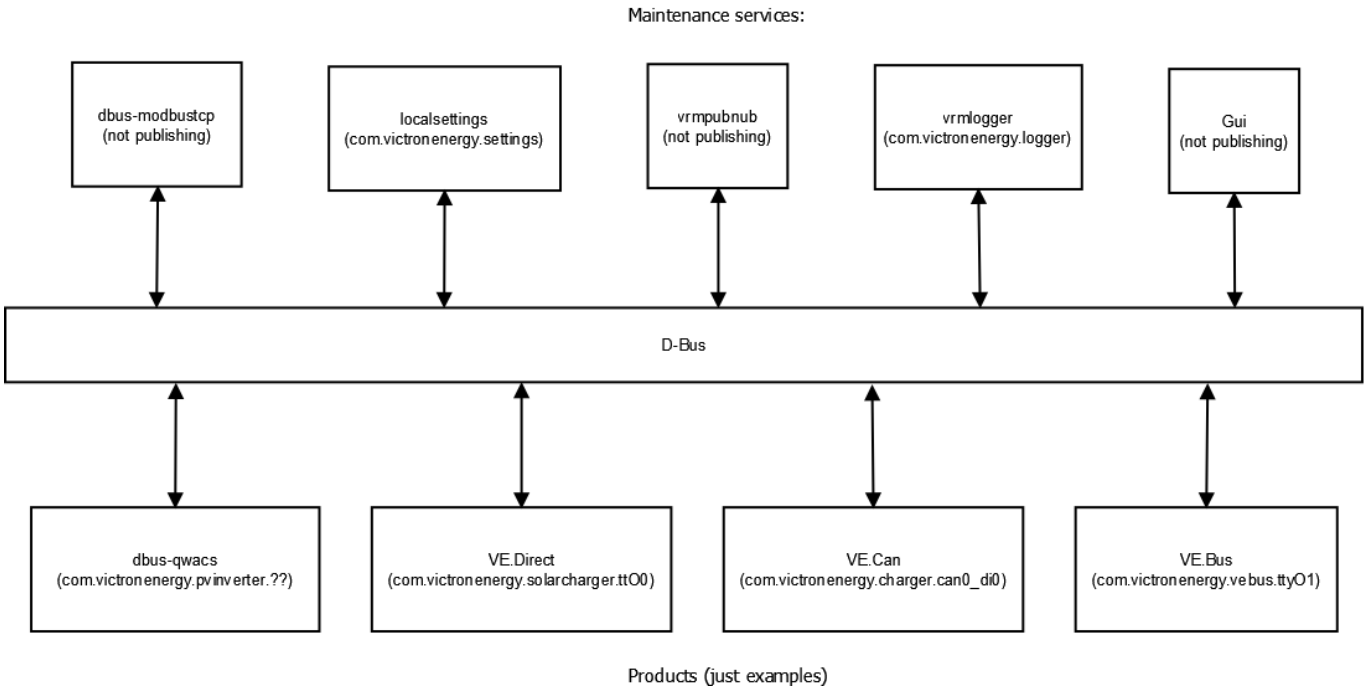
D-Bus API definition

Introduction

D-Bus is used as the main data exchange, to share values such as voltages, as well as settings and other data, between all the processes running on the CCGX. D-Bus is a common linux inter process communication mechanism, see google and the D-Bus page on wikipedia for more information.

Below diagram gives a good overview on the CCGX:



Original file

Basics

- Standard Linux D-Bus has two bus-es: Session and System. On the CCGX the System bus is used.
- The D-Bus connection-name of a D-Bus service always begins with com.victronenergy.
- For D-Bus services representing products connected to the system, the service name rules are:
  - Connection via serial port (onboard VE.Bus connection, VE.Direct ports, usb-serial converters): com.victronenergy.[product type].[tty name]
    Note that this can be VE.Direct, but also a GPS for example
  - Connections via canbus:
    `com.victronenergy.[product type].[interface]_di[device instance]_uc[unique-number]`
- As implied by above naming rules, when one executable connects to different products, it will make create multiple D-Bus services. An example is the process taking care of all canbus communications, vecan-dbus.
- Never use the fourth part in the service name (interface, tty name etc.) to deduct the DeviceInstance or connection method. The only purposes of it is to make the name is unique

and easy to identify from the command line. Use /DeviceInstance/ instead.
- Use SI units when publishing data on the D-bus.
  Exceptions: use kWh for energy and degrees Celsius for temperature.
- Processes representing products will connect to the dbus after they know what kind of product they will represent.
- Never change your the service name once on the D-Bus. Instead remove the service and register a new one.
- When a dbus service cannot communicate anymore with its direct counterpart, the process will disconnect from the D-Bus. And, if it has nothing else to do, will exit. An example is a BMV connected through VE.Direct stops communicating, perhaps because it is unplugged.
- Each object on the D-Bus represents one value. In VE.Bus for example, /Dc/0/Voltage is the voltage, /Dc/0/Current the current.
- An invalid value is represented by an empty array of integers. In Python this is `dbus.Array([], signature=dbus.Signature('i'), variant_level=1)`.

Implementation notes

- Applications publishing values to the dbus (ie a product driver), do not need to invalidate all values before going offline. Applications using values from dbus, such as the GUI or vrmlogger, need to monitor for nameowner changes / service watches. One reason: an application could also crash (so the dbus service dissapears without a chance of first sending out all invalids).
- Applications using values from the D-Bus, such as GUI or vrmlogger, should take care that when a new D-Bus service comes online, it might not immediately have all object paths.
- Applications publishing values to the D-Bus, should, when adding object paths after they have already created their D-Bus service, send a PropertiesChanged signal for each of the afterwards added paths.
- Applications publishing values to the D-Bus, that change their /DeviceInstance, should disconnect their service from the D-Bus and reconnect with the new /DeviceInstance value. Or, change your name, to trigger nameowner changes.
- When putting a D-Bus service online, there is no need to send out a PropertiesChanged signal for all object paths, because all consumers would do a GetValue at the root object when a new D-Bus service comes online.

Service name examples

Services representing products

The services below retrieve measurements from devices connected to the color control, and publish them on the D-Bus. The hyperlinks in the tables below refer to the source code of the projects on github. Services without hyperlink are not (yet) open sourced.

Below is a list of example possible and common service names.

Service name	Description
com.victronenergy.batter.ttyO2	
com.victronenergy.vebus.ttyO1	mk2-dbus
com.victronenergy.charger.ttyO0	Charger connected via VE.Direct at tty0

Service name	Description
com.victronenergy.solarcharger.ttyO2	Solar charger connected via VE.Direct at tty2
com.victronenergy.solarcharger.ttyUSB2	Solar charger connected via VE.Direct via USB cable
com.victronenergy.vebus.socketcan_can0_di0	VE.Bus system connected via canbus 0, and configured for device instance 0
com.victronenergy.vebus.socketcan_can0_di1	VE.Bus system connected via canbus 0, and configured for device instance 1
com.victronenergy.charger.socketcan_can0_di0	Skylla-i charger connected on canbus, device instance 0
com.victronenergy.pvinverter.qwacs_1	Quby AC sensors on input 1
com.victronenergy.pvinverter.fronius_137_37823	Fronius PV inverter with type ID 176 and serial 37823
com.victronenergy.pvinverter.vebusacsensor_input1	Victron AC Current Sensor on input 1
com.victronenergy.grid.ttyUSB0_di32_mb1	Carlo Gavazzi AC sensors on USB0 with device instance 32 and Modbus address 1
com.victronenergy.gps	USB/serial GPS retrieval
com.victronenergy.battery.lg_resu	LG resu battery
com.victronenergy.generator.startstop0	Generator start/stop using CCGX internal relay

Maintenance services

Process	Description
com.victronenergy.settings	interface to xml-based non-volatile settings storage (/data/conf/settings.xml)
com.victronenergy.gui	gui. takes care of buttons and lcd display
vrmlogger	Communication between CCGX and VRM Database
vrmpubnub	Two-communication via internet, (to be) used by mobile apps, Portal and VE Power Setup.
com.victronenergy.system	Computes statistics from devices (total power, consumption, power to grid...)
modbustcp	Modbustcp gateway, allows access to all connected products
com.victronenergy.fronius	Fronius device scan status
com.victronenergy.hub4	Hub-4 advanced control

Device instances

The device instance (object path /DeviceInstance) makes a device unique. The device instance is based on the connection method. The used rules are:

- NMEA2K devices report the NMEA2K device instance (max. 255).
- Devices connected to a (virtual) serial-port report from 256 up. Their instance is determined by the device name:
  - /dev/ttyOx: 256 + x
  - /dev/ttySx: 272 + x
  - /dev/ttyUSBx: 288 + x
  - COMx: 256 + x
- Quby AC sensors: 0 if on input 1, 1 if on output, 2 if on input 2
- Vebus AC sensors: 10 if on input 1, 11 if on output, 12 if on input 2

- Fronius PV inverters: 20 + x (x depends on order of detection)
- Carlo Gavazzi AC sensors: 30 + x (x depends on order of detection), can be changed in Gui
- SmartEnergyMeter / BLE networking: 40
- com.victronenergy.system (aka systemcalc): 0
- com.victronenergy.generator.startstop0: 0

Use of the DeviceInstance in CCGX configuration & settings

The device instance is used for several configuration items:

- Settings → System Setup → Main battery monitor
- Genset start/stop selection of battery monitor and ac source
- and perhaps more config items.

Use of the DeviceInstance on VRM

In logging to the VRM database (vrm.victronenergy.com) the device-instance is used, for example IV1[0] (input 1 voltage of device instance 0), and it (the instance) should therefore not change at random. Maximum of the instance field in the VRM database is 65535.

Use of the DeviceInstance on ModbusTCP

The ModbusTCP server, used by customers to query data from either the total system or separate devices, uses the ModbusTCP server in the addressing. The UnitID aka SlaveID in the request is mapped to the device instance.

This mapping is not one on one and has a bit of history, since we didn't realize at first that a lot of PLCs do not support a device instance above 247. Also one Customer reported that his PLC cant work with unitid 0.

More details: unitid2di.csv map in the sources. And also the user documentation, the modbustcp excelsheet and also the ModbusTCP FAQ.

Object-paths

The naming convention for new object paths is to write a full word. So /Ac/Voltage instead of /Ac/V and /Management instead of /Mgmt.

The list of available device type specific paths is here: https://github.com/victronenergy/venus/wiki/dbus.

Objects paths that all services need to implement

Object-path	Definition
/Mgmt/ProcessName	For example vcan_dbus

Object-path	Definition
/Mgmt/ProcessVersion	For example v1.02
/Mgmt/Connection	Textual description for connection method. To be used in the menu (“VE.Direct port 2”)

Object paths that are mandatory for services representing products

/ProductId
As defined in products.h in velib.

GetValue should return the decimal, as uint32 (ie 516).

GetText should return this as a hex string: 0x204.

/ProductName
As defined in products.c in velib, probably also on VE.Can as PGN 1F014, field 4 ‘Manufacturer’s Model ID’.

/FirmwareVersion
GetText: A string, containing the firmware version which is formatted equal as used in change logs of the product, and/or on labels etcetera. This string is also what the Venus GUI and VRM portal will show to users.

GetValue: Numeric representation of the firmware version. To be prepared for automatic firmware updates, make sure that the numeric value is such that a newer version leads to a higher number. And also that the number is in the same format in the firmware update files (if available for this product); so that when indexed in a library, the number can be used to check up-to-dateness of the product. All downstream software will not, and may not, format this number itself for showing the user. Instead it must use the string as returned by GetText.

/HardwareVersion
This should probably be similar as /FirmwareVersion, note though that this is given very little attention as its not used for Victron products.

/DeviceInstance
The Device Instance, see earlier chapter.

/Connected
Value 0 = not-connected, Value 1 = connected.

Non-mandatory standard object paths

/Serial String, containing one serialnumber. In a system with multiple units, the one from the master is shown.

/N2KUniqueNumber Int, containing the N2K unique number. In a system with multiple units, the one from the master is shown.

/CustomName String, containing an editable name. This can be set by a user to distinguish the device from other devices. When a product name is presented to a user, /CustomName is the preferred name

to present. When /CustomName is blank, /ProductName should be presented to the user. When /CustomName is changed, it will be stored in the device if the device supports this and otherwise in LocalSettings (com.victronenergy.settings).

Venus does not support Custom names for N2K/VE.Can devices. It can be added storing the custom name in a N2K Description field string; which is in PGN 0x1F016, Config info.

For VE.Direct, this is mapped to the VE_REG_DESCRIPTION1,

/Devices/...

Initially, the Devices object path was intented for products that operate in parallel or group mode; and because of that had only one service on the D-Bus. The /Devices path could then be used to address them individually. Examples of products operating in parallel mode are VE.Bus products (Inverters, Multi's, Quattro's), Skylla-i's and MPPT 150/70's. The first device will report under /Devices/0/... , the second device under /Devices/1/.... Etcetera. When relevant this numbering is kept equal to bus numbering. For example MK2-addressing.

Per 2019, this has changed: for solar chargers we are no longer grouping the data on one D-Bus service, instead each solar charger will go on the D-Bus with its own service. Still we use the /Devices path, but then to have a generic API to find all devices. This is used by the VregLink API: the mqtt-rpc vreg-device-list method scans each d-bus service for its entries on the /Devices sub-tree and returns the assembled data.

Some examples:

```
vedirect_interface will publish only one entry per service.

com.victronenergy.battery.ttyO2:
/Devices/0/CustomName        my-bmv
/Devices/0/DeviceInstance    258
/Devices/0/FirmwareVersion   v2.06
/Devices/0/ProductId         0x0201
/Devices/0/ProductName       BMV-700
/Devices/0/ServiceName       com.victronenergy.battery.ttyO2
/Devices/0/VregLink          (API)

vecan-dbus example, first item is a 1st generation solar charger (which has
a grouped entry on D-Bus). and the second one a 2nd generation solar
charger; which are all uniquely on D-Bus:

com.victronenergy.vecan.can0:
/Devices/2CC03E5D/CustomName        east-facing-array
/Devices/2CC03E5D/DeviceInstance    0
/Devices/2CC03E5D/FirmwareVersion   v1.02
/Devices/2CC03E5D/Manufacturer      358                              <-
from the nmea2000 standard.
/Devices/2CC03E5D/Nad               123                              <-
canbus network address.
/Devices/2CC03E5D/ProductId         0x0301
/Devices/2CC03E5D/ProductName       MPPT 150/60 Smart something.
/Devices/2CC03E5D/Serial            0015965 HQ1905944QQ              <- we
```

```
could consider splitting this up.
/Devices/2CC03E5D/ServiceName       com.victronenergy.solarcharger.something
/Devices/2CC03E5D/VregLink          (API)
/Devices/3DD03E5D/CustomName        east-facing-array
/Devices/3DD03E5D/DeviceInstance    0
/Devices/3DD03E5D/FirmwareVersion   v2.04
/Devices/3DD03E5D/Manufacturer      358
/Devices/3DD03E5D/Nad               123
/Devices/3DD03E5D/ProductId         0x0101
/Devices/3DD03E5D/ProductName       MPPT 150/70 Smart something
/Devices/3DD03E5D/Serial            0021124 HQ1250944QQ
/Devices/3DD03E5D/VregLink          (API)

Note that for the second entry, there is no ServiceName entered, since
because of the grouping there is not necessarily a group item.
```

/Interfaces/… The Interfaces object path is used when access to a certain product goes via known interfaces. For VE.Bus products for example you could find the MK2 in /Interfaces/, or the mk2-can and the mk2. The reasons to do this are:

- How is the device connected
- Publish / retrieve firmware versions of intermediate interfaces
- Diagnostics (where is the connection lost)

An example of the interfaces of a VE.Bus system connected through canbus:

/Interfaces/Mk2	
/Interfaces/Mk2/Version	1130132
/Interfaces/Mk2-can/	
/Interfaces/Mk2-can/Version	v1.12

D-Bus interfaces

All D-Bus object paths have the following interfaces:

Interface com.victronenergy.BusItem

(Variant value) GetValue()
Returns the value.

(String value) GetText()
Returns the value as string, including units. For example ‘21.3 W’. When invalid, it returns an empty string.

(Int32 retval) SetValue(Variant value)
Sets the specified value. Returns 0 when success, and something else when not allowed.

(Variant value) GetMin()
Returns the limit minimum.

(Variant value) GetMax()
Returns the limit maximum.

Following is only supported by com.victronenergy.settings

(Int 32 retval) SetDefault()
Sets the value(s) to default value. When this is called on a group (for example object path /Settings/Logscript, all values in this group are set to default.

(Variant value) GetDefault()
Returns the default value.

(Int32 retval) AddSetting(String group, String name, Variant defaultValue, String itemType, Variant minimum, Variant maximum)
Adds a new local setting according to the specified parameters. The possible itemType's are 'i' (integer), 'f' (float) and 's' (string). The minimum and maximum can be specified when the itemType is an integer or float. Minimum and maximum are ignored when both are specified as 0 (zero).

Interface org.freedesktop.DBus.Introspectable

(String data) Introspect()
Returns the introspection data in XML format. Interface org.freedesktop.DBus.Properties (Variant value) Get(String interface, String property)
Returns the value of specified interface and property. For example the interface com.victronenergy.BusItem and the property Valid.