JControl/Sticker

JAVA PROGRAMMABLE SUB-PDA DEVICE WITH 128X64 PIXEL GRAPHIC LCD, MULTIFUNCTION SWITCH, RTC, 128K FLASH MEMORY, BUZZER, RS232-INTERFACE, EXTENSION PORT AND SOFT REALTIME SUPPORT

- **Virtual Machine Core**
  - 8 Bit JAVA™ bytecode execution engine
  - 16 Bit processing word length
  - Max. 255 constant pool entries
  - 2.5k JAVA heap memory
  - 2 MIPS native core speed
  - Automatic garbage collection
  - Multi-threading support with extensions for soft realtime execution
- **Display**
  - 128x64 pixel graphic LCD
  - FSTN technology
  - Viewing area: 46.0 x 23.0 mm
  - Dot pitch: 0.36 mm
  - Contrast adjustable by software
- **Multifunction Switch**
  - 5 switching positions
  - Controls by lever and pushing
  - 2 positions up, 2 positions down
- **Flash Memory**
  - 2 banks of 64k Flash
  - 128 byte sectors
  - > 10,000 erase/write cycles
- **Real-Time-Clock (RTC)**
  - Philips PCF8563 type
  - 32.768kHz crystal oscillator
  - 0.25µA backup current @V_{CC}=3.0V
  - Alarm output activates JControl device
- **Power-Supply**
  - 3.3V power supply
  - Integrated step-up-converter, operating from 1.0-3.3V for battery supply
  - Battery connector for CR2430 lithium cell or 2 AAA-type batteries
  - Input for external power supply
  - External power-on trigger input
  - Power-on reset generator
  - Battery status monitor

- **Buzzer (Piezo Ceramic)**
  - Controlled by PWM output
  - 6.3kHz Resonance Frequency
- **RS232**
  - 3-Wire RS232-Interface
  - 3.5mm stereo-jack connector
  - On-board RS232 transceiver with auto-shutdown mechanism
  - 11 different baud rates from 300 up to 250.000bps
  - None, even or odd parity
  - Automatic flow control by `XON/XOFF` or `RTS/CTS`
- **I²C/SMBus Communication**
  - Master mode
  - 7 and 10 Bit addressing modes
- **I/O-Pins**
  - 12-pin Molex micro connector
  - 6 Digital General-Purpose I/O Pins
  - 2 I/Os usable as PWM outputs
  - 4 I/Os usable as Analog inputs
- **Physical Dimensions**
  - Version with CR2430 battery holder
    - Size: 60x42x10mm
    - Weight: 22g (w/o batteries)
  - Version with Twin-AAA-battery holder
    - Size: 60x42x20mm
    - Weight: 25g (w/o batteries)

© 2003 DOMOLOGIC Home Automation GmbH

DIMENSIONS AND CONNECTORS

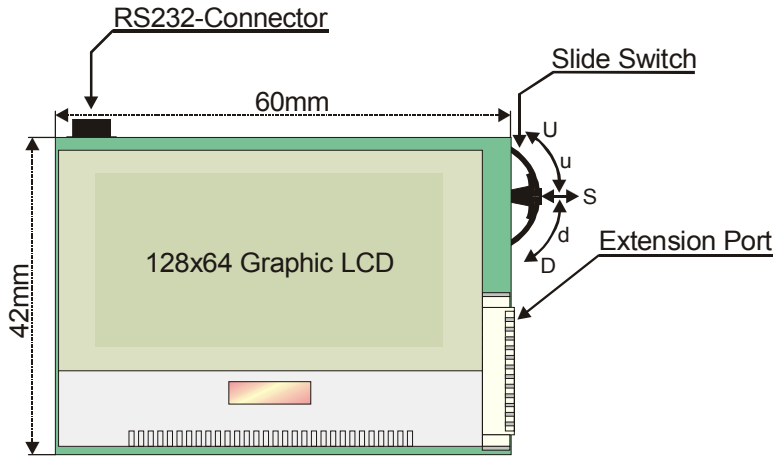


Fig. 1: Dimensions and Connectors of the JControl/Sticker

DEVICE VARIANTS

Sales Type	Battery Holder	Native Core Speed	Serial Baud Rates	Flash Organization
JControl/Sticker BC12	2x Battery AAA-Size (UM-4)	2 MIPS	300-250.000bps	512x128x2
JControl/Sticker BA12	1x Lithium Cell CR2430	2 MIPS	300-250.000bps	512x128x2

Table 1: Different Variants of the JControl/Sticker

© 2003 DOMOLOGIC Home Automation GmbH

GENERAL DESCRIPTION

The JControl/Sticker is a member of the JControl device family, designed as Sub-PDA and offering specific attributes for low-cost/low-power applications. It is programmable in JAVA, based on the JCVM8 8 Bit JAVA™ bytecode execution engine. The core runs with 2 MIPS native speed, providing 16 Bit processing word length, 2.5k JAVA heap memory, automatic garbage collection and multi-threading software execution. Applications in the fields of control, measurement and automation are supported by specific extensions for soft-realtime processing.

The JCVM8 offers a set of built-in classes, providing fundamental support of the JAVA programming language and access to all local peripheral components like LCD, multifunction switch, RTC, Flash memory, buzzer, communication ports (RS232, I²C), general purpose I/O pins, analog inputs and a pulse width modulator. Extended support is given by class libraries, linked automatically to the application by the JControl/DevelopmentSuite. This mechanism saves memory space, because exclusively the required classes are loaded to the system.

Application programs are loaded via a serial communication interface to the Flash memory, organized into two banks of 64kByte. Both banks may be used to store application software or non-volatile data.

Various information about the specific JControl device and its current state is available by accessing the *system properties*. In download mode, system properties may be read or written by remote using the *JControl Download Protocol*. Under normal operating conditions, the system properties can be accessed by application software using the methods `getProperty` and `setProperty` of the built-in class `jcontrol.system.Management`.

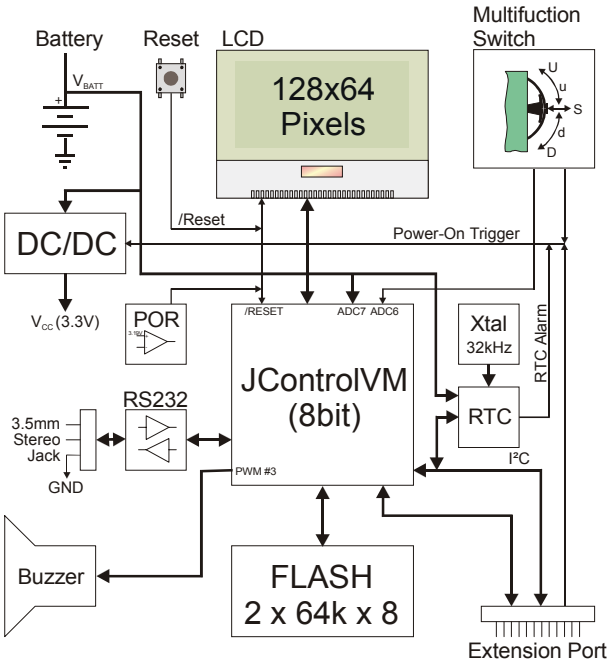


Fig. 2: JControl/Sticker Block Diagram

POWER SUPPLY AND SYSTEM RESET

The JControl/Sticker is powered by batteries inserted into the on-board battery holders or by an external power-supply connected to the extension port. The device operates with 3.3V DC internally, but thanks to the on-board DC/DC converter the voltage range for battery-based or external power supply is from 1.0 to 3.3V DC. Depending on the processing load the current consumption varies between 7 and 14mA @3.3V DC.

When the device is turned off, only the RTC (Real Time Clock) is supplied by the battery voltage to keep the current time. The device is turned on by the power-on trigger signal, starting the DC/DC-converter that supplies the system. The power-on trigger may be activated by one or more of the following conditions:

(1) Pushing-in the multifunction switch

(2) activating the alarm output of the RTC

(3) pulling the signal `/PWR_ON` of the extension port to GND

To ensure a reliable start up phase, a power-on reset generator (POR) holds the reset signal of the JCVM8 and LCD while the output voltage of the DC/DC converter is below 3.19V. During the JCVM8's initialization sequence, the alarm output of the RTC is activated permanently to hold the power-on trigger. When the device is switched off by software (e.g. using the method `powerOff()` of the built-in class `jcontrol.system.Management`), the alarm output will be released.

After the initialization sequence of the JCVM8 has been executed, the application stored in the first Flash bank (bank 0) will be started (sole exception: see chapter Real Time Clock (RTC)). Using the standard configuration of the JControl/Sticker, this will be the SystemSetup software, starting the user application afterwards.

© 2003 DOMOLOGIC Home Automation GmbH

The actual status of the battery voltage may be measured using the method `getPowerStatus()` of the built-in class `jcontrol.system.management`. It returns an 8-bit value, directly proportional to the input voltage of the DC/DC-converter (V_{BATT}). A return value of 255 is maximum, representing V_{BATT}=3.3V. The actual input voltage may be calculated using the following equation:

$$V_{BATT} = \frac{3300 * getPowerStatus()}{255} [mV]$$

FLASH MEMORY ORGANIZATION

The JControl/Sticker offers 2 banks of 64k Flash memory for application software or non-volatile data, numbered as bank 0 and bank 1 (*Fig. 3*). Both banks are organized as 512 sectors by 128 bytes, numbered from sector 0 to sector 511. The Flash memory's organization may be detected automatically by reading the system property `flash.format`. The returned string comprises the parameters `<number of sectors>x<bytes per sector>x<number of banks>` (e.g. "512x128x2" for the JControl/Sticker device).

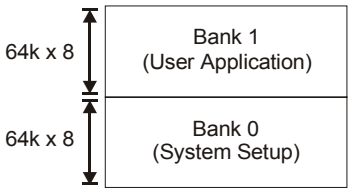


Fig. 3: The two Flash Banks of JControl/Sticker

One Flash bank can be selected for application software execution at a time. The current bank may be changed by restarting the JCVM8 using a different bank (see chapter *Power Supply and System Reset*). After system initialization, always the application software in bank 0 is executed.

By default, bank 0 is reserved for the SystemSetup software, implementing comfortable features to configure the device. The user application may be loaded into Flash bank 1.

The Flash memory can be used to store non-volatile data using the built-in class `jcontrol.io.Flash`. It provides methods to read and write complete sectors in any bank of the Flash memory.

Fig. 4 gives an overview of bank 0's internal structure: Application software is written upwards, starting at sector 0 and non-volatile data is stored downwards, starting at sector 510. This procedure reduces the possibility of resource conflicts between application software and data. To offer a linear ascending number of sectors (starting at sector 0) to the application, the class `jcontrol.io.Flash` maps access to the logical sector 0 to the physical sector 510 of the Flash memory, access to logical sector 1 to the physical sector 509 and so on. The uppermost sector of bank 0 (sector 511) is used to hold non-volatile system properties. The same principle is also used for Flash Bank 1, except that the uppermost sector is not holding the system properties.

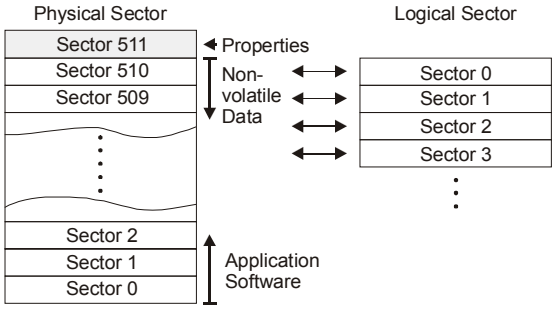


Fig. 4: Internal Structure of Flash Bank 0

For applications that makes use of the flash memory independently of its architecture, the external class `jcontrol.storage.FlashStream` is provided. It represents a memory cached data stream for reading and writing continuous data to or from the non-volatile flash memory.

DOWNLOAD MODE

The *system download mode* is a fundamental functionality of the JCVM8, implemented in every JControl device. It is used for uploading and downloading application software or data from a host computer to the Flash memory, for auto identification of the JControl device and for reading or writing system properties by remote. The download mode is used e.g. by the development tools like *JCManager* and *PropertyEdit*.

The system download mode may be entered by one of following four cases:

(1) Directly after the initialization sequence of the JCVM8: If no valid application software is

© 2003 DOMOLOGIC Home Automation GmbH

available in bank 0 of the Flash memory, the device enters the system download mode.

(2) During normal operating conditions: If the virtual machine is restarted by the method `switchBank()` of the built-in class `jcontrol.system.Management` and the new Flash bank contains no valid application software.

(3) The system download mode may be enforced by pushing the multifunction switch down while resetting the device as shown in *Fig. 5*.

(4) The mode may also be started by software calling the `run()`-Method of an instance of the built-in class `jcontrol.system.Download`.

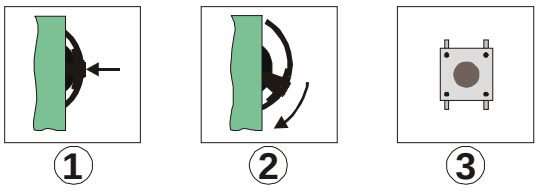


Fig. 5: Entering the System Download Mode

① Turn on the device, ② push the multifunction switch down and hold it down, ③ push the reset button at the backside of the JControl/Sticker

In the first three cases, a splash screen appears as shown in *Fig. 6*. The first line of the splash screen gives information about the JControl device profile ("JControl/Sticker"). The second line shows the build date of the JCVM8, represented as format `yyyymmddhhss` (`yyyy`=year, `mm`=month, `dd`=day, `hh`=hour, `ss`=second). The following "+0100" in the example is optional and gives information about the time zone. The build date is also available as system property `profile.date` and used by the tools to select an appropriate device profile. The bottom line shows the parameters of the RS232 interface, fixed to 19200 bps, 8 data bits, no parity and 1 stop bit.

Fig. 6: Splash screen of the system download mode

Using the Download Mode by Applications

The built-in class `jcontrol.system.Download` may be used to access or extend the download functionality by application software, e.g. to implement comfortable download or upload features for specific applications. The "File Upload" feature of the SystemSetup application makes use of this class for example.

When the system download mode is started by software (see case 4), no splash screen appears and the baud rate is not fixed to 19200bps, but set to the value held by the system property `rs232.baudrate` (default: 19200bps). If data is written to the flash memory, the download mode will perform a system reset when it is quitted. Otherwise it returns to the calling application. See API documentation for more information about this class.

SYSTEM SETUP

By default, the SystemSetup software is loaded to Flash bank 0 of the JControl/Sticker. Because bank 0 is always selected after power up (sole exception: see chapter RTC), the SystemSetup reads the system property `system.userbank` and checks, if a different bank should be selected for executing the user application. In this case, the JCVM8 is restarted using the selected bank. Otherwise, the SystemSetup menu appears on the display, offering a comfortable menu to change various settings of the device (*Fig. 7*). The menu may be enforced by pushing the multifunction switch up during system reset.

Fig. 7: Screenshot of the SystemSetup Menu

The sub-menus of the SystemSetup are:

"Battery status" displays a value directly proportional to the input voltage of the DC/DC-converter. The range is 0..100%.

"Time" is used to set the internal RTC (Real Time Clock) in German date and time format (hh:mm:ss dd.mm.yyyy)

"Display contrast" allows to adjust the contrast setting of the LCD. It sets the system property `display.contrast`.

"Sound" is used to enable or disable sounds, generated by the device in different situations. The system properties `buzzer.enable`,

© 2003 DOMOLOGIC Home Automation GmbH

`buzzer.systembeep` and `buzzer.keyboardbeep` will be modified (see chapter System Properties).

"Boot-Bank" selects the Flash bank from where the user application has to be started. It sets the system property `system.userbank`, evaluatedby the SystemSetup after power up.

"File upload" enables communication to a host computer, running JCManager, JCManagerPro or PropertyEdit. It starts the `run()`-method of the built-in class `jcontrol.system.Download`.

"Credits" just scrolls some text, press the switch to leave.

"Standby Wakeup" sets two features of the device: The auto power off delay (automatic standby time after last key was pressed) and the auto power on time (RTC alarm, turning the device on). The auto power off delay sets the system property `system.standbytimer`. Value 0 disables the auto power off feature. The auto power on time sets the alarm time of the RTC using the method `setAlarm()`. Please note that also the system property `rtc.poweronbank` will modified when using this method, ensuring that the same Flash bank is selected again when the device is turned on by the RTC alarm (see chapter RTC for more information).

"Demo" starts an application residing in a second archive of Flash bank 0 (if available).

"Off" manually turns off the device. The RTC keeps on running (if power supply is not interrupted).

DISPLAY

The JControl/Sticker comes with a 128x64 pixel monochrome graphic LCD in FSTN reflective technology. The display has a viewing direction of 6 o'clock and is driven by a separate display controller (Samsung S6B1713), mounted as chip-on-glass circuit on the top side of the component. To obtain a high data bandwidth, the communication between JCVM8 and display is realized by an 8 bit parallel interface.

The built-in class `jcontrol.io.Display` offers a set of methods for drawing pixels, lines, rectangles, circles, images, characters and strings on the display. It implements the interface `jcontrol.io.Graphics` for hardware abstraction. Images are supported using the pixel-based JControl Image File format (JCIF, revision 0001); fonts have to be formatted using the pixel-based JControl Font Definition format (JCFD, revision 0002). The class `jcontrol.io.Display` includes a proportional system font (8 pixel height) by default.

For detecting the display dimensions automatically, the system property `display.dimensions` returns a string comprising the parameters `<width>x<height>x<colour_depth>`, specified by "128x64x1" for the JControl/Sticker device. The coordinates of the display are organized from left to right and from top to bottom counting from 0 to size-1, see also *Fig. 8*.

The display contrast may be adjusted by software using the system property `display.contrast`. The value is automatically saved to Flash memory and restored by the system during power-up.

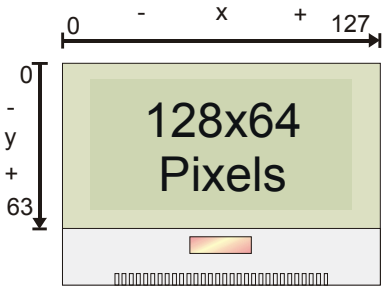


Fig. 8: Coordinates used by the LCD-class

MULTIFUNCTION SWITCH

The multifunction switch enables compound actions by a single knob. It features five degrees of freedom, controlled by lever and push operations.

The built-in class `jcontrol.io.Keyboard` provides methods to read the switch on character basis, including raw access, buffered access, automatic repetition and acoustic feedback.

© 2003 DOMOLOGIC Home Automation GmbH

Fig. 9 gives an overview of the moving directions of the switch. The letters 'S', 'u', 'U', 'd' and 'D' are returned by the method `read()` of the class `jcontrol.io.Keyboard` when the switch is moved to one of these positions. The letters are defined by the default keyboard map, that may be changed by an application program for software compatibility reasons.

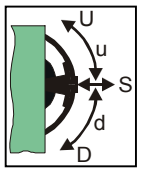


Fig. 9: Moving Directions of the Multifunction Switch

When the switch is pushed in, the power-on trigger is activated, turning the device on when it was turned off before. The push position is also connected to a dedicated I/O pin of the JCVM8 for evaluation.

The lever positions of the switch are connected by a voltage divider connected to the internal analog channel #6. Note that the position 'u' is always passed before the position 'U' is reached, and the position 'd' is always passed before the position 'D' is reached.

REAL TIME CLOCK (RTC)

The real time clock (RTC) of the JControl/Sticker is realized by a separate component, connected internally by the I²C bus to the JCVM8. During power-off state, the RTC is supplied by battery voltage, keeping the time information up-to-date. When the battery is changed, a small backup capacitor bypasses the power supply for a short time.

The RTC provides year, month, day, weekday, hours, minutes and seconds. Besides the current time, an alarm time is supported by hardware. The alarm output of the RTC is connected to the power-on trigger of the device, turning it on when it was turned off before.

The built-in class `jcontrol.system.RTC` implements methods for reading and writing the current time and alarm time. A time information is represented by an instance of the built-in class `jcontrol.system.Time`, combining the fields year, month, day, weekday, hours, minutes and seconds.

When an alarm time is set using the method `setAlarm()`, the value of the currently executed flash bank is copied to the system property `rtc.poweronbank`. This procedure ensures that the same flash bank with the same application will be selected by system when the device is turned on by alarm.

The RTC is based on the Philips PCF8563 integrated circuit, clocked by a separate 32.768 kHz quartz crystal. It operates on a supply voltage down to 1.0 V and consumes approx. 250µA @V_{DD}=3.0V (T_{amb}=25°C). The RTC allocates the I²C-bus slave addresses A3h (for reading) and A2h (for writing). It may also be accessed directly using the built-in class `jcontrol.comm.I2C`.

BUZZER

A piezo element is used as buzzer, mounted behind the LC Display. The buzzer is available for acoustic signals generated by the system or by an application. The system uses the buzzer for acoustic feedback on switch events and for signalling system exceptions. Both features may be enabled or disabled by the system properties `buzzer.systembeep` and `buzzer.keyboardbeep`. Additionally, an application software may control the buzzer using the external class `jcontrol.io.Buzzer`, implementing the interface `jcontrol.io.SoundDevice` for hardware abstraction. This class provides methods to activate the buzzer using a specified frequency (250...32767Hz) for a specified duration (in ms).

The system property `buzzer.enable` is provided to enable or disable the buzzer when it is used by an application.

Furthermore, the external class `jcontrol.toolkit.iMelody` is provided, playing complete melodies given by the iMelody-Format (`IMY`, published by the Infrared Data Association, IrDA).

The buzzer is controlled internally by the reserved PWM channel #3. The piezo element has a diameter of 20mm and its resonance frequency at 6.3kHz.

© 2003 DOMOLOGIC Home Automation GmbH

RS232 COMMUNICATION

The JControl/Sticker provides a serial communication interface according to RS232. The signals are available at the backside of the device on a 3.5mm stereo jack socket. The pin assignment of an appropriate jack is shown in *Fig. 10*. Optionally, two signals for flow control with CMOS/TTL levels are available at pin 4 (output signal `RTS`) and pin 3 (input signal `CTS`) of the extension port.

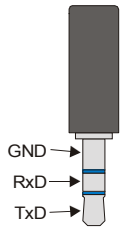


Fig. 10: Pin Assignment of RS232 Connector Jack

The built-in class `jcontrol.comm.RS232` provides methods for reading, writing and configuring the RS232 interface. It supports buffered read access and operates on `byte`, `char`, `string` and `utf8` basis. Automatic echoing is also supported by the `readLine()` method.

The RS232 communication interface supports 11 different baud rates, starting from 300 up to 250.000bps. This includes the MIDI-baud rate of 31250bps. The baud rate is changed using the method `setBaudrate()` of the built-in class `jcontrol.comm.RS232` (see Table 2 for a list of all valid settings). When an application attempts to set an unsupported baud rate, always the fall-back setting 19200bps is used. When no baud rate value is set by the application, the default value specified by the system property `rs232.baudrate` is used.

Additionally, the RS232 communication interface supports a parity bit (9th data bit) as well as flow control (by `XON/XOFF` or `RTS/CTS`). All options are defined by the current *communication parameters*, configured using method `setParams()` of the built-in class

`jcontrol.comm.RS232`. As shown in *Fig. 8*, the options are combined to a single bitmask. Appropriate constant field values are defined by the class `jcontrol.comm.RS232`. When the parameters are not changed by the application software, always the default settings specified by the system property `rs232.params` are used.

Baud Rate	Parameter for `setBaudrate`	Comment
300	300	
600	600	
1200	1200	
2400	2400	
4800	4800	
9600	9600	
19.200	19200	Fall-back setting
31.250	31250	MIDI
62.500	62	
125.000	125	
250.000	250	

Table 2: Supported Baud Rates

The following parity modes are supported: “`8N1`” (8 data bits, no parity, 1 stop bit), “`8E1`” (8 data bits, even parity, 1 stop bit) and “`8O1`” (8 data bits, odd parity, 1 stop bit). For flow control, two different modes are supported: Software flow control (by `XON/XOFF`) and hardware flow control (by `RTS/CTS`). Software flow control uses the ASCII-codes `XON` (`0x11`) and `XOFF` (`0x13`), and hardware flow control is realized by the extension port signals `RS232_RTS` (pin 4) and `RS232_CTS` (pin 3).

A RS232 transceiver MAX3221 is used for voltage conversion, meeting EIA/TIA-562 specifications down to 2.7V. The circuit supports auto shutdown, reducing the supply current to less that 10µA when it is not used.

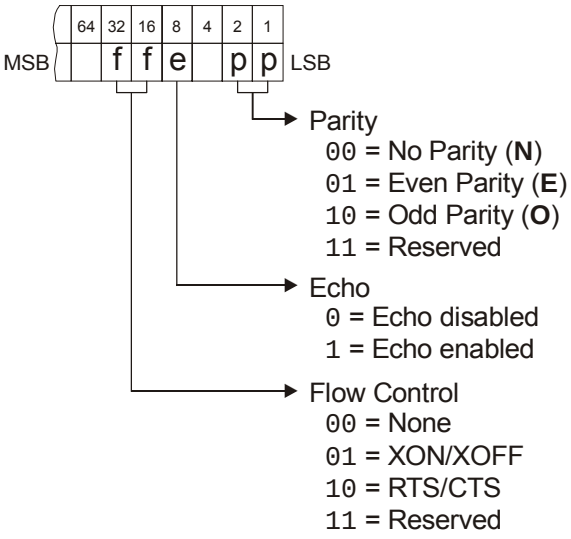


Fig. 11: RS232 Communication Parameters

© 2003 DOMOLOGIC Home Automation GmbH

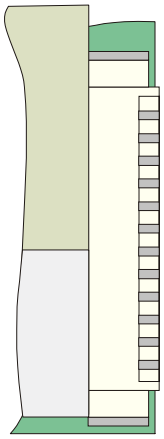
EXTENSION PORT

For system extensions, a 12-pin extension port connector is available at the right edge of the JControl/Sticker. It provides the internal operating voltage (+3.3V), a connector for external battery supply, a power-on trigger input, I²C bus and six universal I/O-pins.

Fig. 12 gives an overview of the extension port (Molex micro connector type 53261-1290). For adaptation, use a Molex 51021-1200 female crimp connector.

Pin V_{cc} (1) provides the internal operating voltage of +3.3V in reference to `GND` (12). V_{cc} may be used to supply external hardware by 3.3V, generated by the DC/DC-converter of the JControl/Sticker. Pin V_{BATT} (11) is a connector for the battery voltage, used to supply the device by an external power supply. Because this pin is connected in parallel to the internal power supply, the batteries inserted into the on-board battery holders have to be removed when using it.

The signal `/PWR ON` (2) is the power-on trigger input, pulled up internally to V_{CC} by a 1MΩ resistor. If this signal is tied to `GND`, the JControl/Sticker turns on. Some I/O pins are available at pin 3 to pin 8. Depending on the RS232 configuration, pins 3 and 4 may also be used for hardware flow control. The I²C bus interface is provided by the signals `I²C_SCL` (9) and `I²C_SDA` (10).



PIN	Function
● 1	V_{cc} (3.3V)
2	PWR ON
3	GPIO #5 / ADC #3 / RS232_CTS
4	GPIO #4 / ADC #2 / RS232_RTS
5	GPIO #3 / ADC #1
6	GPIO #2 / ADC #0
7	GPIO #1 / PWM #1
8	GPIO #0 / PWM #0
9	I²C_SCL
10	I²C_SDA
11	V_{BATT} (1.0-3.3V)
12	GND

Connector Type:
PCB: Molex 53261-1290
Crimp: Molex 51021-1200

Fig. 12: Extension Port of the JControl/Sticker

I/O PINS (GPIO, PWM, ADC)

Six I/O pins are available for system extensions at the extension port connector, numbered from pin 3 to pin 8. Each pin may be used as digital input or output (*General Purpose I/O*, GPIO). The built-in class `jcontrol.io.GPIO` is provided to control the digital I/Os, numbered as GPIO #0 to GPIO #5. Four different configuration modes are supported:

- `FLOATING`: Standard digital input
- `PULLUP`: Digital input with integrated pull up resistor (60-240kΩ)
- `PUSHPULL`: Standard digital output
- `OPENDRAIN`: Digital output, set to high-impedance state when `HIGH`

Two of the ten GPIO pins are also connected to an integrated Pulse Width Modulator (PWM) with a resolution of up to 8 bits. This feature is controlled by the built-in class `jcontrol.io.PWM`. The generated signals are available via the PWM channels 0 and 1. The device uses a single frequency generator for all channels, hence the frequency of the channels has to be the same. Please note that every pin configured as PWM output is not available as GPIO.

Four pins are connected to the internal 8-bit A/D converter and may be used as analog inputs. The built-in class `jcontrol.io.ADC` is provided to control this feature. When a pin is used as analog input, it should be configured to `FLOATING` mode using the class `jcontrol.io.GPIO`. The following table gives an overview on the features of each pin.

© 2003 DOMOLOGIC Home Automation GmbH

Extension Port Pin	GPIO					PWM	ADC
	Channel #	Floating Input	Input With Pullup	Push Pull Output	Open Drain Output	Channel #	Channel #
8	0	✓	✓	✓	✓	0	-
7	1	✓	✓	✓	✓	1	-
6	2	✓	✓	✓	✓	-	0
5	3	✓	✓	✓	✓	-	1
4	4	✓	✓	✓	✓	-	2
3	5	✓	✓	✓	✓	-	3

Table 3: Features of pins of the extension port

I²C/SMBUS COMMUNICATION

A separate I²C/SMBus communication interface is available at the extension port of the JControl/Sticker.

The I²C bus is a de facto standard for on-board inter integrated circuit communication. It was developed by Philips Semiconductors in the early 1980's. Many integrated circuits supporting the I²C bus, including the integrated RTC of the JControl/Sticker. SMBus is a kind of extended I²C bus, developed by Intel in 1995 as System Management Bus. It is used e.g. in personal computers and servers for low-speed system management communications. Mostly, the SMBus is used to interconnect the sensors for temperatures, voltages, rotation speed of fans etc.

The built-in class `jcontrol.comm.I2C` provides methods to use the JControl device as bus master. It supports 7 bit and 10 bit addressing schemes as well as reading and writing single chars or byte streams. It implements a simple hardware layer, therefore any bus error and any arbitration lost results in an `IOException` after a few retries. To avoid blocking, the class implements a bus timeout (in contrast to the I²C bus specification, but complementary to the SMBus specification).

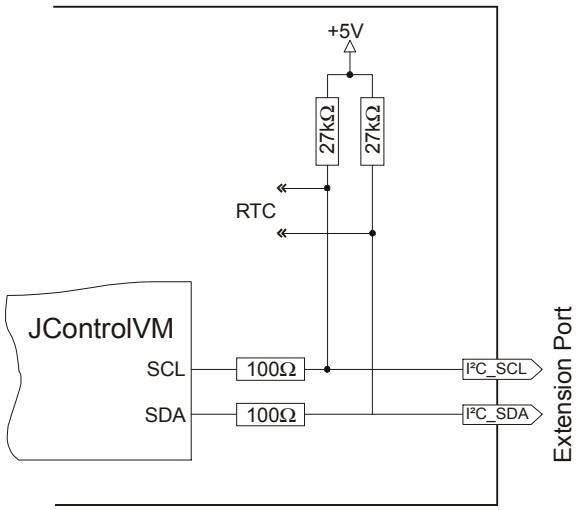


Fig. 13: Electrical wiring diagram of I²C bus/SMBus signals

The signal `I²C_SCL` (pin 9) is the clock signal of the I²C bus (or `SMBCLK` of SMBus). The signal I²C_SDA (pin 10) is the data signal of the I²C bus (or `SMBDAT` of SMBus). Both signals are pulled up to VCC internally using a 27kΩ resistor, Fig. 13 shows the internal electrical wiring diagram. When connecting external devices, note that the on-board RTC PCF8563 is also connected, allocating the addresses A3h for reading and A2h for writing.

© 2003 DOMOLOGIC Home Automation GmbH

JCVM8 RESTRICTIONS

Not all JAVA features are implemented by the JCVM8. The following list gives an overview on the restrictions:

- Data type `int` is limited to 16 bit processing word length (not 32 bit)
- Data types `long`, `float` and `double` are not implemented. When used, one of the following two error codes is generated (context dependent):
  - `BytecodeNotSupportedError` (6)
  - `UnsupportedArrayTypeError` (9)
- The number of constants in the constant pool is limited to 255 (will be checked by the JCManager before upload)
- Cast check for primitive arrays is not supported and causes an error (`NotImplementedError`)
- It is not possible to call object methods on primitive arrays, e.g.

  ```
  new int[25].equals(myObject)
  ```

- Some exceptions can not be catched by an application, because they generate an error code. When thrown, the JCVM8 is restarted in error condition and the error handler is called (see also chapter *Error Codes*).
- Implementation of classes in the package `java.lang` is incomplete (see JControl JAVADOC)

ERROR CODES

When an exception is thrown and not handled by the application, the JCVM8 generates an error code. Some of the errors (listed in the following *Table 4*) are specific to the JCVM8 and not common in the JAVA programming language (labeled with [1]). Other error codes are *masked exceptions*, because they are generated instead of an exception (labeled with [2]).

Every error restarts the JCVM8 in error condition. Afterwards the method `onError()` of the built-in class `jcontrol.system.ErrorHandler` is invoked. More details about the error state is passed by parameters to the `onError()` method.

The built-in error handler may be overwritten by a user-defined error handler stored in Flash bank 0. See the error handler included in the SystemSetup software for demonstration.

Following table gives an overview on the error codes generated by the JCVM8.

ID	Name	Description
1	`HandleError` [1]	Internal VM error
2	`NullPointerException` [2]	Attempt to use `NULL` where an object is required.
3	`OutOfMemoryError`	Generated when no memory is available
4	`BytecodeNotAvailableError` [1]	Attempt to execute an invalid bytecode
5	`BytecodeNotSupportedError` [1]	Attempt to execute an unsupported bytecode, e.g. bytecodes for 64-bit arithmetic or floating point processing
6	`BytecodeNotDefinedError` [1]	Attempt to execute an undefined bytecode
7	`ArithmeticException` [2]	Exception during arithmetic processing, e.g. division by zero
8	`NegativeArraySizeException` [2]	Attempt to create an array with negative size
9	`UnsupportedArrayTypeError` [1]	Arrays of this type are not supported
10	`ArrayIndexOutOfBoundsException` [2]	Array index is out of bounds
11	`ClassCastException` [2]	Attempt to cast an object which is not of an appropriate runtime type.
12	`NoCodeError` [1]	Thrown when a method is called that implements no code
13	`WaitForMonitorSignal` [1]	Used internally by the VM
14	`ExternalNativeError` [1]	Generated when a native method is called that is not stored in ROM
15	`FatalStackFrameOverflowError` [1]	Generated when the stack size is not sufficent

© 2003 DOMOLOGIC Home Automation GmbH

ID	Name	Description
16	InstantiationException [2)]	Attempt to instantiate an abstract class or interface
17	IllegalMonitorStateException [2)]	E.g. when a wait is called without an appropriate monitor
18	UnsatisfiedPrelinkError [1)]	Error due to a failed prelinking process
19	ClassFormatError [1)]	Generated when a failed while class loading
20	ClassTooBigError [1)]	The size of a class exceeds the limitations
21	PreLinkError [1)]	Error due to a failed prelinking process
22	PreLinkedUnresolvedError [1)]	Error due to a failed prelinking process
23	UnsupportedConstantTypeError [1)]	Generated when the type of a constant is not supported by the JCVM8 (long, float or double)
24	MalformatedDescriptorError [1)]	Error while dereferencing constant pool, e.g. due to wrong class file format
25	RuntimeRefTableOverrunError [1)]	More class references used than specified in a class file
26	NoSuchFieldError	Referenced field not found
27	IllegalAccessError	Tried to access a field or method from wrong scope (e.g. private)
28	NoSuchMethodError	Could not find referenced method
29	TooMuchParametersError [1)]	A method uses more parameters than supported by the JCVM8 (max. 16)
30	ThrowFinalError [1)]	Uncatched user defined exception. Exception name is passed to the onError( ) method.
31	NoClassDefFoundError [1)]	Unable to find a class by name
32	IndexOutOfBoundsException [2)]	Thrown by some methods using String or array parameters and indices that are out of bounds
33	ArrayDimensionError [1)]	Generated when an array is created with more than 2 dimensions (only 1 or 2 dimensions supported)
34	DeadlockError [1)]	Generated by the JCVM8 scheduler when two or more threads inheriting from each other
35	IncompatibleClassChangeError	Generated when an interface is invoked for an object, that is not implementing the interface
36	NotImplementedError [1)]	Generated when an unimplemented JAVA feature is used

Table 4: Error Codes generated by the JCVM8

[1)] Error codes generated exclusively by the JCVM8. Not common in the JAVA programming language.

[2)] JCVM8 error codes generated by the JCVM8 instead of exceptions. Can not be handled by an exception handler. May be replaced by JAVA exceptions in future revisions of the JCVM8.

© 2003 DOMOLOGIC Home Automation GmbH

SYSTEM PROPERTIES

System properties providing specific information about the JControl device. All properties are identified by a fixed string (the content is always formatted as string). The properties may be read or written using the methods `getProperty()` and `setProperty()` of the built-in class `jcontrol.system.Management`. In download mode, the tool PropertyEdit may be used to read or write the properties by remote.

The system properties are categorized into ROM properties and non-volatile properties. ROM properties are stored in read-only memory of the device and can not be changed. Non-volatile properties are held in the upper sector of Flash bank 0 and may be changed by software.

Key	Type	Value	Description
`profile.name`	String	JControl/Sticker	JControl Profile Name
`profile.date`	String	yyyymmddhhmm	Date of JCVM build
`system.heapsize`	Int	2688	Size of internal JAVA heap memory
`flash.format`	String	“512x128x2”	Flash Organization (bytes x blocks x banks)
`io.gpiochannels`	Int	6	Number of GPIO channels
`io.pwmchannels`	Int	2	Number of PWM channels
`io.adcchannels`	Int	4	Number of ADC channels
`display.dimensions`	String	“128x64x1”	Display dimensions (width x height x colour_depth)

Table 5: ROM Properties (saved in ROM, read access only)

Key	Type	Range	Default	Description
`system.standbytimer`	Int	0..32767	60	Auto-standby time (in seconds)
`system.userbank`	Int	0..1	0	Flash bank used for user application
`rtc.poweronbank`	Int	0..1	0	Bank selected to start application after power on initiated by RTC alarm
`buzzer.enable`	Bool	true, false	true	Enable or disable buzzer to be used by application software
`buzzer.systembeep`	Bool	true, false	true	Enable or disable system sound (set independent from `buzzer.enable`)
`buzzer.keyboardbeep`	Bool	true, false	true	Enable or disable keyboard beep (set independent from `buzzer.enable`)
`display.contrast`	Int	0..255	40	LCD contrast adjustment
`rs232.params`	Int	0, 1, 8, 9	0	Bitmask holding RS232 configuration ▪ Bit 1:0 `00` = No Parity `01` = `PARITY_EVEN` enabled `10` = `PARITY_ODD` enabled ▪ Bit 3 `1` = `ECHO` enabled ▪ Bit 5:4 `00` = No flow control `01` = `FLOWCONTROL_XONOFF` enabled `10` = `FLOWCONTROL_RTSCTS` enabled
`rs232.baudrate`	Int	300, ..., 31250, 62, ..., 250	19200	Sets default RS232 Baudrate

Table 6: Non-Volatile Properties (saved in Flash, read and write access)

© 2003 DOMOLOGIC Home Automation GmbH

SUPPORTED DATA FORMATS

The device supports following data formats:

Format used for	Format suffix	Rev.	Description	Used by class	Editor
Images	JCIF	0001	JControl Image File 8-Bit pixel-based image definition format	`jcontrol.io.Display`	PictureEdit
Fonts	JCFD	0002	JControl Font Definition 8-Bit pixel-based font definition format	`jcontrol.io.Display`	FontEdit
Melodies	IMY	V1.2	iMelody Melody format specified by Infrared Data Association (IrDA)	`jcontrol.toolkit.iMelody`	MelodyEdit

Table 7: Supported Data Formats for the JControl/Sticker

The format specifications are available online at http://www.jcontrol.org.

SUPPORTED JAR LIBRARIES

The device supports base JAR Libraries as listed below:

JCVM8 Build Date	Path JControl/profiles/jar/...	Type	Description
20030109	builtin/JControl_Sticker_20030109.jar	built-in	Library of all classes provided internally
	standard/JControl_ALL_20030206_lib.jar	standard	Standard JControl API programming environment
20030205	builtin/JControl_Sticker_20030205.jar	built-in	Library of all classes provided internally
	standard/JControl_ALL_20030206_lib.jar	standard	Standard JControl API programming environment
20030303	builtin/JControl_Sticker_20030228.jar	built-in	Library of all classes provided internally
	standard/JControl_ALL_20030206_lib.jar	standard	Standard JControl API programming environment
20030404	builtin/JControl_Sticker_20030404.jar	built-in	Library of all classes provided internally
	standard/JControl_ALL_20030404_lib.jar	standard	Standard JControl API programming environment
20030512	builtin/JControl_Sticker_20030512.jar	built-in	Library of all classes provided internally
	standard/JControl_ALL_20030404_lib.jar	standard	Standard JControl API programming environment
	optional/JControl_vole_20030505.jar	optional	Vole Graphical User Interface package
20030620	builtin/JControl_Sticker_20030620.jar	built-in	Library of all classes provided internally
	standard/JControl_ALL_20030620_lib.jar	standard	Standard JControl API programming environment
	optional/JControl_vole_20030620.jar	optional	Vole Graphical User Interface package

© 2003 DOMOLOGIC Home Automation GmbH

JCVM8 Build Date	Path JControl/profiles/jar/...	Type	Description
20030703	builtin/JControl_Sticker_20030620.jar	built-in	Library of all classes provided internally
	standard/JControl_ALL_20030620_lib.jar	standard	Standard JControl API programming environment
	optional/JControl_vole_20030620.jar	optional	Vole Graphical User Interface package
20030722	builtin/JControl_Sticker_20030722.jar	built-in	Library of all classes provided internally
	standard/JControl_ALL_20030620_lib.jar	standard	Standard JControl API programming environment
	optional/JControl_vole_20030620.jar	optional	Vole Graphical User Interface package
20030825	builtin/JControl_Sticker_20030825.jar	built-in	Library of all classes provided internally
	standard/JControl_ALL_20030620_lib.jar	standard	Standard JControl API programming environment
	optional/JControl_vole_20030620.jar	optional	Vole Graphical User Interface package

Table 8: Base JAR libraries supported by versions of the JControl/Sticker

BUILT-IN PACKAGES

Summary of Packages

Package	Description
`jcontrol.comm`	Complex communication features for JControl.
`jcontrol.io`	Classes for basic I/O and peripheral control.
`jcontrol.lang`	Replacement classes, fundamental to the design of the JAVA programming language.
`jcontrol.system`	JControl core classes and JControl specific JAVA extensions.
`java.lang`	Provides classes that are fundamental to the design of the JAVA programming language. Subset of the standard-package `java.lang`.
`java.io`	Subset of the standard `java.io`-package (only `java.io.IOException`)

Packages in Detail

Name	Type	Description
`Package jcontrol.comm`		
`ConsoleInputStream`	Interface	Provides a set of high-level communication methods to read from a console.
`ConsoleOutputStream`	Interface	Provides a set of high-level communication methods to write to a console
`RS232`	Class	Implements RS232 communication for JControl
`Package jcontrol.io`		
`ADC`	Class	Control of JControls analog-digital converter. Used to measure the voltage at portpins connected to the internal A/D converter
`BasicInputStream`	Interface	Interface providing a set of low-level communication methods for reading from a stream
`BasicOutputStream`	Interface	Interface providing a set of low-level communication methods

© 2003 DOMOLOGIC Home Automation GmbH

Name	Type	Description
		for writing to a stream
ComplexInputStream	Interface	Interface providing a set of high-level communication methods for reading from a stream
ComplexOutputStream	Interface	Interface providing a set of high-level communication methods for writing to a stream
Display	Class	Class to control the on-board 128x64 BW-LC-Display. Coordinates are from left to right and from top to bottom counting from 0 to size-1.
Drawable	Interface	Defines Object-behaviour for use with jcontrol.io.Display.drawImage()
File	Interface	Provides a set of methods for file-system access
Flash	Class	Raw access to JControl's integrated Flash memory. The methods are designed to access complete sectors of memory, not single bytes.
Graphics	Interface	Interface definition for graphics devices (e.g. offscreen images, displays, ...)
I2C	Class	Controls I^2C devices connected to JControl.
Keyboard	Class	Accesses JControl's keyboard, the multifunction switch in the case of the JControl/Sticker
Portpins	Class	Controls available portpins of JControl
PWM	Class	Controls the Pulse Width Modulation outputs of JControl
Resource	Class	Implements read access to the application's resource. The resource stores additional application data like pictures, fonts, text etc.
Package jcontrol.lang		
Deadline	Class	Constructs a new JControl deadline, useful for soft real-time applications
ThreadExt	Class	Thread extensions for JControl, useful for soft real-time applications
Math	Class	Provides some simple math functions
Package jcontrol.system		
Download	Class	Manages to download new JAVA applications to a JControl module
ErrorHandler	Class	The JControl Error-Handler. May be overwritten to implement more comfortable error handlers.
Management	Class	Controls various system management functions
RTC	Class	Access to JControl's integrated Real Time Clock
Time	Class	The Time object stores a date and time.
Package java.lang		
Exception	Class	The class Exception and its subclasses are a form of Throwable, indicating conditions that a reasonable application might want to catch
Integer	Class	The Integer class wraps a value of the primitive type int in an object. An object of type Integer contains a single field whose type is int.
Object	Class	Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.
Runnable	Interface	The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.
String	Class	The String class represents character strings. All string

© 2003 DOMOLOGIC Home Automation GmbH

Name	Type	Description
		literals in JAVA programs, such as "abc", are implemented as instances of this class
Thread	Class	A Thread is a thread of execution in a program. The JAVA Virtual Machine allows an application to have multiple threads of execution running concurrently.
Throwable	Class	The Throwable class is the superclass of all errors and exceptions in the JAVA language. Only objects that are instances of this class (or one of its subclasses) are thrown by the JAVA Virtual Machine or can be thrown by the JAVA throw statement. Similarly, only this class or one of its subclasses can be the argument type in a catch clause.

EXTENSION PORT SUMMARY

Pin	Name	Description
1	V_{CC}	Internal Power Supply (3.3V) May be used to supply external hardware [3)]
2	/PWR_ON	Power On Trigger Input (Active low signal turns the device on)
3	GPIO #5 ADC #3	▪ GPIO channel #5 ▪ Input modes: FLOATING, PULLUP ▪ Output modes: PUSHPULL, OPENDRAIN ▪ ADC channel #3
4	GPIO #4 ADC #2	▪ GPIO channel #4 ▪ Input modes: FLOATING, PULLUP ▪ Output modes: PUSHPULL, OPENDRAIN ▪ ADC channel #2
5	GPIO #3 ADC #1	▪ GPIO channel #3 ▪ Input modes: FLOATING, PULLUP ▪ Output modes: PUSHPULL, OPENDRAIN ▪ ADC channel #1
6	GPIO #2 ADC #0	▪ GPIO channel #2 ▪ Input modes: FLOATING, PULLUP ▪ Output modes: PUSHPULL, OPENDRAIN ▪ ADC channel #0
7	GPIO #1 PWM #1	▪ GPIO channel #1 ▪ Input modes: FLOATING, PULLUP ▪ Output modes: PUSHPULL, OPENDRAIN ▪ PWM channel #1
8	GPIO #0 ADC #0	▪ GPIO channel #0 ▪ Input modes: FLOATING, PULLUP ▪ Output modes: PUSHPULL, OPENDRAIN ▪ ADC channel #0
9	I²C_SCL	Clock Signal of I²C-Bus (SMBCLK of SMBus)
10	I²C_SDA	Data Signal of I²C-Bus (SMBDAT of SMBus)
11	V_{BATT}	Positive Supply Voltage [4)]
12	GND	Ground Voltage

Table 9: Extension Port Summary

[3)] Maximum supply current for external hardware:
@V_{IN} = 1.5V: 30mA max.
@V_{IN} = 3.0V: 75mA max.

© 2003 DOMOLOGIC Home Automation GmbH

[4] External power supply/battery voltage range:
$1.0V \leq V_{IN} \leq 3.3V$

Information furnished is believed to be accurate and reliable. However, DOMOLOGIC Home Automation GmbH assumes no responsibility for the consequences of use such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of DOMOLOGIC Home Automation GmbH. Specifications mentioned in this publication are subject of change without notice. This publication supersedes and replaces information previously supplied. DOMOLOGIC Home Automation GmbH products are not authorized to use as critical components in life support devices or systems without express written approval of DOMOLOGIC Home Automation GmbH.

© 2003 DOMOLOGIC Home Automation GmbH – All Rights Reserved