**Modelos de datos de los Analizadores eléctricos**

**Objetos principales**

|  |  |
| --- | --- |
| **Objetos JSON** | **Descripción** |
| ***ppl:energy***  *Read-only* | *Objeto (de sólo lectura) que incluye toda la información asociada a un analizador eléctrico. Tanto las variables del estado actual <****ppl:energy:stat****>, como los parámetros de configuración <****ppl:energy:cfg****>.*  *ppl:energy = {*  *"stat": ppl:energy:stat*  *"cfg": ppl:energy:cfg,*  *}* |
| ***ppl:energy:stat***  *Read-only* | *Objeto (de sólo lectura) que incluye todas las variables de estado de un analizador eléctrico. Incluye los totales de energía y las medidas instantáneas entre otros:*   * *Totales de energía:* ***ppl:energy:stat:totals*** * *Medidas instantáneas:* ***ppl:energy:stat:measure*** |
| ***ppl:energy:cfg***  *Read-Write* | *Objeto (de lectura/escritura) que incluye todos los parámetros de configuración de un analizador eléctrico. Entre ellos, se incluyen los rangos para la generación de alarmas, datos de calibración de la medida, etc.:*   * *Rangos para generación de alarmas:* ***ppl:energy:cfg:minmax*** * *Parámetros de calibración de medida:* ***ppl:energy:cfg:calib*** |

**Identificación de versiones**

Todos los modelos de datos pueden sufrir cambios y representar diferentes versiones de un objeto base. Para identificar la versión de un objeto, se incluirá el campo <version> dentro del objeto JSON y se definirá un nuevo tipo de objeto a partir del objeto base, con la siguiente nomenclatura: **objeto:base:version**

La siguiente tabla muestra algunos ejemplos:

|  |  |
| --- | --- |
| **Objetos JSON** | **Descripción** |
| ***ppl:energy:vers\_descr*** | *Objeto que representa la versión <vers\_descr> del objeto <ppl:energy>* |
| ***ppl:energy:stat:vers\_descr*** | *Objeto que representa la versión <vers\_descr> del objeto <ppl:energy:stat>* |
| ***ppl:energy:cfg:vers\_descr*** | *Objeto que representa la versión <vers\_descr> del objeto <ppl:energy:cfg>* |

**Descripción de los diferentes objetos**

|  |  |
| --- | --- |
| ***ppl:energy:stat*** | |
| *{*  *"version": "vers\_descr", [OPCIONAL]*  *"flags": 16777224,*  *"energyValues": ppl:energy:stat:totals, // versión dependiente del driver de medida utilizado*  *"measureValues": ppl:energy:stat:measure // versión dependiente del driver de medida utilizado*  *}* | |
|  | |
| *version : string*  *[OPCIONAL]*  *Read-only* | *Descripción*  *Es un campo opcional, que identifica la versión de este objeto. En caso de no estar presente, se tomará la versión por defecto.*  *La versión por defecto tendrá la siguiente configuración:*   * *energyValues :* ***ppl:energy:stat:totals*** *[versión por defecto]* * *measureValues :* ***ppl:energy:stat:measure*** *[versión por defecto]* |
| *flags : u32*  *Read-only* | *Descripción*  *Conjunto de flags, que indican los eventos y/o alarmas activos en el instante actual*  *Valores*  *AMNoEvents* = 0, //!< Flag para indicar que no hay eventos  *AMVoltageOverLimitEvt* = (1 << 0), //!< Evento al superar el límite superior (voltaje)  *AMVoltageBelowLimitEvt* = (1 << 1), //!< Evento al superar el límite inferior (voltaje)  *AMVoltageInRangeEvt* = (1 << 2), //!< Evento al volver a entrar en rango (voltaje)  *AMCurrentOverLimitEvt* = (1 << 3), //!< Evento al superar el límite superior (corriente)  *AMCurrentBelowLimitEvt* = (1 << 4), //!< Evento al superar el límite inferior (corriente)  *AMCurrentInRangeEvt* = (1 << 5), //!< Evento al volver a entrar en rango (corriente)  *AMPhaseOverLimitEvt* = (1 << 6), //!< Evento al superar el límite superior (fase)  *AMPhaseBelowLimitEvt* = (1 << 7), //!< Evento al superar el límite inferior (fase)  *AMPhaseInRangeEvt* = (1 << 8), //!< Evento al volver a entrar en rango (fase)  *AMPFactorOverLimitEvt* = (1 << 9), //!< Evento al superar el límite superior (factor de potencia)  *AMPFactorBelowLimitEvt* = (1 << 10), //!< Evento al superar el límite inferior (factor de potencia)  *AMPFactorInRangeEvt* = (1 << 11), //!< Evento al volver a entrar en rango (factor de potencia)  *AMCosPhyOverLimitEvt* = (1 << 12), //!< Evento al superar el límite superior (coseno phy)  *AMCosPhyBelowLimitEvt* = (1 << 13), //!< Evento al superar el límite inferior (coseno phy)  *AMCosPhyInRangeEvt* = (1 << 14), //!< Evento al volver a entrar en rango (coseno phy)  *AMActPowOverLimitEvt* = (1 << 15), //!< Evento al superar el límite superior (potencia activa)  *AMActPowBelowLimitEvt* = (1 << 16), //!< Evento al superar el límite inferior (potencia activa)  *AMActPowInRangeEvt* = (1 << 17), //!< Evento al volver a entrar en rango (potencia activa)  *AMReactPowOverLimitEvt* = (1 << 18), //!< Evento al superar el límite superior (potencia reactiva)  *AMReactPowBelowLimitEvt* = (1 << 19), //!< Evento al superar el límite inferior (potencia reactiva)  *AMReactPowInRangeEvt* = (1 << 20), //!< Evento al volver a entrar en rango (potencia reactiva)  *AMFrequencyOverLimitEvt* = (1 << 21), //!< Evento al superar el límite superior (frecuencia)  *AMFrequencyBelowLimitEvt* = (1 << 22), //!< Evento al superar el límite inferior (frecuencia)  *AMFrequencyInRangeEvt* = (1 << 23), //!< Evento al volver a entrar en rango (frecuencia)  *AMThdAOverLimitEvt* = (1 << 24), //!< Evento al superar el límite superior (THD-A)  *AMThdABelowLimitEvt* = (1 << 25), //!< Evento al superar el límite inferior (THD-A)  *AMThdAInRangeEvt* = (1 << 26), //!< Evento al volver a entrar en rango (THD-A)  *AMThdVOverLimitEvt* = (1 << 27), //!< Evento al superar el límite superior (THD-V)  *AMThdVBelowLimitEvt* = (1 << 28), //!< Evento al superar el límite inferior (THD-V)  *AMThdVInRangeEvt* = (1 << 29), //!< Evento al volver a entrar en rango (THD-V)  *AMInstantMeasureEvt* = (1 << 30), //!< Evento al realizar una medida  *AMInvalidEvt* = (1 << 31) |
| *energyValues : ppl:energy:stat:totals*  *Read-only* | *Descripción*  *Objeto que incluye las medidas totales de energía (activa, reactiva, etc...). Puesto que estos datos son dependientes del tipo de driver de medida, existirá un objeto por defecto y diferentes implementaciones dependiendo del driver utilizado:*  ------------------------------------  ***[VERSION POR DEFECTO]***  *Objeto:* ***ppl:energy:stat:totals***  *ppl:energy:stat:totals = {*  *"active": double (KW), // Totales de energía activa (KW)*  *"reactive": double (KW) // Totales de energía reactiva (KW)*  *}*  ------------------------------------  ***Versión con energía importada-exportada***  *Objeto:* ***ppl:energy:stat:totals:impexp***  *ppl:energy:stat:totals:impexp = {*  *"version": "impexp", // Versión con datos de energía importada-exportada*  *"active": double (KW), // Totales de energía activa (KW)*  *"reactive": double (KW) // Totales de energía reactiva (KW)*  *"aImport": double (KW), // Totales de energía activa importada (KW)*  *"rImport": double (KW) // Totales de energía reactiva importada (KW)*  *"aExport": double (KW), // Totales de energía activa exportada (KW)*  *"rExport": double (KW) // Totales de energía reactiva exportada (KW)*  *}* |
| *measureValues : ppl:energy:stat:measure*  *Read-only* | *Descripción*  *Objeto que incluye los valores de medida instantánea. Puesto que estos datos son dependientes del tipo de driver, existirá un objeto por cada driver:*  ------------------------------------  ***Driver Emi10 YTL [VERSION POR DEFECTO]***  *Objeto:* ***ppl:energy:stat:measure***  *ppl:energy:stat:measure = {*  *"voltage": double (V), // Voltaje (V)*  *"current": double (A), // Corriente (A)*  *"phase": double (º), // Fase (º)*  *"pfactor": double (n/a), // Factor de potencia*  *"aPow": double (KW), // Potencia activa (KW)*  *"rPow": double (KW), // Potencia reactiva (KW)*  *"msPow": double (KVA), // Potencia media aparente (KVA)*  *"freq": double (Hz), // Frecuencia (Hz)*  *"thdA": double (n/a), // Distorsión armónica total de corriente*  *"thdV": double (n/a) // Distorsión armónica total de tensión*  *}*  ------------------------------------  ***Driver M90E26***  *Objeto:* ***ppl:energy:stat:measure:m90e26***  *ppl:energy:stat:measure:m90e26 = {*  *"version": "m90e26", // Versión para el driver M90E26*  *"voltage": double (V), // Voltaje (V)*  *"current": double (A), // Corriente (A)*  *"phase": double (º), // Fase (º)*  *"pfactor": double (n/a), // Factor de potencia*  *"aPow": double (KW), // Potencia activa (KW)*  *"rPow": double (KW), // Potencia reactiva (KW)*  *"msPow": double (KVA), // Potencia media aparente (KVA)*  *"freq": double (Hz), // Frecuencia (Hz)*  *}* |

|  |  |
| --- | --- |
| ***ppl:energy:cfg*** | |
| *{*  *"version": "vers\_descr",*  *"updFlags": 1,*  *"evtFlags": 16777216,*  *"measPeriod": 30,*  *"minmaxData": ppl:energy:cfg:minmax, // versión dependiente del driver de medida utilizado*  *"calibData": ppl:energy:cfg:calib, // versión dependiente del driver de medida utilizado*  *"verbosity": 4*  *}* | |
|  | |
| *version : string*  *[OPCIONAL]*  *Read-Write* | *Descripción*  *Es un campo opcional, que identifica la versión de este objeto. En caso de no estar presente, se tomará la versión por defecto.*  *La versión por defecto tendrá la siguiente configuración:*   * *minmaxData :* ***ppl:energy:cfg:minmax*** *[versión por defecto]* * *calibData :* ***ppl:energy:cfg:calib***  *[versión por defecto]* |
| *updFlags : u32*  *[default = EnableAMCfgUpdNotif]*  *Read-Write* | *Descripción*  *Conjunto de flags, que indican si hay que notificar una actualización de alguno de los parámetros de configuración*  *Valores*  *EnableAMCfgUpdNotif* = (1 << 0), /// notifica cambios en la configuración del objeto |
| *evtFlags : u32*  *[default = AMInstantMeasureEvt]*  *Read-Write* | *Descripción*  *Conjunto de flags, que habilitan las notificaciones de los diferentes eventos/alarmas. Para que una alarma sea notificada, su flag debe estar activado, en caso contrario, aunque la condición de alarma se produzca, no se generará ninguna notificación.*  *Valores*  *AMNoEvents* = 0, //!< Flag para indicar que no hay eventos  *AMVoltageOverLimitEvt* = (1 << 0), //!< Evento al superar el límite superior (voltaje)  *AMVoltageBelowLimitEvt* = (1 << 1), //!< Evento al superar el límite inferior (voltaje)  *AMVoltageInRangeEvt* = (1 << 2), //!< Evento al volver a entrar en rango (voltaje)  *AMCurrentOverLimitEvt* = (1 << 3), //!< Evento al superar el límite superior (corriente)  *AMCurrentBelowLimitEvt* = (1 << 4), //!< Evento al superar el límite inferior (corriente)  *AMCurrentInRangeEvt* = (1 << 5), //!< Evento al volver a entrar en rango (corriente)  *AMPhaseOverLimitEvt* = (1 << 6), //!< Evento al superar el límite superior (fase)  *AMPhaseBelowLimitEvt* = (1 << 7), //!< Evento al superar el límite inferior (fase)  *AMPhaseInRangeEvt* = (1 << 8), //!< Evento al volver a entrar en rango (fase)  *AMPFactorOverLimitEvt* = (1 << 9), //!< Evento al superar el límite superior (factor de potencia)  *AMPFactorBelowLimitEvt* = (1 << 10), //!< Evento al superar el límite inferior (factor de potencia)  *AMPFactorInRangeEvt* = (1 << 11), //!< Evento al volver a entrar en rango (factor de potencia)  *AMCosPhyOverLimitEvt* = (1 << 12), //!< Evento al superar el límite superior (coseno phy)  *AMCosPhyBelowLimitEvt* = (1 << 13), //!< Evento al superar el límite inferior (coseno phy)  *AMCosPhyInRangeEvt* = (1 << 14), //!< Evento al volver a entrar en rango (coseno phy)  *AMActPowOverLimitEvt* = (1 << 15), //!< Evento al superar el límite superior (potencia activa)  *AMActPowBelowLimitEvt* = (1 << 16), //!< Evento al superar el límite inferior (potencia activa)  *AMActPowInRangeEvt* = (1 << 17), //!< Evento al volver a entrar en rango (potencia activa)  *AMReactPowOverLimitEvt* = (1 << 18), //!< Evento al superar el límite superior (potencia reactiva)  *AMReactPowBelowLimitEvt* = (1 << 19), //!< Evento al superar el límite inferior (potencia reactiva)  *AMReactPowInRangeEvt* = (1 << 20), //!< Evento al volver a entrar en rango (potencia reactiva)  *AMFrequencyOverLimitEvt* = (1 << 21), //!< Evento al superar el límite superior (frecuencia)  *AMFrequencyBelowLimitEvt* = (1 << 22), //!< Evento al superar el límite inferior (frecuencia)  *AMFrequencyInRangeEvt* = (1 << 23), //!< Evento al volver a entrar en rango (frecuencia)  *AMThdAOverLimitEvt* = (1 << 24), //!< Evento al superar el límite superior (THD-A)  *AMThdABelowLimitEvt* = (1 << 25), //!< Evento al superar el límite inferior (THD-A)  *AMThdAInRangeEvt* = (1 << 26), //!< Evento al volver a entrar en rango (THD-A)  *AMThdVOverLimitEvt* = (1 << 27), //!< Evento al superar el límite superior (THD-V)  *AMThdVBelowLimitEvt* = (1 << 28), //!< Evento al superar el límite inferior (THD-V)  *AMThdVInRangeEvt* = (1 << 29), //!< Evento al volver a entrar en rango (THD-V)  *AMInstantMeasureEvt* = (1 << 30), //!< Evento al realizar una medida  *AMInvalidEvt* = (1 << 31) |
| *measPeriod : u32*  *[default = 900]*  *Read-Write* | *Descripción*  *Cadencia de envío de las medidas eléctricas, expresada en segundos.* |
| *minmaxData : ppl:energy:cfg:minmax*  *Read-Write* | *Descripción*  *Objeto que incluye los rangos min-max-thres de diferentes parámetros relativos a la generación de eventos y/o alarmas al salir de rango por encima (value > max), por debajo (value < min) o al volver al rango tras haber salido (value < [max-thres]) o (value > [min+thres]). Puesto que estos datos son dependientes del tipo de driver, existirá un objeto por cada uno.*  ------------------------------------  ***Driver Emi10 YTL [VERSION POR DEFECTO]***  *Objeto:* ***ppl:energy:cfg:minmax***  *ppl:energy:cfg:minmax = {*  *"voltage": {"min": double,"max": double,"thres": double}, // Voltaje (V)*  *"current": {"min": double,"max": double,"thres": double}, // Corriente (A)*  *"phase": {"min": double,"max": double,"thres": double}, // Fase (º)*  *"pfactor": {"min": double,"max": double,"thres": double}, // Factor de potencia*  *"aPow": {"min": double,"max": double,"thres": double}, // Potencia active (KW)*  *"rPow": {"min": double,"max": double,"thres": double}, // Potencia reactiva (KW)*  *"msPow": {"min": double,"max": double,"thres": double}, // Potencia media aparente (KVA)*  *"freq": {"min": double,"max": double,"thres": double}, // Frecuencia (Hz)*  *"thdA": {"min": double,"max": double,"thres": double}, // Distorsión armónica corriente*  *"thdV": {"min": double,"max": double,"thres": double} // Distorsión armónica tensión*  *}*  ------------------------------------  ***Driver M90E26***  *Objeto:* ***ppl:energy:cfg:minmax:m90e26***  *ppl:energy:cfg:minmax = {*  *"version": "m90e26",*  *"voltage": {"min": double,"max": double,"thres": double}, // Voltaje (V)*  *"current": {"min": double,"max": double,"thres": double}, // Corriente (A)*  *"phase": {"min": double,"max": double,"thres": double}, // Fase (º)*  *"pfactor": {"min": double,"max": double,"thres": double}, // Factor de potencia*  *"aPow": {"min": double,"max": double,"thres": double}, // Potencia active (KW)*  *"rPow": {"min": double,"max": double,"thres": double}, // Potencia reactiva (KW)*  *"msPow": {"min": double,"max": double,"thres": double}, // Potencia media aparente (KVA)*  *"freq": {"min": double,"max": double,"thres": double}, // Frecuencia (Hz)*  *}* |
| *calibData : ppl:energy:cfg:calib*  *Read-Write* | *Descripción*  *Objeto que incluye los valores de calibración del driver de medida. Puesto que estos datos son dependientes del tipo de driver, existirá un objeto por cada driver.*  ------------------------------------  ***Driver Emi10 YTL [VERSION POR DEFECTO]***  *Objeto:* ***ppl:energy:cfg:calib***  *En este caso, puesto que el driver no permite calibración, se adjunta un objeto vacío.*  *ppl:energy:cfg:calib = {}*  ------------------------------------  ***Driver M90E26***  *Objeto:* ***ppl:energy:cfg:calib:m90e26***  *En este caso, se corresponde con los datos de calibración para el chip M90E26.Ver sección 5.3 del datasheet* <http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-46002-SE-M90E26-Datasheet.pdf>  *ppl:energy:cfg:calib:m90e26 = {*  *"version": "m90e26", // Versión del objeto para el driver M90E26*  *"meterRegs": u16[16], // Registros de calibración del medidor. Sección 5.3.1 del datasheet.*  *"measRegs": u16[16] // Registros de calibración de la medida. Sección 5.3.2 del datasheet.*  *}* |
| *verbosity : u8*  *[default = ESP\_LOG\_WARN]*  *Read-Write* | *Descripción*  *Nivel de visualización de las trazas de depuración.*  *Valores*  *ESP\_LOG\_NONE* /\*!< No log output \*/  *ESP\_LOG\_ERROR* /\*!< Critical errors, software module can not recover on its own \*/  *ESP\_LOG\_WARN* /\*!< Error conditions from which recovery measures have been taken \*/  *ESP\_LOG\_INFO* /\*!< Information messages which describe normal flow of events \*/  *ESP\_LOG\_DEBUG* /\*!< Extra information which is not necessary for normal use (values, pointers, …). \*/  *ESP\_LOG\_VERBOSE* /\*!< Bigger chunks of information, or messages which can flood the terminal \*/ |