 apre_thm_tot = 0;
 apre_thm_equip = 0;
 apre_thm_htg = 0;
 apre_thm_shw = 0;
 apre_kwpp_tot = 0;
 apre_kwpp_ltg = 0;
 apre_kwpp_equip = 0;
 astd_kwh_tot = 0;
 astd_kwh_ltg = 0;
 astd_kwh_task = 0;
 astd_kwh_equip = 0;
 astd_kwh_htg = 0;
 astd_kwh_clg = 0;
 astd_kwh_twr = 0;
 astd_kwh_aux = 0;
 astd_kwh_vent = 0;
 astd_kwh_venthtg = 0;
 astd_kwh_ventclg = 0;
 astd_kwh_refg = 0;
 astd_kwh_hpsup = 0;
 astd_kwh_shw = 0;
 astd_kwh_ext = 0;
 astd_thm_tot = 0;
 astd_thm_equip = 0;
 astd_thm_htg = 0;
 astd_thm_shw = 0;
 astd_kwpp_tot = 0;
 astd_kwpp_ltg = 0;
 astd_kwpp_equip = 0;
 base_techid = 'NotSet';
-- Get the Measure tech results:
SELECT * INTO MsrResRec from simresults.tech_results_wtd
```

```
     WHERE  techrefid = MeasRunsRec."MsrTechID"
       and  simqual   = MeasRunsRec."MsrSimQual"
       and  bldgtype  = MeasRunsRec.bldgtype
       and  bldgvint  = MeasRunsRec.bldgvint
       and  bldgloc   = MeasRunsRec.bldgloc
       and  bldghvac  = MeasRunsRec.bldghvac
       and  tstat     = -1;
   IF NOT FOUND THEN
    INSERT INTO simresults.missing_sim VALUES
(MeasRunsRec."MeasureID",MeasRunsRec."MsrTechID",MeasRunsRec."MsrSimQual",'Msr',MeasRunsRec.bldgtype,MeasRunsRec
.bldgvint,MeasRunsRec.bldgloc,MeasRunsRec.bldghvac,-1);
      -- RAISE NOTICE 'no tech results found for %', MeasRunsRec."MeasureID"||':'||MeasRunsRec."MsrTechID";
    CONTINUE;
   END IF;
    -- Get the Pre-Existing tech results:
   IF MeasRunsRec."PreTechID" IS NOT NULL THEN
    base_techid = MeasRunsRec."PreTechID";
    IF MeasRunsRec."PreTechID" = 'IP' THEN
     -- Get pre-existing results from Initialized Prototype results:
     SELECT * INTO PreResRec from simresults.tech_results_wtd
      WHERE  techrefid = 'IP'
        and  simqual   = 'None'
        and  bldgtype  = MeasRunsRec.bldgtype
        and  bldgvint  = MeasRunsRec.bldgvint
        and  bldgloc   = MeasRunsRec.bldgloc
        and  bldghvac  = MeasRunsRec.bldghvac
        and  tstat     = -1;
      IF NOT FOUND THEN
       INSERT INTO simresults.missing_sim VALUES
(MeasRunsRec."MeasureID",'IP','None','Pre',MeasRunsRec.bldgtype,MeasRunsRec.bldgvint,MeasRunsRec.bldgloc,MeasRun
sRec.bldghvac,-1);
         --RAISE EXCEPTION 'no Initialized Prototype results found for %',
MeasRunsRec.techrefid||':'||MeasRunsRec.bldgtype;
       CONTINUE;
      END IF;
    ELSE
     -- Get pre-existing results from Tech Results:
     SELECT * INTO PreResRec from simresults.tech_results_wtd
      WHERE  techrefid = MeasRunsRec."PreTechID"
        and  simqual   = MeasRunsRec."PreSimQual"
        and  bldgtype  = MeasRunsRec.bldgtype
        and  bldgvint  = MeasRunsRec.bldgvint
        and  bldgloc   = MeasRunsRec.bldgloc
        and  bldghvac  = MeasRunsRec.bldghvac
        and  tstat     = -1;
      IF NOT FOUND THEN
       INSERT INTO simresults.missing_sim VALUES
(MeasRunsRec."MeasureID",MeasRunsRec."PreTechID",MeasRunsRec."PreSimQual",'Pre',MeasRunsRec.bldgtype,MeasRunsRec
.bldgvint,MeasRunsRec.bldgloc,MeasRunsRec.bldghvac,-1);
         -- RAISE EXCEPTION 'no pre-existing tech results found for %',
MeasRunsRec.measure_id||':'||MeasRunsRec.pre_refid;
       CONTINUE;
      END IF;
     END IF; -- of IF PreTechID = 'IP'
     numunits = MsrResRec.numunits;
     apre_kwh_tot     = (PreResRec.kwh_tot     - MsrResRec.kwh_tot)/numunits;
     apre_kwh_ltg     = (PreResRec.kwh_ltg     - MsrResRec.kwh_ltg)/numunits;
     apre_kwh_task    = (PreResRec.kwh_task    - MsrResRec.kwh_task)/numunits;
     apre_kwh_equip   = (PreResRec.kwh_equip   - MsrResRec.kwh_equip)/numunits;
     apre_kwh_htg     = (PreResRec.kwh_htg     - MsrResRec.kwh_htg)/numunits;
     apre_kwh_clg     = (PreResRec.kwh_clg     - MsrResRec.kwh_clg)/numunits;
     apre_kwh_twr     = (PreResRec.kwh_twr     - MsrResRec.kwh_twr)/numunits;
     apre_kwh_aux     = (PreResRec.kwh_aux     - MsrResRec.kwh_aux)/numunits;
     apre_kwh_vent    = (PreResRec.kwh_vent    - MsrResRec.kwh_vent)/numunits;
     apre_kwh_venthtg = (PreResRec.kwh_venthtg - MsrResRec.kwh_venthtg)/numunits;
     apre_kwh_ventclg = (PreResRec.kwh_ventclg - MsrResRec.kwh_ventclg)/numunits;
     apre_kwh_refg    = (PreResRec.kwh_refg    - MsrResRec.kwh_refg)/numunits;
     apre_kwh_hpsup   = (PreResRec.kwh_hpsup   - MsrResRec.kwh_hpsup)/numunits;
     apre_kwh_shw     = (PreResRec.kwh_shw     - MsrResRec.kwh_shw)/numunits;
     apre_kwh_ext     = (PreResRec.kwh_ext     - MsrResRec.kwh_ext)/numunits;
     apre_thm_tot     = (PreResRec.thm_tot     - MsrResRec.thm_tot)/numunits;
     apre_thm_equip   = (PreResRec.thm_equip   - MsrResRec.thm_equip)/numunits;
     apre_thm_htg     = (PreResRec.thm_htg     - MsrResRec.thm_htg)/numunits;
     apre_thm_shw     = (PreResRec.thm_shw     - MsrResRec.thm_shw)/numunits;
     apre_kwpp_tot    = (PreResRec.kwpp_tot    - MsrResRec.kwpp_tot)/numunits;
     apre_kwpp_ltg    = (PreResRec.kwpp_ltg    - MsrResRec.kwpp_ltg)/numunits;
     apre_kwpp_equip  = (PreResRec.kwpp_equip  - MsrResRec.kwpp_equip)/numunits;
   END IF; -- of IF PreTechID NOT NULL

   IF MeasRunsRec."StdTechID" IS NOT NULL THEN
    -- if the base_techid has not been set yet, then this is the base_techid (no pre-existing case for this
measure)
    If base_techid = 'NotSet' then base_techid = MeasRunsRec."StdTechID"; End IF;
    IF MeasRunsRec."StdTechID" = 'IP' THEN
     -- Get standard results from Initialized Prototype results:
     SELECT * INTO StdResRec from simresults.tech_results_wtd
      WHERE  techrefid = 'IP'
        and  simqual   = 'None'
        and  bldgtype  = MeasRunsRec.bldgtype
        and  bldgvint  = MeasRunsRec.bldgvint
        and  bldgloc   = MeasRunsRec.bldgloc
        and  bldghvac  = MeasRunsRec.bldghvac
        and  tstat     = -1;
      IF NOT FOUND THEN
       INSERT INTO simresults.missing_sim VALUES
(MeasRunsRec."MeasureID",'IP','None','Std',MeasRunsRec.bldgtype,MeasRunsRec.bldgvint,MeasRunsRec.bldgloc,MeasRun
sRec.bldghvac,-1);
         --RAISE EXCEPTION 'no Initialized Prototype results found for %',
MeasRunsRec.measure_id||':'||MeasRunsRec.pre_refid;
       CONTINUE;
      END IF;
    ELSE
    -- Get Code/Standard results from Tech Results (I don't think it's ever from IP results):
     SELECT * INTO StdResRec from simresults.tech_results_wtd
      WHERE  techrefid = MeasRunsRec."StdTechID"
        and  simqual   = MeasRunsRec."StdSimQual"
        and  bldgtype  = MeasRunsRec.bldgtype
        and  bldgvint  = MeasRunsRec.bldgvint
        and  bldgloc   = MeasRunsRec.bldgloc
        and  bldghvac  = MeasRunsRec.bldghvac
        and  tstat     = -1;
     IF NOT FOUND THEN
      INSERT INTO simresults.missing_sim VALUES
(MeasRunsRec."MeasureID",MeasRunsRec."StdTechID",MeasRunsRec."StdSimQual",'Std',MeasRunsRec.bldgtype,MeasRunsRec
.bldgvint,MeasRunsRec.bldgloc,MeasRunsRec.bldghvac,-1);
      --RAISE EXCEPTION 'no code/standard results found for %', MeasRunsRec.||':'||MeasRunsRec.std_refid;
      CONTINUE;
     END IF;
    END IF; -- of IF StdTechID = 'IP'
     numunits = MsrResRec.numunits;
     astd_kwh_tot     = (StdResRec.kwh_tot     - MsrResRec.kwh_tot)/numunits;
     astd_kwh_ltg     = (StdResRec.kwh_ltg     - MsrResRec.kwh_ltg)/numunits;
     astd_kwh_task    = (StdResRec.kwh_task    - MsrResRec.kwh_task)/numunits;
     astd_kwh_equip   = (StdResRec.kwh_equip   - MsrResRec.kwh_equip)/numunits;
     astd_kwh_htg     = (StdResRec.kwh_htg     - MsrResRec.kwh_htg)/numunits;
     astd_kwh_clg     = (StdResRec.kwh_clg     - MsrResRec.kwh_clg)/numunits;
     astd_kwh_twr     = (StdResRec.kwh_twr     - MsrResRec.kwh_twr)/numunits;
     astd_kwh_aux     = (StdResRec.kwh_aux     - MsrResRec.kwh_aux)/numunits;
     astd_kwh_vent    = (StdResRec.kwh_vent    - MsrResRec.kwh_vent)/numunits;
     astd_kwh_venthtg = (StdResRec.kwh_venthtg - MsrResRec.kwh_venthtg)/numunits;
     astd_kwh_ventclg = (StdResRec.kwh_ventclg - MsrResRec.kwh_ventclg)/numunits;
     astd_kwh_refg    = (StdResRec.kwh_refg    - MsrResRec.kwh_refg)/numunits;
     astd_kwh_hpsup   = (StdResRec.kwh_hpsup   - MsrResRec.kwh_hpsup)/numunits;
     astd_kwh_shw     = (StdResRec.kwh_shw     - MsrResRec.kwh_shw)/numunits;
     astd_kwh_ext     = (StdResRec.kwh_ext     - MsrResRec.kwh_ext)/numunits;
     astd_thm_tot     = (StdResRec.thm_tot     - MsrResRec.thm_tot)/numunits;
     astd_thm_equip   = (StdResRec.thm_equip   - MsrResRec.thm_equip)/numunits;
     astd_thm_htg     = (StdResRec.thm_htg     - MsrResRec.thm_htg)/numunits;
     astd_thm_shw     = (StdResRec.thm_shw     - MsrResRec.thm_shw)/numunits;
     astd_kwpp_tot    = (StdResRec.kwpp_tot    - MsrResRec.kwpp_tot)/numunits;
     astd_kwpp_ltg    = (StdResRec.kwpp_ltg    - MsrResRec.kwpp_ltg)/numunits;
     astd_kwpp_equip  = (StdResRec.kwpp_equip  - MsrResRec.kwpp_equip)/numunits;
   END IF;

   -- Copy the impacts into the Measure Impacts table
   INSERT INTO simresults.meas_impacts_wtd VALUES
         (DEFAULT,
          MeasRunsRec."MeasureID",
          MeasRunsRec.bldgtype,
          MeasRunsRec.bldgvint,
          MeasRunsRec.bldgloc,
          MeasRunsRec.bldghvac,
          -1,
          MsrResRec.normunit,
          MsrResRec.numunits,
          MsrResRec.measarea,
          -- above pre-existing impacts:
          case when apre_kwh_tot     = 0 then 0 else round(apre_kwh_tot::NUMERIC(15,3)    ,(2-
floor(log(abs(apre_kwh_tot))))::INT) end,
          case when apre_kwh_ltg     = 0 then 0 else round(apre_kwh_ltg::NUMERIC(15,3)    ,(2-
floor(log(abs(apre_kwh_ltg))))::INT) end,
          case when apre_kwh_task    = 0 then 0 else round(apre_kwh_task::NUMERIC(15,3)   ,(2-
floor(log(abs(apre_kwh_task))))::INT) end,
          case when apre_kwh_equip   = 0 then 0 else round(apre_kwh_equip::NUMERIC(15,3)  ,(2-
floor(log(abs(apre_kwh_equip))))::INT) end,
          case when apre_kwh_htg     = 0 then 0 else round(apre_kwh_htg::NUMERIC(15,3)    ,(2-
floor(log(abs(apre_kwh_htg))))::INT) end,
          case when apre_kwh_clg     = 0 then 0 else round(apre_kwh_clg::NUMERIC(15,3)    ,(2-
floor(log(abs(apre_kwh_clg))))::INT) end,
          case when apre_kwh_twr     = 0 then 0 else round(apre_kwh_twr::NUMERIC(15,3)    ,(2-
floor(log(abs(apre_kwh_twr))))::INT) end,
          case when apre_kwh_aux     = 0 then 0 else round(apre_kwh_aux::NUMERIC(15,3)    ,(2-
floor(log(abs(apre_kwh_aux))))::INT) end,
          case when apre_kwh_vent    = 0 then 0 else round(apre_kwh_vent::NUMERIC(15,3)   ,(2-
floor(log(abs(apre_kwh_vent))))::INT) end,
          case when apre_kwh_venthtg = 0 then 0 else round(apre_kwh_venthtg::NUMERIC(15,3),(2-
floor(log(abs(apre_kwh_venthtg))))::INT) end,
          case when apre_kwh_ventclg = 0 then 0 else round(apre_kwh_ventclg::NUMERIC(15,3),(2-
floor(log(abs(apre_kwh_ventclg))))::INT) end,
          case when apre_kwh_refg    = 0 then 0 else round(apre_kwh_refg::NUMERIC(15,3)   ,(2-
floor(log(abs(apre_kwh_refg))))::INT) end,
          case when apre_kwh_hpsup   = 0 then 0 else round(apre_kwh_hpsup::NUMERIC(15,3)  ,(2-
floor(log(abs(apre_kwh_hpsup))))::INT) end,
          case when apre_kwh_shw     = 0 then 0 else round(apre_kwh_shw::NUMERIC(15,3)    ,(2-
floor(log(abs(apre_kwh_shw))))::INT) end,
          case when apre_kwh_ext     = 0 then 0 else round(apre_kwh_ext::NUMERIC(15,3)    ,(2-
floor(log(abs(apre_kwh_ext))))::INT) end,
          case when apre_thm_tot     = 0 then 0 else round(apre_thm_tot::NUMERIC(15,3)    ,(2-
floor(log(abs(apre_thm_tot))))::INT) end,
          case when apre_thm_equip   = 0 then 0 else round(apre_thm_equip::NUMERIC(15,3)  ,(2-
floor(log(abs(apre_thm_equip))))::INT) end,
          case when apre_thm_htg     = 0 then 0 else round(apre_thm_htg::NUMERIC(15,3)    ,(2-
floor(log(abs(apre_thm_htg))))::INT) end,
          case when apre_thm_shw     = 0 then 0 else round(apre_thm_shw::NUMERIC(15,3)    ,(2-
floor(log(abs(apre_thm_shw))))::INT) end,
          case when apre_kwpp_tot    = 0 then 0 else round(apre_kwpp_tot::NUMERIC(15,6)   ,(2-
floor(log(abs(apre_kwpp_tot))))::INT) end,
          case when apre_kwpp_ltg    = 0 then 0 else round(apre_kwpp_ltg::NUMERIC(15,6)   ,(2-
floor(log(abs(apre_kwpp_ltg))))::INT) end,
          case when apre_kwpp_equip  = 0 then 0 else round(apre_kwpp_equip::NUMERIC(15,6) ,(2-
floor(log(abs(apre_kwpp_equip))))::INT) end,
          -- above standard/code impacts:
          case when astd_kwh_tot     = 0 then 0 else round(astd_kwh_tot::NUMERIC(15,3)    ,(2-
floor(log(abs(astd_kwh_tot))))::INT) end,
          case when astd_kwh_ltg     = 0 then 0 else round(astd_kwh_ltg::NUMERIC(15,3)    ,(2-
floor(log(abs(astd_kwh_ltg))))::INT) end,
          case when astd_kwh_task    = 0 then 0 else round(astd_kwh_task::NUMERIC(15,3)   ,(2-
floor(log(abs(astd_kwh_task))))::INT) end,
          case when astd_kwh_equip   = 0 then 0 else round(astd_kwh_equip::NUMERIC(15,3)  ,(2-
floor(log(abs(astd_kwh_equip))))::INT) end,
          case when astd_kwh_htg     = 0 then 0 else round(astd_kwh_htg::NUMERIC(15,3)    ,(2-
floor(log(abs(astd_kwh_htg))))::INT) end,
          case when astd_kwh_clg     = 0 then 0 else round(astd_kwh_clg::NUMERIC(15,3)    ,(2-
floor(log(abs(astd_kwh_clg))))::INT) end,
          case when astd_kwh_twr     = 0 then 0 else round(astd_kwh_twr::NUMERIC(15,3)    ,(2-
floor(log(abs(astd_kwh_twr))))::INT) end,
          case when astd_kwh_aux     = 0 then 0 else round(astd_kwh_aux::NUMERIC(15,3)    ,(2-
floor(log(abs(astd_kwh_aux))))::INT) end,
          case when astd_kwh_vent    = 0 then 0 else round(astd_kwh_vent::NUMERIC(15,3)   ,(2-
floor(log(abs(astd_kwh_vent))))::INT) end,
          case when astd_kwh_venthtg = 0 then 0 else round(astd_kwh_venthtg::NUMERIC(15,3),(2-
floor(log(abs(astd_kwh_venthtg))))::INT) end,
          case when astd_kwh_ventclg = 0 then 0 else round(astd_kwh_ventclg::NUMERIC(15,3),(2-
floor(log(abs(astd_kwh_ventclg))))::INT) end,
          case when astd_kwh_refg    = 0 then 0 else round(astd_kwh_refg::NUMERIC(15,3)   ,(2-
floor(log(abs(astd_kwh_refg))))::INT) end,
          case when astd_kwh_hpsup   = 0 then 0 else round(astd_kwh_hpsup::NUMERIC(15,3)  ,(2-
floor(log(abs(astd_kwh_hpsup))))::INT) end,
          case when astd_kwh_shw     = 0 then 0 else round(astd_kwh_shw::NUMERIC(15,3)    ,(2-
floor(log(abs(astd_kwh_shw))))::INT) end,
          case when astd_kwh_ext     = 0 then 0 else round(astd_kwh_ext::NUMERIC(15,3)    ,(2-
floor(log(abs(astd_kwh_ext))))::INT) end,
          case when astd_thm_tot     = 0 then 0 else round(astd_thm_tot::NUMERIC(15,3)    ,(2-
floor(log(abs(astd_thm_tot))))::INT) end,
          case when astd_thm_equip   = 0 then 0 else round(astd_thm_equip::NUMERIC(15,3)  ,(2-
floor(log(abs(astd_thm_equip))))::INT) end,
          case when astd_thm_htg     = 0 then 0 else round(astd_thm_htg::NUMERIC(15,3)    ,(2-
floor(log(abs(astd_thm_htg))))::INT) end,
          case when astd_thm_shw     = 0 then 0 else round(astd_thm_shw::NUMERIC(15,3)    ,(2-
floor(log(abs(astd_thm_shw))))::INT) end,
          case when astd_kwpp_tot    = 0 then 0 else round(astd_kwpp_tot::NUMERIC(15,6)   ,(2-
floor(log(abs(astd_kwpp_tot))))::INT) end,
          case when astd_kwpp_ltg    = 0 then 0 else round(astd_kwpp_ltg::NUMERIC(15,6)   ,(2-
floor(log(abs(astd_kwpp_ltg))))::INT) end,
          case when astd_kwpp_equip  = 0 then 0 else round(astd_kwpp_equip::NUMERIC(15,6) ,(2-
floor(log(abs(astd_kwpp_equip))))::INT) end,
          base_techid
          );
  END LOOP;
 RETURN TotProc;
END;
```