WARNING NOTICES

IMPORTANT SAFETY INSTRUCTIONS

1. Read instructions – All the safety and operating instructions should be read before the appliance is operated.
2. Retain instructions – The safety and operating instructions should be retained for future reference.
3. Heed Warnings – All warnings on the appliance and in the operating instructions should be adhered to.
4. Follow instructions – All operating instructions should be followed.
5. Water and Moisture – Do not use this appliance near water – for example, near a bath tub, wash bowl, kitchen sink, or laundry tub, in a wet basement, or near a swimming pool, and the like.
6. Ventilation – Slots and openings in the cabinet are provided for ventilation and to ensure reliable operation of the appliance and to protect it from overheating, and these openings must not be blocked or covered. The openings should never be blocked by placing the appliance on a bed, sofa, rug or other similar surface. This appliance should never be placed near a radiator or heat register. This appliance should not be placed in a built-in installation such as a bookcase or rack unless proper ventilation is provided or the manufacturer's instructions have been adhered to.
7. Power-Cord Protection – Power-Supply cords should be routed so that they are not likely to be walked on or pinched by items placed upon or against them, playing particular attention to cords at plugs, convenience receptacles, and the point where they exit from the appliance.
8. Use of the AC to DC 12V power adapter supplied with this product is mandatory!
9. Lightning – For added protection for this product during a lightning storm, or when it is left unattended and unused for long periods of time, unplug it from the wall outlet. This will prevent damage to the product due to powerline surges.
10. Overloading – Do not overload wall outlets and extension cords as this can result in a risk of fire or electric shock.
11. Object and Liquid Entry – Never push objects of any kind into this product through openings as they may touch dangerous voltage points or short-out parts that could result in fire or electric shock. Never spill liquid of any kind on the product.
12. Servicing – Do not attempt to service the product yourself, as opening or removing covers may expose you to dangerous voltage or other hazards. Refer all servicing to qualified service personnel.
13. Damage Requiring Service – Unplug this product from the wall outlet and refer servicing to qualified service personnel under the following conditions:
   When the power-supply cord or plug is damaged.
   If liquid has been spilled, or objects have fallen into the product.
   If the product has been exposed to rain or water.
   If the product does not operate normally by following the operating instructions. Adjust only those controls that are covered by the operating instructions as an improper adjustment of other controls may result in damage and will often require extensive work by a qualified technician to restore the product to its normal operation.
   If the product has been dropped or the cabinet has been damaged.
   When the product exhibits a distinct change in performance – this indicates a need for service.
14. Heat – The product should be situated away from heat sources such as radiators, heat registers, stoves or other products that produce heat.

CAUTION: THIS PRODUCT HAS NO USER-SERVICEABLE PARTS. REFER ALL SERVICING TO QUALIFIED PERSONNEL.

WARNING: TO PREVENT FIRE OR SHOCK HAZARD, DO NOT EXPOSE THIS EQUIPMENT TO RAIN OR MOISTURE

Contents

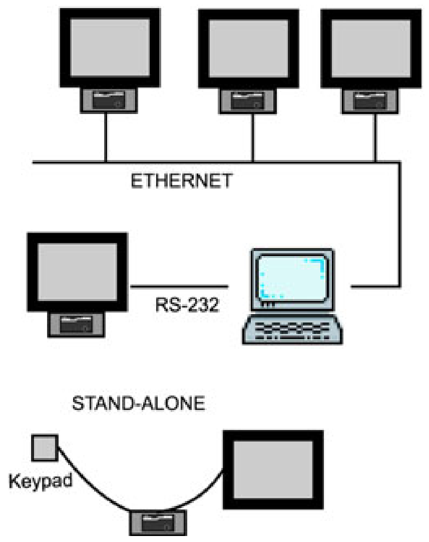
Chapter 1: Introduction

System overview

The *DVP* is a versatile playback system for MPEG digital video. It is suited for all applications requiring high quality pre-recorded video presentations. Among many other applications one finds video in theme parks, sport stadiums, museums, exhibitions, announcements in public buildings and commercial advertising in shops.
The reproduction of pictures and sound meets the highest quality levels for professional use. The decoded video is displayed on a TV (PAL or NTSC).
The DVP system uses *CompactFlash compliant cards (CF or IBM Microdrive)* as default storage medium for the compressed video and associated audio. As the playback does not require any moving part, there is no degradation of the sound or video quality. Even after many playbacks the video and audio messages still have the same quality as the original recording. This also ensures that the DVP system can be used almost anywhere. It is not affected by dust, humidity, temperature shifts nor by vibrations. As a result, the maintenance costs are very low.
Nowadays, one single CompactFlash Card can easily contain more then 10 minutes of very high quality video clips. For those applications, the DVP can be equipped with an *internal IDE hard disk drive*, thereby extending the capacity beyond several hours of very high quality programmes.

Remote configuration and control

The DVPs are equipped with several *communication ports* intended for *remote configuration* and *control*.
The *RS-232 port* allows one manager to control a single DVP per port. The standard *Ethernet interface* allows a single manager to control a virtually unlimited number of DVPs. In the two later cases, the manager can selectively address each DVP or *broadcast* the commands to all (or some) of them.
Thanks to its high degree of connectivity, the DVP can easily be integrated in complex infrastructures combining the use of different types of devices (audio/video, light/ pyrotechnic shows,...). In these cases, the DVPs are under control of remote managers via one of these communication ports.



Stand-alone operation

Should the DVP be used in stand alone mode, the user can still (re)configure the player by means of a standard *PS/2 keyboard*. The user browses through configuration menus that are displayed on the same TV as the decoded video (*OSD: On Screen Display*).
The DVP can autonomously execute complex scenarios defined by the statements of the so-called *script files* (readable text files generated by the user using a standard text editor). The execution of scripts can be synchronized with the activation of up to 8 input signals (*input triggers*). Conversely, the DVP controls 4 output signals (*output triggers*).

Generating and distributing the video/audio content

The DVP is a *playback-only* device. The video clips can be MPEG-encoded on any MPEG-compliant video encoder system.
The resulting file can be distributed/downloaded in different ways:
Direct download on CompactFlash Cards at a *distribution centre* (using a PC equipped with a standard *CompactFlash drive*), the cards being distributed by a *carrier* to the end-users.
A *carrier* distributes *intermediate copies* on CD-ROM to the end-user. The download on the CompactFlash Card is done locally by the user, using the CompactFlash Card drive of the DVP. The file is transferred through one of the communication ports (ideally Ethernet).
Instead of using CDs, the file is sent via a *network* (VPN or public Internet via ADSL, Cable, ISDN or POTS) to each end-user who will afterwards proceed to the download as explained here above.
As the DVP uses the *FAT16/FAT32 file system*, the download of clips is achieved by means of *file copy procedures* that are familiar to almost all of the users.
What has been explained above about the video/audio material also applies to software upgrades, downloading scripts, graphic overlays, …

The video and audio outputs

The decoded interlaced video is available in CVBS (composite) and Y/C (S-Video).
Optionally, the DVP may also be equipped with a VGA/SVGA Output.
For the audio component, the DVP has an *analog stereo line output*.

Graphic overlay

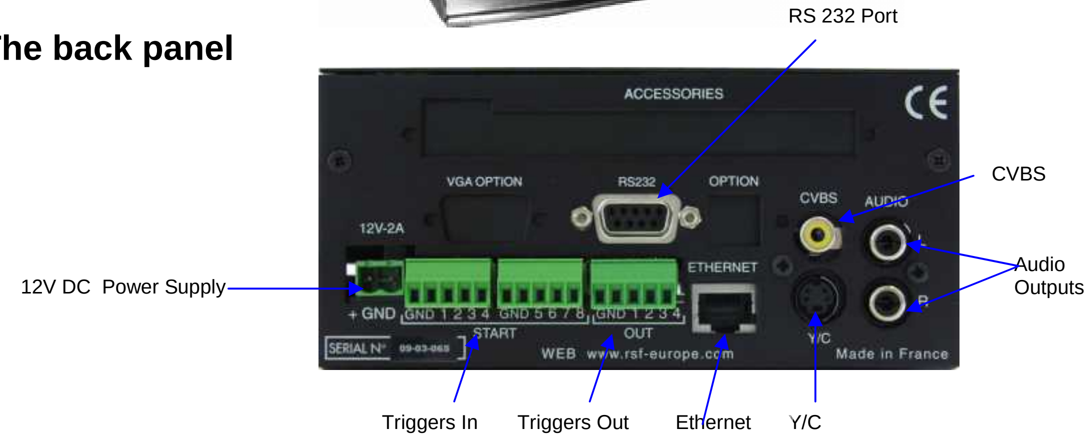
The DVP permits to overwrite the video with user defined *text* and *graphics* (e.g. a logo).

Chapter 2: Connections

This section takes a closer look at the elements located on the front and the back panels of the DVP.

The front panel

The back panel



Triggers Output/Input

Triggers Out

Output type: open collector (35 mA maximum) + internal 10 kOhm pull-up to +5V.

Triggers in

Input triggers are active low
TTL levels compatible
Internal 10 kOhm pull-up to +5V

VGA Out

High Density Female SUBD15 connector

Ethernet

RJ-45 plug

RS-232

The pinning for the RS-232 connector is:

Pin	Function
1	DCD
2	RX
3	TX
4	DTR
5	GND
6	DSR
7	RTS
8	CTS
9	RI

Chapter 3: Technical specifications

Video/audio content

Video/Audio formats

Elementary video and audio streams (MP@ML)
MPEG-1 system streams
MPEG-2 program streams (MP@ML)
DVD (.VOB) streams
Maximum *video bit rate*: up to 12 Mbps guaranteed from HDD, 8 Mbps or more from flash memory devices or a Microdrive
Audio *sampling frequencies*: 32 kHz, 44.1 kHz and 48 kHz

Control modes & interfaces

Stand-alone mode: execution of scripts synchronized on timer events or on external triggers, configuration by means of a *PS/2 keyboard* (menus displayed in OSD).
Remote control: via serial communication port (RS-232(DB9)) or via Ethernet (10/100base T Ethernet (RJ-45)).

CompactFlash Card interface

Supported Solid State devices: CompactFlash cards & IBM MicroDriveTM cards
Push button for *card extraction*.

Video output interfaces

CVBS Composite Video (PAL/NTSC or Cinch/RCA)
Y/C (4 pin DIN)

Audio output interfaces

Analog stereo line output (dual Cinch/RCA)

Miscellaneous

Input triggers: 8 inputs, TTL compatible with internal pull-ups
Output triggers: 4 outputs, TTL and *Open Collector* compatible
Optional internal IDE type of 3.5” Hard Disk Drive (5400 rpm drives recommended)
Optional removable IDE type of 2.5” Hard Disk
Optional VGA/S-VGA compatible video output on high density DB15 connector
LED indicators: power & ethernet link integrity
Power supply: 12 Volt DC, autoranging external power adapter 100-250VAC 50-60Hz 25W included in the package
Operating conditions:
Max. outside ambient temperature (limited by the hard drive specification): around 40°C (with a hard drive installed)
Max. inside temperature (limited by the hard drive specification): 55°C
Approvals: CE, UL
Trademarks: IBM MicrodriveTM

Chapter 4: Hardware Configurations

Preparing removable and non-removable drives

When preparing a hard disk or other storage medium, you must take several things into account before being able to get the DVP to work. These include jumper settings, harddisk/ compact flash configurations and volume labels.

Hard Disk / Compact Flash configurations

This section describes how to configure storage devices on the different types of DVP units.

DVP

The DVP has only one IDE controller

Configuration	J3 setting	Remarks
Bootable CF(*) present (single bootable MASTER)	Closed	The CF(*) holds OS, SSVP application and content (clips, scripts and images).

Volume Labels

The volume labels determine the name of the drive partitions when working with the DVP (i.e. the DOS naming convention 'C', 'D','E', …).
The first "SSVPNRx" series is used for non-removable drive partitions and the second "SSVPRx" for removable drives partitions.
The drives are named according to the following order:

1. the SSVPNR0 is always called 'C',
2. the DVP searches for SSVPNR1 (- if it exists it will be called 'D'),
3. the DVP will search for SSVPNR2, …

Chapter 5: Getting started

The goal of this section is to let the user familiarize himself with his DVP without the need for an external control terminal, provided your DVP is equipped with an internal hard disk drive.

For testing your DVP, proceed as follows:

1. Unpack your DVP.
2. Connect a line cord.
3. Connect a TV monitor either to the CVBS output or to the Y/C output of the DVP.
4. Switch on the DVP.
5. After about 10 seconds it will start playing a clip with colored bars stored on your hard disk drive.

Practically, at power-up the system checks which script has been selected by the user as the one that must be performed each time the DVP is restarted. This script is called the *start-up script* (in this case the script *getting.svp*).

Connect a PS/2 keyboard to the DVP and press [Enter] twice within a second. A menu will appear in overlay on your TV monitor. With the <UP> and <DOWN> keys, highlight the line *system settings* and press the <ENTER> key. The *settings* menu is now displayed. In the same way as above enter the *start-up script* sub-menu. The line *getting.svp* is highlighted and flagged with the > character, indicating that the *getting* script is currently the *start-up script*.

Highlight the *alive* script and press <ENTER>. Then press the <ESC> key three times and finally press <ENTER> while *yes* is highlighted. Doing so, you have selected the script *alive.svp* as *start-up script*.

Switch off and switch on your DVP. It will play now a single clip (*alive.mpg*) in *loop mode*.

As an exercise, re-install the *getting* script as *start-up* script.

You are now invited to browse through the different menus to discover the different settings that can be selected. The purpose of these settings is extensively described further in this document.

Disclaimer!

RSF recommends to use only the supplied AC to 12V DC power adapter and will in no respect be responsible for any damage to the DVP(R) unit, caused by improper application of alternate power supplies or AC to DC power adapters. The DVP(R) requires a stable 12V DC (+-5%) / 2.5A power supply in order to work properly with almost any brand and/or type of IDE hard disk drive.

Chapter 6: The OSD menu

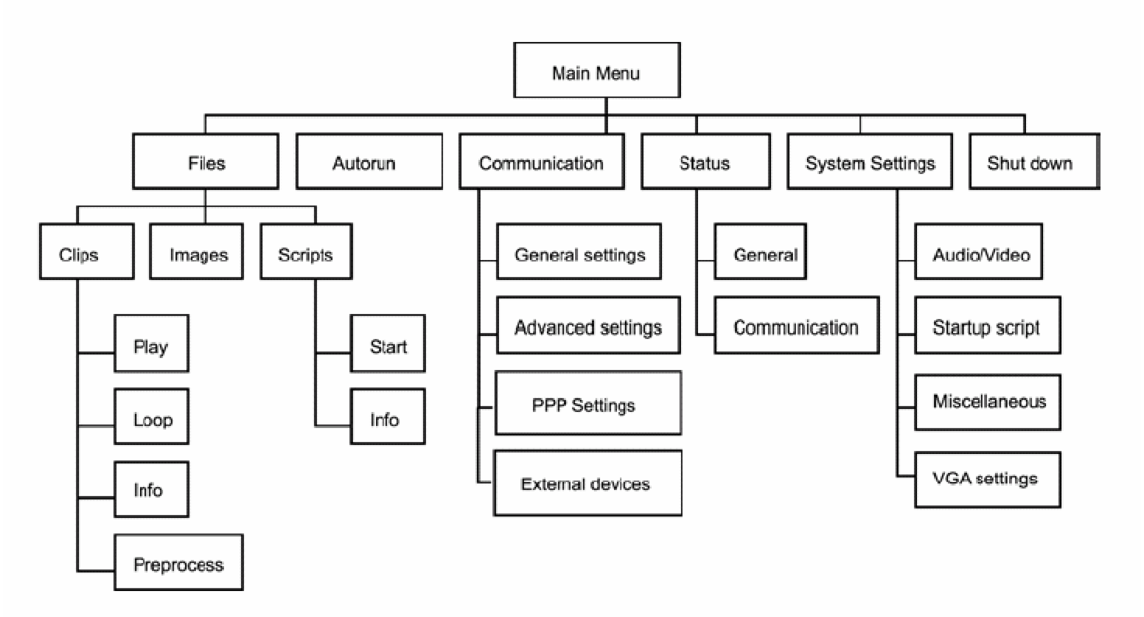
This chapter describes how to configure a DVP locally.

Introduction

During a local update session, the values of the configuration parameters are displayed in overlay mode (OSD: On Screen Display) on the TV set used to display the decoded video.
An update session starts by entering the *Main Menu*, i.e. either as soon as the <ENTER> key of the *PS/2 keyboard* is pressed twice within a second.
All the parameter settings can be modified using the *PS/2 alphanumeric keyboard*.

Note
It might be that in order to access the submenu's Communication and System Settings a password needs to be entered. This password is the same as the FTP password.

Hierarchical structure of OSD menu's



General remarks

1. To go one step forward in the tree of menus, browse (by means of the <UP> and <DOWN> keys) through the different options and press the <ENTER> key.
2. To go one step backwards in the tree of menus press the <ESC> key. Pressing the <ESC> key at the level of the *Main Menu* will terminate the update session. At some level the user may be invited to confirm whether he wants to save the new settings or not. To do so, highlight your choice and press the <ENTER> key.
3. To modify the value of parameters, highlight the name of the one that must be updated (using the <UP> and <DOWN> keys) and:

- ❑ For those parameters whose value must be chosen among a limited set of predefined values, all of the potential settings are displayed sequentially each time the <ENTER> key is pressed. When the right one is selected, either jump to another parameter (<UP> and <DOWN>) or press the <ESC> key to leave the current submenu.
- ❑ For those parameters whose update requires the entry of an alphanumerical string, enter the new character string, then jump to another parameter (<UP> and <DOWN>) or press then <ESC> key to leave the current submenu.

4. The selections that are not relevant (depending on the options the DVP is equipped with) are displayed in light grey.
5. An implicit exit without saving of the eventual new settings (and the deactivation of the OSD) is assumed when no keys have been pressed for more than one minute.
6. When leaving the *Main Menu*, the system checks whether any configuration parameter was modified or not. If so, the user is asked to confirm if the new settings may be saved, overwriting the current ones.
7. The DVP supports long filenames, but it is recommended to restrict their length to 25 character.

Main Menu

1. *Files*: the files menu allows you to see the contents of your DVP and to start playing clips and scripts.
2. *Autorun*: select a script to be executed at start-up.
3. *Communication*: various communication settings.
4. *Status*: DVP configuration overview.
5. *System Settings*: DVP configuration.
6. *Shut Down*: stop the DVP.

Files

1. Clips
2. Images
3. Scripts

Clips

The list of the available video clips (filenames) is displayed. The user can point out a filename (<UP> and <DOWN> keys). Pressing the <ENTER> key will jump to the following options: Play, Loop, Info, Preprocess. Simply press enter on 'Play' to play the clip, press enter on 'Loop' to play the clip in an endless loop.
Preprocessing generates some extra information about the clip that is used with certain script commands (e.g. TMIn, TMOut, TCIn, TCOut, …) in scripts. A warning appears on the screen to indicate the DVP is busy.
Note that preprocessing is limited to clips with less than 30 min playtime.
Selecting and pressing enter on the info menu reproduces extensive information about the clip.

Images

This menu allows to list the filenames of the files containing graphics, logos, pictures, texts, … which can be displayed in overlay over the video. The list of the available overlay files (filenames) is displayed (see the example hereafter).

Scripts

The list of the available script files (filenames) is displayed (see the example hereafter). The user can point out a filename (<UP> and <DOWN> keys). Pressing the <ENTER> key will jump to a new menu containing the items “Info” and “Start”.
Also the start-up script will be marked with the ”>” symbol in the list. Note that a start-up script isn’t mandatory.
When pressing “Info”, a dialog appears, describing the purpose of the script. This information can be entered in the first nine lines of your scripts in comment (see section script language). Therefore, it can be useful to use comments in the most effective way.
The “Start” item will run the selected script.

Autorun

Due to the “Autorun” feature it is possible to transfer files between the removable and the non-removable drives (and vice-versa). This can be used, for example, to download new clips or overlay graphics from an CompactFlash Card to the internal HDD. It can also be used to perform upgrades of the application software running on the DVP. Finally it can be used to retrieve files located on the internal HDD. Practically, the transfers that must be carried out are defined by the content of the so-called *autorun.svp* script. That file must be located in the \DVP\SCRIPTS subdirectory of the removable drive. This script file takes care of all the required transfers without any intervention of the user.
The following settings are possible:

- ❑ *Autorun script* (Enabled/Disabled): when set to Enabled, the system will automatically start the transfer procedure at the next power-up of the DVP.
- ❑ *Manual start* (No/Yes): when set to Yes, the transfer procedure is initiated when leaving the configuration menus.

Communication

General Settings (TCP/IP)

- ❑ *Protocol (TCP/IP | CDMP)*: here you can choose the communication protocol to use.
- ❑ *Interface (Ethernet/SLIP/PPP)*: the connection interface you have chosen. Be aware that when SLIP or PPP is chosen, it uses the RS-232 port. In that case, the selected driver in *External Devices* will be ignored. Additional settings can be found in the *PPP Settings* menu.
- ❑ *IP address (dotted decimal)*: contact your system administrator for this information
- ❑ *Subnet mask (dotted decimal)*: contact your system administrator for this information
- ❑ *UDP server port (decimal)*: the UDP port on which the server listens to establish connections with DVPs.
- ❑ *UDP client port (decimal)*: the UDP port on which the DVP should listen to establish a connection with the server.
- ❑ *UDP message interval (sec.)(decimal)*
- ❑ *Socket keep-alive (Enable/Disable)*: sometimes, there is no data traffic between a server and a DVP for longer periods of time. To verify whether the physical connection is still there, the server will regularly send a message to the DVP, which the DVP will return to the server. In this way both will know that the connection still exists.
- ❑ *Router address (dotted decimal)*: contact your system administrator for this information
- ❑ *Ethernet speed (10/100/auto)*: this option lets you select the Ethernet speed.(10 Mbits/s-100Mbits/s). If you choose the option “AUTO”, the system shall determine the speed through handshaking.

- *SLIP speed (300/1200/2400/4800/9600/19200/38400/56000/57600/115200)*
- *SSC protocol (enabled/disabled):* See "Simple Serial Control" on page 173 for more information.
- *SSC speed (300/1200/2400/4800/9600/19200/38400/56000/57600/115200)*
- *SSC address (decimal 0..254)*

General Settings (CDMP)

- *Protocol (TCP/IP | CDMP)*: here you can choose the communication protocol to use.
- *CDMP speed (baud) (300/1200/2400/4800/9600/19200/38400)*
- *CDMP polling interval (sec.) (decimal)*: by means of this interval the running software will check on the devices to see if the connection is still there.
- *CDMP address (decimal 1..254)*

Advanced Settings

- *System Name*
- *Scheduling (Enabled/disabled)*: indicates that scheduling will be used for communication.
- *IDS Keep-Alive (Enabled/Disabled)*
- *IDS Keep-Alive Timeout (s):* This parameter is only valid if the *IDS keep-alive* is enabled. In this case, enter the correct keep-alive time-out. Its value ranges from 15 to 3600. In the case of Ethernet communication, a kernel-level timeout can be used to check if the kernel receives its any data from the server. However, it is still possible that the application becomes de-synchronized with the server application. To check for the presence of application-level data, this extra function can be enabled. This application-level keep-alive function ensures that if the application board does not receive data from its server for the specified time, the board will drop its connection with the server. (Note: in the case of CDMP communication, this is the only keep-alive timer (there is no kernel-level time-out). In this case, it should be enabled.
- *Idle Timeout (s)*: when the communication is idle for Idle Timeout seconds, and PPP with modem is used as interface, the modem connection will be closed. For all other interfaces, this timeout is not applicable.
- *DHCP (ENABLED/DISABLED)*
- *Server Connect (ENABLED/DISABLED)*: the DVP will connect automatically at start-up to the server. If this setting is enabled, then the following settings must be filled in.
- *Server Address (decimal or dotted decimal)*: the address of the server
- *Server connect timeout (sec.) (numerical)*: the DVP blocks every communication in order to try to establish a connection to the server during the value that has been set in the *Server connect timeout* setting.
- *Server connect interval (sec.) (numerical)*: if the DVP didn't succeed in establishing connection with the server, there will be retries after the value (in seconds) that has been set in this setting
- *TCP Server Port (numerical)*: the port on which the server listens to establish connection with the DVP
- *FTP Password*: the DVP can act as an FTP server in order to up- or download files. The FTP Password is used to protect this service. When connecting, enter *admin* as the username and the content of this field as the password. It is important to know that the OSD menu will never show the real password on the screen, it will only display a string of asterisks (****). Typing *none* as the password disables password checking. The default password is set to this value. *Note: If any FTP password other than "none" is set, some OSD submenu's (Communication and System Settings) are also protected by this password.*
- *FTP (enabled or disabled)*: this field enables or disables the FTP service of the DVP. When updating this setting, the DVP will automatically reboot to let the change have effect.

PPP Settings

- *PPP Speed (300/1200/2400/4800/9600/19200/38400/56000/57600/115200)*
- *User name*: user name used PAP logon.
- *Password*: password used PAP logon.
- *Modem (Enabled/Disabled)*: indicates whether or not a modem will be used for the PPP connection. If no modem is used, all of the following settings are not applicable.
- *Modem initialization*: initialization string sent to the modem at startup.
- *Modem retries*: the number of times to retry until a succesfull PPP connection is established.
- *Modem Number*: phone number used by the modem.
- *Alternate Number*: alternate phone number.

External Devices

It might be necessary to reset the DVP when changes in this menu are made!

- *External device (None/TouchScreen/Barcode reader/ Generic)*: select the device that will be connected to the RS-232 port. Make sure SSC or PPP is disabled, otherwise this setting will be ignored.
- *TouchScreen Driver (none, 3M, Elo Serial TS, Old 3M)*: This field is used to enable the driver that is used to support 3M MicroTouch Serial (RS-232) TouchScreen Controllers. Note: since the SSC protocol is using the same serial port it is mandatory to disable the SSC Protocol setting in the General Communications Settings menu. If not, SSC will get priority over the TouchScreen driver.
- *TouchScreen Log (enabled or disabled)*: When enabled, the unit will send TouchScreen co-ordinates (X,Y) over the network (when conncected) so that they become visible in the Reply log window on the DVP Server (+). This way, it is easy to determine the proper "button" co-ordinates to be used in a script featuring TouchScreen functions.
- Bar code reader (None, Serial): any bar code reader can be connected to the RS-232 port, as long as it is compatible with the Phoenix II. The configuration of the bar code reader must be done only once. Select the RS-232 interface with the following settings:
- Baudrate: 9600
- Databits: 8
- Parity: none
- Flowcontrol: hardware

  The bar code reader must send a <CR> to indicate the end of each code. It is obvious that the *External Device* should be set to *Bar code reader*!
- *Generic Serial Driver (Enabled, Disabled)*: enable this driver to use custom defined serial commands in scripts.
- *Generic Serial Speed (300, 1200, 2400, 4800, 9600, 19200, 38400, 56000, 57600, 115200)*

Status

General

- *Hardware (alphanumerical)*
- *Software (alphanumerical)*
- *Operating Mode (MASTER/SLAVE)*: defines whether the DVP is operated in master or in slave mode.
- *Free Space (Non-Removable) (xyMB)*: gives the free space on the internal non-removable hard disk drive.
- *Free Space (Removable)(xyMB)*: gives the free space on the external removable drive.
- *Free Memory (bytes)*: displays the available memory in bytes.
- *Input Triggers*: eight characters are used to indicate the status of the eight input triggers. Inactive inputs are represented by a dot. An active input trigger is represented by its index. Example: ..6..32. means that the input triggers 6, 3 and 2 are active, the other ones being inactive.
- *Output Triggers*: four characters are used to indicate the status of the four output triggers. Inactive outputs are represented by a dot. An active output trigger is represented

by its index. Example: .32. means that the output triggers 3 and 2 are active, the other ones being inactive.

Communication

- *Ethernet (yes/no)*: when set to YES, means that the system is equipped with the Ethernet option (set to NO otherwise).
- *Protocol (TCP/IP | CDMP | NONE)*: the protocol set to TCP/IP, means that the active communication port is the Ethernet interface or the RS-232. CDMP means that one of the serial communication ports (RS-232) is used for the remote control and configuration of the system. SSC means that the RS-232 interface can be used.

Attention!
The Status Menu consists of different OSD screens. To toggle between screens, simply press the up or down arrow.

- *Link integrity*: checks if there is a connection
- *Speed (10mbits Half Duplex/Full duplex | 100mbits Full Duplex)*: display's the actual Ethernet speed.
- *DHCP (ENABLED/DISABLED)*
- *IP address (dotted decimal)*: displays the actual IP address.
- *Subnetmask (dotted decimal)*: displays the actual subnetmask.
- *Router (dotted decimal)*: displays the actual router address.
- *DNS (ENABLED/DISABLED)*

The "Input/Output triggers", "Free Memory" and the "Link integrity" options will continuously
be *polled*. This means that the DVP will check for changes every second.

System Settings

1. Audio/Video
2. Startup Script
3. Miscellaneous
4. VGA Settings

Audio/Video

- *Video standard (PAL/NTSC)*: this option determines which video standard is effective at the power-up of the system.
- *Freeze Timeout (min.)*: this is the maximum time (in minutes) it is allowed for the DVP not to play a clip. The minimum and default setting is five minutes.
- *Freeze Unlock (Disabled/Reboot/Restart Script)*: the action to take when no clip has played for 'Freeze Timeout' minutes. The Freeze Unlock mechanism can act as a screen saver to prevent Plasma TVs burning in due to script errors, badly encoded clips, ...
- *Audio Volume*: the audio volume setting is a number ranging from *zero* (muted) to *10* (+3dB).
- *Audio Language*: the DVP supports 16 different audio streams mixed together with the video. Audio streams are typically used for dubbing a movie in different languages. When a non-existing audio stream is chosen, the audio will be muted. The default language is set to *zero*.
- *Audio Channel Mode (Stereo/LToL&R/RToL&R/Mono)*: it is possible to set the default audio output channel routing. This default setting may/can be overruled by some specific Script language commands. See "DVP Script language description" on page 37 for more information.

Start-up script

The list of the available script files (filenames) is displayed.
The user can point out a filename (<UP> and
<DOWN> keys). The new start-up script file is the one
that is highlighted when the <ENTER> key is pressed.
This script will be started the next time the DVP
is powered on. In the same way you can disable the
start-up scripts. Just highlight the script preceded by the ">" flag and press on <ENTER>.
The ">" flag will disappear.

Miscellaneous

- *Mini Keyboard (Disabled | OSD Menu | Volume/Backlight | Custom)*: when set to OSD Menu, the front buttons on the DVP can be used to access the OSD menu. Alternatively, the Mini keyboard can be used to change the audio volume and the backlight level. It is even possible to program a custom functionality in *DVP.ini*. Below is an example:
- *Keyboard Layout (AZERTY/QWERTY)*
- *Trigger Mode (input) (FAST/MEDIUM/SLOW)*: when the input triggers are activated by means of electromechanical devices (switches, relays, …), the contact will likely rebound for a given lapse of time that is specific to each control device. As long as the control device rebounds, the DVP will see not pertinent transients on these inputs. The DVP filters out these undesirable transients (debouncing). The user can select which of the three following debouncing filters is the best suited to the control device he is using.
  Fast mode: suited when the input trigger is stable again within 10 ms following any change of state. The input trigger must then remain stable at least 50 ms.
- *Medium mode*: suited when the input trigger is stable again within 50 ms following any change of state. The input trigger must then remain stable at least 75 ms.
- *Slow mode*: suited when the input trigger is stable again within 100 ms following any change of state. The input trigger must then remain stable at least 100 ms.
  The default mode is *Fast mode*.
- *System Logging (Enabled/Disabled)*: reserved to the support. Should always be disabled in normal use.
- *User Logging*: this feature is used to track activity of a DVP. Currently, the DVP keeps track of the most essential commands. The OSD menu offers three possibilities for user logging:
- *Disabled*: no log file is created
- *Enabled (NR0)*: the log file resides in the \DVP\LOGS directory on the first partition of the non-removable drive.
- *Enabled (R0)*: the log file resides in the \DVP\LOGS directory of the first partition of the removable drive.
  The logs are created on a daily base and are stored in the "\DVP\LOGS" directory. When the DVP is heavily used, this directory will grow quite rapidly in size. For this reason, it is safe to keep the logging turned off, unless it is needed.
- *Language (EN/FR/NL)*: selects the language of the OSD menu
- *Backlight (Enabled/Disabled)*
- *Backlight Level*: the backlight level parameter controls the backlight luminance of TFT displays. This option is only available on the DVP.
- *Boot Job*: a boot job is an operation that will be executed the next time that the DVP is restarted. It is mandatory to restart by means of the Shut Down option from the Main OSD menu. The logic behind this is that some essential operations need to be completed first. The supported boot jobs are:
- *CR0-Check*: checks the integrity of the *removable disk* (R0).
- *FR0-Format*: deletes all files and directories on the *removable disk* and installs the *DVP directory structure*. Note: it is mandatory that the removable disk is present when the unit reboots and even recommendable to have the removable disk already present at the initial boot.
- *DUB-Copy System*: this boot job is especially designed to create bootable storage devices that can be used in other units, mainly in DVP players (pure solid state CF only operation) but also occasionally in DVP players (e.g.: internal bootable CF Card). It deletes all files and directories on the CF disk present in the CF slot (hidden behind the a small metal front plate) and installs all system files, drivers and the application program. The CF card can then be used as a bootable device on a DVP.

Note: it is mandatory that the CF disk is present when the unit reboots and even recommendable to have the CF disk already present at the initial boot.

- *NOP-Normal Op.*: Normal operating mode (no boot job).

Warning

It is not recommended for end-users to enter this mode, since it requires specific keyboard actions to progress through the test sequence.

- *Summertime*: the DVP has the capability to switch automatically between summer- and wintertime but some information depending on your location has to be entered first. This can be done in this menu. The summertime-value has to be in the following format:

Where:

- w: weekday (0…6, 0 is Sunday)
- d: day (1…5 i.e. Nth weekday of the month, 5 equals "the last")
- mm: month (1…12). The month always takes 2 characters
- bias: (minutes) usually + 60

This structure (wdmm) is repeated twice. First for the 'normal' date (in the autumn) and a second time for the daylight date (spring).

For example: 0510-2203+60

Meaning: switch between summertime and wintertime of one hour ('+60') between the last ('5') Sunday ('0') of October ('10') and the second ('2') Tuesday ('2') of March ('03').

When the summertime specification has a wrong syntax, it will be ignored by the DVP and no summer/wintertime conversion will take place.

It is safe to reboot the DVP after changing the summertime settings as it may influence certain timers and scripts.

- *Date (DD:MM:YYYY)*
- *New time(..:..:..)*
- *Current time*

VGA Settings

- *Deinterlace Mode*
- Brightness Red (0-100): default value is 50
- Brightness Green (0-100): default value is 50
- Brightness Blue (0-100): default value is 50
- Contrast Red (0-100): default value is 50
- Contrast Green (0-100): default value is 50
- Contrast Blue (0-100): default value is 50
- Sharpness (0-8): default value is 6. Recommended values are 5 and 6
- Osd TimeCode Display (*Disabled, Bottom Right, Bottom Left, Top Left, Top Right*): position of the TimeCode Display.
- Osd TimeCode Colour (*Red, Green, Blue, Yellow, Cyan, Magenta, White, Black*): color of the TimeCode Display.
- Osd Transparency (*0%, 12.5%, 25%, 37.5%, 50%, 62.5%, 75%, 87.5%*): transparency level of the TimeCode Display and the OsdText objects.
- Osd TextSize (*Small, Medium*): Text size of the TimeCode Display and OsdText objects.
- Flip Horizontal (*Enabled, Disabled*): Mirrors the video image.
- Flip Vertical (*Enabled, Disabled*): Flips the video image around its horizontal axis.

Chapter 7: DVP Script language

Description

In this chapter you will learn the basics of using the DVP scripts

Introduction

The DVP can perform complex scenarios defined by the content of the so called *script files*.
The *script files* are *readable text files*.
This *script chapter* is divided into six parts:

- ❑ Classes
- ❑ Objects
- ❑ Functions
- ❑ Associate Logical Status
- ❑ Control Structures
- ❑ Examples

In order to understand scripting it is necessary to read the first three parts and put emphasis on the introduction of the third part as it explains the basic structure of a script.
This manual contains a lot of detailed information, yet not all of it is required to understand and use the scripts. So, more difficult topics can be skipped.
Important preliminary remark: The syntax is *case-sensitive*, meaning *clip* is not the same as *Clip*.

Classes

A *class* is a data type. These data types may be used to create objects. When you create, or in programming lingo *instantiate*, a class, you create an object.
Example:

```
Clip alive(“ALIVE.MPG”);                //INSTANTIATED object “alive”
                                        //from the “Clip” class.
```

Members:

BarCode
BarCodePlayListFile
Button
Channel
Clip
Date
Flag
Frame
Image
Input
OsdText
Output
PlayList
PlayListFile
SerialIn
SerialOut
Time
TimeCode
TimeInterval
Timer
Trigger

BarCode

Bar codes are defined as a string of characters. They can be used in the same way as *Input*, *Button* and *SerialIn*. To use BarCode objects, one must enable the *Bar Code Reader* in the Osd Menu (*Communication Settings* > *External Devices*). Note that the *Simple Serial Control Protocol* must be disabled. Also, it is impossible to use PPP in combination with any of the other external devices, like the Bar Code Reader.

Syntax:

```
      BarCode code("<string>");
```

Examples:

```
//////////////////////////////////////////////////////////////////////
// DVP Example Script (RSF ) DVP SW version 01.06.XX
//
// This example shows the of a bar code reader.
////////////////////////////////////////////////////////////////////////
      BarCode code0("S7GN1CC187172");
      BarCode code1("9518426541592");
      BarCode code2("7519623247641");
      Clip clip1("fox8.mpg");
      Clip clip2("bmwz8.mpg");
      code0.Wait();
      while (true)
      {
                   if (code1)
                   {
                   clip1.Play();
                   }
                   if (code2)
                   {
                   clip2.Play();
                   }
      }
```

BarCodePlayListFile/BCPlayListFile

To ease the use of hundreds of bar codes. One can create BarCodePlayListFiles which take one or more filenames. Those files are much like ordinary playlist files. Each line contains a bar code followed by a comma and the name of a clip. Using the Wait&Play function the BCPlayListFile will wait until a bar code is scanned, look up the bar code and play the clip that is associated with it.

Syntax:

```
      BCPlayListFile playlist("barcodelist1.pvp", "barcodelist2.pvp");
```

Examples:

```
//////////////////////////////////////////////////////////////////////
// DVP Example Script (RSF ) DVP SW version 01.06.XX
//
// This example shows of a bar code reader in combination
// with a special playlistfile.
//////////////////////////////////////////////////////////////////////
      BarCodePlayListFile pl("ex_BCPlayList.pvp");
      while (true)
      {
                   pl.Wait&Play();
                   System.Log("Received bar code and started clip.");
      }
```

ex_BCPlayList.pvp:

```
      2498548321573, sun.mpg
      3496548321536, abba.mpg
      9498548329876, anim-ads.mp2
      5321548365436, stars.mpg
      1493578321951, product739.mpg
```

Button

The *Button* class associates a logical name to a physical area on a touchscreen. Such a physical area on a touchscreen will be composed by a certain number of elementary *grid zones*. The default grid size devides the screen in 16 (horizontal) by 16 (vertical) elementary touch-sensitive zones (256 in total). The default grid size can be overruled by using the *Grid* function. Each elementary zone is identified by its (X,Y) co-ordinates where the origin is the upper left corner of the touchscreen. The upper left corner zone's co-ordinates are (0,0), while the lower right corner elemenatry zone's co-ordinates are (15,15) when the default 16 by 16 grid is used. When defining a physical area on the touchscreen one can simply consider 1 or more elementary zones by its elementary co-ordinates, or "group" a number of elementary zones by indicating the upper left and lower right corner of the considered sub-area. The definition of a button to be used in a script is subsequently done by an enumeration of 1 ore more "sub-areas".

Syntax:

```
Button buttonname ("<sub-area1>;<sub-area2>;...;<sub-areaN>")
```

where a sub-area can be:

- ❑ an elemetary zone defined by its X,Y co-ordinates
- ❑ a grouped zone defined by the X,Y co-ordinates of its upper left and lower right corners

Examples:

```
Button Menu1 ("2,3");
Button StartClip23 ("4,4-7,6");
Button FourCornersAndCenter ("0,0;15,0;0,15;15,15;7,7-8,8");
Button FullScreen ("0,0-15,15");
```

Clip

The purpose of the *Clip* class is to associate a *logical name* to a given MPEG file.

Syntax:

```
Clip clipname (“VIDEO_filename”);
Clip clipname (“VIDEO_filename”,<duration>);
```

Examples:

```
Clip alive(“alive.MPG”); //associate the logical name
//“alive” to the clip
//“alive.MPG”
Clip letstick(“letstick.MPG”); //associate the logical name
//“letstick” to the clip
//letstick.MPG”
Clip lord(“lord.VOB”,20); //associate the logical name
//“lord” to the clip
//”lord.VOB” and if playback,
//playback for 20 seconds
```

Remark:

A clip can be instantiated in two ways (see *syntax*). In the latter case the parameter `<duration>` stands for a duration limit of the clip. If for example `<duration> = 20`, then the playback of the clip stops after 20 seconds.

The `VIDEO_filename` does not contain any indication about the actual location of the file, but just the file name and its extension.

Date

The *Date* class associates a *logical name* to a date. This way it becomes possible for a script to be driven by the internal *real-time clock* of the DVP.

Syntax:

```
Date datename (DD, MM, YYYY);
```

Examples:

```
Date Christmas (25,12,*); //associate the logical name
//“Christmas”
//to the 25 december of every
//year
Date Date2 (20,2,2001);
```

Remark:

DD, MM and YYYY are numbers or can be a *wildcard*, which is represented by the * character.

Flag

The *Flag* class associates a logical name to a *boolean* variable. A flag object can thus be *True* or *False*. The default state of a flag object when defined in a script is *False*. The use of boolean variables in scripts can be very helpful in creating some more complex scripts, e.g. scripts that allow the user to navigate through a multi-level menu structure by using *pushbuttons* or *touchscreen buttons*. It is even possible to store the status of a flag on disk and recall it later in the script or when the script is restarted.

Syntax:

```
Flag flagname ();
```

Examples:

```
Flag ExitToMain();
Flag Button1Pressed();
```

Frame

The *Frame* class associates a logical name to a given framenumber and clip. This way one can define framenumber based *markers* which can be used to start an event when the playback of a clip passes a given framenumber.

Syntax:

```
Frame markername (ClipObjectName,FFFFF); where FFFFF stands
for a (max.) 5 digit
framenumber
```

Examples:

```
Clip Alive ("alive.mpg");
Frame Marker1 (Alive,200);
Frame Marker2 (Alive,10112);
Frame Marker3 (Alive,00777);
```

Image

The *Image* class allows to show *overlay graphics*.

Syntax:

```
Image Name (“filename.ext”, xpos, ypos, width, height);
```

Example:

```
Image Logo (“logo.gif”, 1, 60, 70, 80);
```

This statement declares an entity named *Logo* whose content is given by the *logo.gif* file. Only the 70 first pixels (starting from the left) of the first 80 lines (starting from top) will be displayed. The upper line of the displayed area is the 60th line on the screen. The displayed
area is *flushed* left on the screen (xpos=1).

Remark:

This function will associate a file to a *logical name* together with parameters like the *position* on the screen of a *reference point* of the image (its upper left corner) and the *dimensions* of the part of the image that is actually to be displayed. These parameters are expressed in pixels. `(xpos,ypos) = (0,0)` corresponds to the upper left corner of the screen, `xpos` dealing with the *horizontal* position and `ypos` with the *vertical* one. The parameters `width` and `height` are determining which part of the image will effectively be shown. If `(width, height) = (0,0)` the image will be shown entirely. If you specify a width or a height bigger than the actual dimension, the whole image will be centered in the specified window.

It is possible to overlay graphics (e.g. logo’s) with the video output provided by the MPEG decoder through the usage of dedicated functions that will be described further in this manual. Note that the use of the *GIF format* is mandatory.

Input

An object derived from the “Input” class will be set to “TRUE” if the inputs match a given mask.

Syntax:

```
Input object (<state_input_1>, <state_input_2>, …);
```

Example:

```
Input myinput (1, 0, *, 1);
```

This object will become true if Input1 and Input4 is set, and Input2 is unset. The *asterix* (‘*’) indicates a ‘*don’t care*’ state.

OsdText

The OsdText class is used to define text that can be displayed on screen. The coordinates indicate the position of the first character of the text. In this case this will be row 14, column 0. The position 0, 0 corresponds to the upper left corner of the screen. For more information have a look at the Osd object and the extra information on the *Macronix Osd*.
The following functions apply to OsdText.

Show
Clear
Scroll

Syntax:

```
OsdText mytext(14, // Row, counting from 0.
0, // Column, counting from 0.
"my text", // The text you want to display.
0, // The index of the foreground color.
```

```
      0, // The index of the background color.
      False, // Blinking text?
      True, // Transparent background?
      True); // Draw a border around each character?
```

Examples:

```
////////////////////////////////////////////////////////////////////////
// DVP Example Script (RSF ) DVP SW version 01.06.XX
//
// This example demonstrates the use of Osd Text. (dLite only)
////////////////////////////////////////////////////////////////////////
      Osd.Clear();
      Osd.TextSize(1);
      Osd.Transparency(3);
      Osd.Color(0, 255, 255, 255);// white
      Osd.Color(1, 255, 0, 0);// red
      Osd.Color(2, 0, 255, 0);// green
      Osd.Color(3, 0, 0, 255);// blue
      Osd.Color(4, 255, 255, 0);// yellow
      Osd.Color(5, 255, 0, 255);// cyan
      Osd.Color(6, 0, 255, 255);// magenta
      Osd.BorderColor(100, 100, 100);// grey
      OsdText Capitals(6, 0, "ABCDEFGHIJKLMNOPQRSTUVWXYZ", 0, 1, True, False,
      False);
      OsdText Small (7, 0, "abcdefghijklmnopqrstuvwxyz", 4, 6, False, False,
      True);
      OsdText Numbers (8, 0, "012345678901234567890123456789", 7, 5, False,
      False, True);
      // Not all special characters can be displayed.
      OsdText Others (9, 5, "[{}]*+&(§!)-,;:?./");
      OsdText TopLeft (0, 0, "TopLeft", 0, 0, False, True, True);
      OsdText TopRight (0, 22, "TopRight", 4, 6, False, False, True);
      OsdText BottomLeft (14, 0, "BottomLeft", 0, 0, False, True, True);
      OsdText BottomRight (14, 19, "BottomRight", 7, 5, False, False, True);
      Timer Delay(20);
      TopLeft.Show();
      TopRight.Show();
      BottomLeft.Show();
      BottomRight.Show();
      Delay.Start();
      Delay.Wait();
      TopLeft.Clear();
      TopRight.Clear();
      BottomLeft.Clear();
      BottomRight.Clear();
      Capitals.Show();
      Small.Show();
      Numbers.Show();
      Others.Show();
      Delay.Start();
      Delay.Wait();
      Osd.Clear();
```

Here is another example:

```
////////////////////////////////////////////////////////////////////////
// DVP Example Script (RSF ) DVP SW version 01.06.XX
//
// This example shows how to scroll a text using the Macronix
// Osd. (dLite only)
////////////////////////////////////////////////////////////////////////
      Osd.TextSize(1);
      Osd.Transparency(3);
      Osd.Colour(0, 255, 255, 255); // white
      Osd.Colour(1, 255, 0, 0); // red
      Osd.Colour(2, 0, 255, 0); // green
      Osd.Colour(3, 0, 0, 255); // blue
      Osd.BorderColour(100, 100, 100);// grey
      OsdText Short (0, 0, "Scroll Text - ", 0, 1, false, false, true);
      OsdText Long (6, 0, "Scroll Text, which is too long to fit on one
      screen. - ", 2, 3, false,
      true, true);
      Short.Scroll();
      Long.Scroll(100);
      Timer delay(30);
      delay.Start();
      delay.Wait();
      Short.Clear();
      Long.Clear();
      Osd.Clear();
```

Output

An object derived from the *Output* class will be set to *True* if the outputs match a given mask.

Syntax:

```
Output objectname (<state_Output1>,<state_Output2, ...);
```

Examples:

```
Output MyOutput (1,0,*,1);
```

MyOutput will become true if Input1 and Input4 are set and Input2 is unset. The asterisk ('*') indicates a "don't care" state. A MyOutput.Set(); command will obviously set MyOutput in the true state.

PlayList

The purpose of the *PlayList* class is to associate a *logical name* to a *sequence* of clips.

Syntax:

```
PlayList listname (clipname1, clipname2, clipname3, …, clipnameN);
PlayList listname (VIDEO_filename1, …, VIDEO_filenameN);
```

Examples:

```
PlayList Hitparade (alive,letstick);
```

Associate the clips “alive” and “letstick” to the playlist “Hitparade”

```
PlayList Hitparade (“alive.MPG”,”letstick.MPG”);
```

Associate the video files “alive.MPG” and “letstick.MPG” to the playlist “Hitparade”.

Remark:

If a clip has been associated using the `<duration>` parameter, this parameter will be ignored when the clip is used in a playlist

PlayListFile

The purpose of the *PlayListFile* class is to associate a *logical name* to a *sequence* of clips found in one or more *playlist-file*. Such a playlist-file is a readable textfile with extension ‘.PVP’. It is recommended to use the *DVP Script Editor*, provided with the DVP Server package to edit such a playlist-file. Playlist-files can be modified without stopping the script. PlayListFiles can also be defined with *pauses* between two consecutive clips. Therefore, in the file, one must add a comma behind each clip and the duration of the pause in seconds. Use 0 for no pause. If a clip is not followed by a pause, the playlist will halt the playback.

Syntax:

```
PlayListFile listname (“name1.pvp”,”name2.pvp”,...,”nameN.pvp”);
```

Example:

```
PlayListFile Hitparade (“music.pvp”);
```

Where “music.pvp” could look like this:

Alive.mpg, 5

Lord.vob, 3

Dancing.mpg, 10

Example 2:

```
PlayListFile Week (“monday.pvp”,...,”saturday.pvp”);
```

Remark:

If you download playlist-files to the DVP, these files must be placed in the folder

SerialIn

Using *SerialIn*, the script can react to a custom defined serial command received on the RS-232 port. The *Generic Serial Driver* must be enabled and set to the correct *baudrate*. Serial commands are defined as *ASCII text*. To allow the use of special characters, escape sequences [see note on escape sequences] can be mixed with other characters. All serial commands are assumed to end with a special *End-Of-Text* character or a period of *silence*. This special character must not be used in the definition of the SerialIn code itself. If *ReplyOnChar* is used (see *SerialDriver*), the *Generic Serial Driver* will respond with the *reply character* for each character received.

Syntax:

```
SerialIn incode(“<code in>”);
```

Example:
```
//////////////////////////////////////////////////////////////////////
// DVP Example Script (RSF ) DVP SW version 01.06.XX
//
// This example demonstrates the use of custom defined serial
// commands by means of the Generic Serial Driver.
//////////////////////////////////////////////////////////////////////
      Clip clip("fox8.mpg");
      SerialIn Code3("Send");
      SerialOut Code4("Reply\r");
      SerialOut Code5("Reply\n");
      SerialOut Code6("Rep\\ly\r");
      SerialOut Code7("\x20Reply\r");
      SerialOut Code8("\X30Reply\r");
      SerialDriver.ReplyOnChar(true);
      //SerialDriver.SetReplyChar("\x06"); // default
      SerialDriver.SetTimeout(1000);
      //SerialDriver.SetEndOfText("\x0D"); // default
      SerialDriver.EndOfText(false);
      SerialDriver.Log(true);
      while(true)
      {
      Code3.Wait();
      clip.Play();
      Code4.Send();
      Code5.Send();
      Code6.Send();
      Code7.Send();
      Code8.Send();
      }
```

SerialOut

If the user needs to send serial commands a *SerialOut* object can be defined. Just like *SerialIn* it can contain *escape sequences*. The only function that applies is:

Syntax:
```
      SerialOut outcode(“>code out>”);
```
Example:
```
//////////////////////////////////////////////////////////////////////
// DVP Example Script (RSF ) DVP SW version 01.06.XX
//
// This example demonstrates the use of custom defined serial
// commands by means of the Generic Serial Driver.
//////////////////////////////////////////////////////////////////////
      Clip clip("fox8.mpg");
      SerialIn Code3("Send");
      SerialOut Code4("Reply\r");
      SerialOut Code5("Reply\n");
      SerialOut Code6("Rep\\ly\r");
      SerialOut Code7("\x20Reply\r");
      SerialOut Code8("\X30Reply\r");
      SerialDriver.ReplyOnChar(true);
      //SerialDriver.SetReplyChar("\x06"); // default
      SerialDriver.SetTimeout(1000);
      //SerialDriver.SetEndOfText("\x0D"); // default
      SerialDriver.EndOfText(false);
      SerialDriver.Log(true);
      while(true)
      {
      Code3.Wait();
      clip.Play();
      Code4.Send();
      Code5.Send();
      Code6.Send();
      Code7.Send();
      Code8.Send();
      }
```

Time

The purpose of the *Time* class is to associate a *logical name* to a *time-stamp*. This way it becomes possible for a script to be driven by the internal *real-time clock* of the DVP.

Syntax:
```
      Time timename(HH, MM, SS);
```
Examples:
```
      Time Noon (12,0,0); //associate the logical name
      //“Noon” to
      //12 o’clock, 0 minutes, 0 s
      Time Time2 (*,30,0);
```
Remark:

HH, MM and SS are numbers or can be a *wildcard*, which is represented by the *asterisk* ‘*’ character.

TimeCode

The *TimeCode* class associates a logical name to a given timecode value and clip. This way one can define timecode based *markers* which can be used to start an event when the playback of a clip passes a given framenumber.

Syntax:

```
TimeCode markername (ClipObjectName,HH,MM,SS,FF); where HH,MM,SS,FF
stand for hours, minutes,
seconds and frames.
```

Example:

```
Clip Alive ("alive.mpg");
Frame Marker1 (Alive,0,0,8,0);
Frame Marker2 (Alive,0,2,30,10);
```

TimeInterval

TimeInterval can be used to perform actions that relate to a specific period of the day. It can be used like Time and Date. The first three parameters are the starting hour, minute, and second of the interval. The last three parameters are ending time of the interval.

Syntax:

```
TimeInterval evening (H1,M1,S1,H2,M2,S2);
```

Example:

```
//////////////////////////////////////////////////////////////////////////
// DVP Example Script (RSF ) DVP SW version 01.06.XX
//
// This example demonstrates the use TimeInterval objects.
//////////////////////////////////////////////////////////////////////////
TimeInterval evening(17, 30, 0, 20, 30, 0);
TimeInterval late(20, 30, 0, 23, 00, 0);
TimeInterval night (23, 00, 0, 6, 00, 0);
Clip mill("mill.mpg");
Clip fox("fox8.mpg");
while (true)
{
while (evening)
{
if (Input1)
{
mill.Play();
}
}
while (late)
{
if (Input1)
{
fox.Play();
}
}
while (night)
{
if (Any)
{
Any.Stop();
}
}
}
```

Timer

The purpose of the *Timer* class is to associate a *logical name* to an *internal timer* and to specify after how many *seconds* this timer *expires*.

Syntax:

```
Timer timername (<duration>);
```

Examples:

```
Timer oneminute (60);
```

associate the logical name *oneminute* to a timer with a duration of *60* seconds

```
Timer tenseconds(10);
```

associate the logical name *tenseconds* to a timer with a duration of *10* seconds

Trigger

The *Trigger* class lets your script respond to a command on the *SSC*. If the appropriate trigger identification number is detected, then the *Trigger-variable* will be set to *TRUE*.

Syntax:

```
Trigger object (<trigger#>);
```

Example:

```
Trigger mytrigger (10);
```

This object will become true if the *SSC* command for *ID 10* is sent.

Remarks:

A trigger object must not be confused with *trigger inputs*. Trigger inputs are a set of parallel connectors in the DVP that correspond to the *Input* class. A trigger object, on the contrary, corresponds to an SSC command. *Trigger#* must be an *integer* value between *zero* and *999*. Once the trigger has been used in the script, its state is *reset* and it can only be set again with a new SSC command.

Objects

An object is a variable that has a certain set of *functions* that can act on it. There are two types of objects:

- *Constant* objects
- Objects that are *instantiated* from a class (see classes).

Example 1:

```
Audio.Language (0);
```

CONSTANT OBJECT. Sets the audio stream we're using. Audio is the object and language is the function.

Example 2:

```
Clip clip1 ("alive.mpg");
clip1.Play;
```

INSTANTIATED OBJECT. Play a clip.

Members:

- Any
- Audio
- Input1, Input2, Input3, Input4, Input5, Input6, Input7, Input8
- Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
- Output1, Output2, Output3, Output4
- OSD
- SerialDriver
- System
- Video

Every object that is instantiated from a *class*. The list of classes can be found in the previous section of this chapter.

Any

The *Any* object is a constant object. It is used when the current active clip is not known in the script

Example:

```
Any.Stop(); //Stops playback of the clip
//that's currently playing
```

Remarks:

The following functions can be called on the *Any* object:

- Stop
- Pause
- Resume

Audio

A constant object that allows you to change the behaviour of the audio.

Example:

```
Audio.Volume (0); //Mute
```

Remarks:

The following functions are supported on this object:

- ❑ Language
- ❑ Volume
- ❑ FromClip
- ❑ Stereo
- ❑ Mono
- ❑ LToL&R
- ❑ RToL&R

Days of the week

These entities represent the days of the week. This is another way of driving a script by the internal real-time clock of the DVP.

Note that the use of the days of the week statement is always combined with a *Boolean type control structure* (waitevent, while, if).

Syntax:

```
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

Example:

```
//This example will playback the clip "Welcome" on Mondays
if (Monday)
{
Clip Welcome ("HEYJUDE.MPG");
Welcome.Play();
}
```

Input1, Input2, …, Input8

These entities are the image of 8 dedicated input pins (also called *input triggers*). They are used to control the sequencing of the actions defined in the scripts.

Output1, Output2, …, Output4

These entities are the logical representation of the status of 4 dedicated output pins (also called *output triggers*).

OSD

This is an object that gives access to some of the settings of the *Macronix OnScreen Display*. Modifying these settings will also affect the appearance of the time code display. For more information see the OsdText class and the TimeCode Display. The following functions apply to the OSD object:

- ❑ TextSize
- ❑ Transparency
- ❑ Color/Colour
- ❑ BorderColor/BorderColour
- ❑ Clear

SerialDriver

To send/receive serial commands from/to a script, one can activate the *Generic Serial Driver* as *External Device* in the Osd Menu. *SerialOut* objects can send a custom defined command from the script to any device attached to the RS-232 port. To receive a command, a *SerialIn* object must be declared. The SerialDriver has two ways to detect the end of a command. The command ends with a specific 'End-Of-Text' character (see *Set-EndOfText* and *EndOfText*) or the command is followed by a period of *silence* (see *SetTimeout*). The Generic Serial Driver can reply on every character received with a so called *reply character* or acknowledge (see *SetReplyChar* and *ReplyOnChar*). Finally, there is a logging feature. For each command received by the Generic Serial Driver a message is sent to the *ReplyLog* window of the *DVP Server*. The following functions apply to SerialDriver.

- ❑ SetReplyChar
- ❑ ReplyOnChar
- ❑ SetTimeOut
- ❑ SetEndOfText
- ❑ EndOfText
- ❑ Log

System

A constant object that allows you to execute some system related functions. This object can also be used to send a message to the DVP.

Example:

```
System.Get ("script.svp"); //Format Flash disk
```

Remarks:

The following functions are supported on System:

- ❑ Put
- ❑ Get
- ❑ Log

Video

A constant object that allows you to change the behaviour of the video.

Example:

```
Video.On (); //Turn the video on
```

Remarks:

The following functions are supported on Video:

- ❑ On
- ❑ Off
- ❑ Black
- ❑ FromClip
- ❑ Grid
- ❑ ClearImages

Functions

Functions are the actual actions of a script as they can tell the DVP what to do. A function is always related to an object and will act on that object.

Example:

```
[1] // INSTANTIATE
[2] // Make 'alive' object from the 'Clip' class
[3] Clip alive("ALIVE.MPG");
[4]
[5] // FUNCTIONS
[6] // Execute 'Play' function on the 'alive' object
[7] alive.Play();
```

This example demonstrates the basic structure of a script: It is a script to start the playback of the 'ALIVE.MPG' clip.

Line 1 & 2: The first two line are comments. They only are of use to the reader and are ignored by the DVP.
Line 3 makes an object. Instantiate the 'alive' object from a clip (see previous chapter).
Line 4 – 6: Empty lines and comments are, again, ignored by the DVP.
Line 7: Perform an action on the alive object. The Play function will act on the 'alive' object. In consequence: the DVP will start playing the 'ALIVE.MPG' clip.

Escape sequences

The parameters of *SerialIn*, *SerialOut* and *SerialDriver.SetReplyChar()* can contain escape sequences (character(s) preceeded by a '\'). The following are supported:

- ❑ \r : a carriage return, ASCII value 13.
- ❑ \n: a linefeed, ASCII value 10.
- ❑ \\: a backslash. A single \ can not be used.
- ❑ \xHH: a hexnumber. E.g. \x0D is carriage return, \x20 is space, ...
- ❑ \XHH: same as \xHH.

Remarks:

1. It is good practice to start your script with the list of instantiations and after that with the functions.
2. Not all objects need to be instantiated. For example, to turn the video output black the following script will suffice:

```
// Turn the Video black
Video.Black();
```

It is not necessary to instantiate an object from a class like in the previous example as "Video" is a predefined (or 'constant') object.

Function overview

This section is an overview of which functions can be called on which class or object. A more detailed list of functions can be found in the following part.

Clip

ClearLB/ClearLetterBox *Clear letterbox video output mode*
FRIn/FrameIn *Set start of playback*
FROut/FrameOut *Set stopmarker*
Loop *Plays a clip repeatedly*
Pause *Pauses a clip*
Play *Plays a clip*
Prepare *Prepares a clip*
Prepare&Show *Prepares a clip and show first frame*
Resume *Resumes a clip after Pause*
Search *Locate framenumber (video paused & black)*
Search&Show *Locate framenumber (video paused)*
Search&Start *Locate framenumber (video starts playing)*
SetDefaults *Reset attributes*
SetLB/SetLetterBox *Set letterbox video output mode*
SetLoops *Set repeat-count*
Slow *Plays a clip in slow motion*
Start *Starts playback after Prepare*
Step *Steps N frames*
Stop *Stops playback*
TCIn/TimeCodeIn *Set start of playback*
TCOut/TimeCodeOut *Set stopmarker*
TMIn/TimeIn *Set start of playback*
TMOut/TimeOut *Set stopmarker*
Wait *Wait until clip ends*

PlayList and PlayListFile

First *Play first clip*
Last *Play last clip*
Loop *Loops through the list*
Next *Proceed to next clip*
Number *Play Nth clip*
Play *Plays the list*

Previous *Revert to previous clip*
Random *Randomly select a clip from a playlist object or a playlist file*
SeamlessModeOff *Don't avoid flicker during transition from one clip to the other*
SeamlessModeOn *Avoid flicker during transition from one clip to the other*
Stop *Stops playback*
TrackModeOff *Don't stop after each clip*
TrackModeOn *Stop after each clip*
Wait *Wait until list has finished*

OutputN

Clear *Clears output N*
Pulse *Set output trigger during a given period of time*
Set *Set output N*

Image

Clear *Clears the graphic overlay*
Show *Shows the graphic overlay*

Timer

Start *Start timer*
Stop *Stop timer*
Wait *Wait until timer ends*

Any

Pause *Pause playback*
Resume *Resumes playback after pause*
Stop *Stop playback*

System

Get *Copy file*
Put *Copy file*
Log *Send a message to the DVP and display it in the Reply log*

Input and Input1, Input2, …

Wait *Wait until hardware input is set*

Time

Wait *Wait until time matches real-time-clock*

Date

Wait *Wait until date matches real-time-clock*

Monday, Tuesday, … (Day of the week)

Wait *Wait for day*

Audio

Language *Selects audio stream*
LToL&R *Routes left clip channel to both audio outputs*
Mono *Mixes both audio channels to mono*
RToL&R *Routes right clip channel to both audio outputs*
Stereo *Routes stereo sound (when available) to both audio outputs*
Volume *Set volume*

Video

BacklightOff *OEM function (not applicable)*
BacklightOn *OEM function (not applicable)*
Black *Turn video output black*
Calibrate *Starts TouchScreen calibration process*
Grid *Set grid size*
Off *Disconnect video*
On *Connect video*

Flag

Clear *Clears a flag (boolean variable)*
Recall *Load status from disk*
Set *Sets a flag (boolean variable)*
Store *Save status to disk*

SerialDriver

EndOfText *Enables or disables the use of an end-of-text character*
Log *Enables or disables the logging of all received commands*
ReplyOnChar *Generic Serial Driver replies on each character received*
SetEndOfText *Sets the end-of-text character*
SetReplyChar *Sets the reply character*
SetTimeout *Sets the time-out*

OSD

TextSize *Define the text size of all Osd text*
Transparency *Set the transparency level of all OsdText and TimeCode display*
Color/Colour *Program the color palette (8 colors)*
BorderColor/ *Define the color of the border around each character*
Border Colour
Clear *Clear all text off the screen*

OsdText

Clear *Clear the Osd text off the screen*
Scroll *Scroll text from left to right*
Show *Show the Osd text*

SerialIn

Wait *Wait until the serial code is received*

SerialOut

Send *Sends a serial code to an external device*

TimeInterval

Wait *Wait until the current time falls within the interval*

BarCodePlayListFile/BCPlayListFile

Wait&Play *Wait until a barcode is received and play the corresponding clip*

All objects

Status *Shows class info in the Reply log*

Black

This function generates a *black screen* on the monitors connected to one of the *analog* outputs. Still the *video syncs* are active.

Applies to:
```
Video
```
Example:
```
Video.Black ();
```

BorderColor/BorderColour

Defines the color of the border around each character. It only has effect on the TimeCode Display and OsdText objects that have their last parameter set to true.

Applies to:
```
Osd
```
Syntax:
```
Osd.BorderColor(<Red value>,<Green value>,<Blue value>);
```

Clear

This function is intended to deactivate the selected trigger output (electrically, the inactive level is high) or to clear the selected image off the screen.

Applies to:
```
OutputN, Image, Flag, Osd, OsdText
```
Syntax:
```
outputname.Clear ();
imagename.Clear ();
Osd.Clear();
```
Example:
```
Output1.Clear (); //deactivates the output trigger “Output1”
Image logo (“logo.GIF”, 1, 60, 70, 80);
logo.Show ();
logo.Clear (); //Removes the image “logo” from the screen
```

ClearImages

This function erases all images.

Applies to:
```
Video
```
Syntax:
```
Video.ClearImages();
```

ClearLB/ClearLetterBox

This function is used to reset the video output from the letterbox converted format, in case the video output was previously set to this format by using the *SetLB* function.

Applies to:
```
Clip, Video
```
Syntax:
```
clipname.ClearLB ();
```
Example:
```
Clip BarTime ("TIMEG6E.M2V");
Bartime.SetLB (); // sets the video output to
the letterbox converted
format
BarTime.Play (); // BarTime will start playing
in letterbox format.
BarTime.Wait ();
BarTime.ClearLB (); // Resets the video output to
the normal full screen format
BarTime.Play (); // BarTime will now start
playing full screen.
```
Remarks:

When applied to *Video*, the setting affects all *Clips* that are not set individually. So to play all Clips in *16/9 format* except for clip2, call: Video.SetLB() and clip2.ClearLB().

Color/Colour

Used to program the 8 colors of the color palette. Note that color 7 is used for the Time-Code Display. Changing that color will have affect on the TimeCode Display, even when the script has already ended.

Applies to:
```
Osd
```
Syntax:
```
Ods.Color(<index>,<Red value>,<Green value>,<Blue value>);
```

EndOfText

Enables or disables to use of an End-of-text character to detect the end of a serial command.

Applies to:
```
SerialDriver
```
Syntax:
```
SerialDriver.EndOfText(<true or false>);
```

First

This function plays the first clip of a playlist object or a playlist-file.

Applies to:
```
PlayList, PlayListFile
```
Syntax:
```
object.First ();
```
Example:
```
PlayList List (Clip1, Clip2);
List.Number (2); //start playing the second clip
WaitEvent (Input1); //wait for Input1 to start playing Clip1
List.First ();
```

FRIn/FrameIn

This function will start the playback of the clip on a position determined by the framenumber.

Applies to:
```
Clip
```
Syntax:
```
object.FRIn (<frame-number>);
```
Example:

```
Clip lord ("lord.VOB");
lord.FRIn (100); //start playback from frame 100
lord.Prepare ();
lord.Start ();
```

Remarks:

This function requires a pre-processing step where a .PIC file will be created.
This preprocessing must be started manually by using the OSD or the DVP server (see appropriate sections in this manual).
This function only has effect on the `Prepare`, `Prepare&Show` and the `Start` function.
You can also use `FrameIn`, which has the same functionality.

FROut/FrameOut

This function will stop the playback of the clip on a position determined by the framenumber.

Applies to:

`Clip`

Syntax:

```
object.FROut (<frame-number>);
```

Example:

```
Clip heyjude ("heyjude.MPG");
heyjude.FROut (100); //start playback until frame 100 is reached
heyjude.Prepare&Show ();
heyjude.Start ();
```

Remarks:

This function requires a pre-processing step where a .PIC file will be created.
This preprocessing must be started manually by using the OSD or the DVP server (see appropriate sections in this manual).
This function only has effect on the `Prepare`, `Prepare&Show` and the `Start` function.
You can also use `FrameOut`, which has the same functionality.

Get

Scripts can also be used to automatically perform the download (copying files from the removable disk drive to the non-removable disk drive) or upload (copying files from the non-removable disk drive to the removable disk drive). For uploading files we can use the Get function.

Applies to:

`System`

Syntax:

```
System.Get ("filename + file-extension");
```

Example:

```
System.Get ("alive.MPG");
```

This piece of script copies the clip "alive.MPG" from the directory of the non-removable disk drive containing the MPEG clips to the homologous directory of the removable disk drive.

Remark:

The location of the files on the disk drives is implicitly determined by the file-extension (see examples here after). The following file-extensions are recognized by the system:

- MPEG files:".MPG", ".VOB", ".MP1", ".MP2", ".MPV", ".MPA", ".M1V", ".M2V", ".M1p", ".M2P" (\DVP\clips)
- Script files:".SVP", ".PVP" (\DVP\scripts)
- Image files:".GIF" (\DVP\images)
- Binary files (executables): all other extensions (\DVP)

Grid

This function aplies to the *Video* object and allows to divide a touchscreen into a predefined number of elementary *touch-sensistive zones*, by entering an integer number for both the horizontal and the vertical grid size values. The default grid size is 16 (H) by 16 (V) and devides the touchscreen in 256 elementary zones.

Applies to :

`Video`

Syntax:

```
Video.Grid(<HorizontalGridSizeValue>,<VerticalGridSizeValue>);
```

Examples:

```
Video.Grid(32,32);
Video.Grid(20,16);
Video.Grid(720,576); // corresponds to a pixel sized grid (25Hz
MPEG2 MP@ML)
```

Remark:

The default 16 by 16 grid suits to many applications. One can overrule the default gride size by any other integer horizontal and vertical grid size values, where 720 by 576 (25Hz/Pal) and 720 by 480 (29,97Hz/NTSC) can be considered as the "practical" upper limits.

Language

This function selects the audio stream of the DVP used when playing back sound. Audio streams are commonly used for dubbing a movie in different languages.

Applies to:

```
Audio
```

Syntax:

```
Audio.Language (<stream ID>);
```

Example:

```
Clip Welcome (“letstick.MPG”);
Audio.Language (1); //play the clip with audio stream 1
Welcome.Play ();
```

Remarks:

The DVP supports up to 16 audio streams (or ‘languages’) ranging from zero to fifteen. The Language can be set at any time, even during playback. However this is not guaranteed to work. In any case some chirp-sounds or small delays may occur.

Last

This function plays the last clip of a playlist object or a playlist-file.

Applies to:

```
PlayList, PlayListFile
```

Syntax:

```
object.Last ();
```

Example:

```
PlayList List (Clip1, Clip2);
List.Play (); //start playing the playlist
WaitEvent (Input1); //wait for Input1 to start the last clip
List.Last ();
```

Log

Send a message to the DVP and display it in the Reply log. This function is very useful for debugging complex scripts. When applied to *SerialDriver*, it enables or disables to logging of all received commands. The log information is sent to the DVP Server *ReplyLog* window.

Applies to:

```
System, SerialDriver
```

Syntax:

```
System.Log(<string>);
SerialDriver.Log(<true,false>);
```

Example:

```
System.Log(“This message is sent to the DVP Server”);
```

Loop

This function is intended to play iteratively several times the selected *clip* or *playlist*.

Applies to:

```
Clip, PlayList, PlayListFile
```

Syntax:

```
clipname.Loop (N);
listname.Loop (N);
playlistfilename.Loop (N);
```

Examples:

Play the clip *alive* 3 times:

```
Clip alive (“alive.MPG”);
alive.Loop (3);
```

Play the *Hitparade* 5 times:

```
PlayList Hitparade (“alive.MPG”, “letstick.MPG”);
Hitparade.Loop (5);
```

Start the playback of the playlist-file *Hitparade* in an *endless* loop:
```
PlayListFile Hitparade ("music.pvp");
Hitparade.Loop ();
```
Remarks:

If you don't specify the parameter *N*, the loop function will start playing the object in an *endless loop*. Do not loop a clip that only has one frame. For such clips, use the *Play* function instead.

LToL&R

The *LToL&R* function will redirect the left channel of the original stereo sound (when applicable) to both the left and right audio output channels. Thus, both audio output channels will provide exactly the same sound signal, being the original left channel.

Applies to:
```
Audio
```
Syntax:
```
Audio.LToL&R();
```

Mono

The *Mono* function will convert the original stereo sound (when applicable) to mono sound on the audio outputs. The original left and right channels are mixed together, so that both audio output channels will provide exactly the same sound signal.

Applies to:
```
Audio
```
Syntax:
```
Audio.Mono();
```

Next

This function plays the next clip of a playlist object or a playlist-file.

Applies to:
```
PlayList, PlayListFile
```
Syntax:
```
object.Next ();
```
Example:
```
PlayList List (Clip1, Clip2);
List.Start ();
WaitEvent (Input1);
List.Next ();
```

Number

This function starts the playback of the Nth clip in a playlist or a playlist-file.

Applies to:
```
PlayList, PlayListFile
```
Syntax:
```
object.Number (N);
```
Example:
```
PlayList List (Clip1, Clip2);
List.Number (2); //starts playing second clip
```

Off

This function switches off the analog video signals (CVBS, Y/C, RGB). However the DVP continues decoding and the audio remains available.

Applies to:
```
Video
```
Example:
```
Video.Off ();
```
Remark:

By default *Video* is switched *on*.

On

This function switches the video back *on*.

Applies to:

```
Video
```

Example:

```
Video.On ();
```

Pause

Pauses a clip.

Applies to:

```
Clip, Any
```

Syntax:

```
object.Pause ();
```

Example:

Any.Pause ();

Pauses the currently active clip, playlist or playlist-file.

Remark:

When the object is “Any”, then the currently active clip, playlist or playlist-file will be frozen in time.

Playback can be resumed at the point of the interrupt by calling the *Resume* function.

Play

This function is intended to start the playback (once) of a selected object.

Applies to:

```
Clip, PlayList, PlayListFile
```

Syntax:

```
clipname.Play();
playlistname.Play();
playlistfilename.Play();
```

Example:

Start playback of clip “alive”:

```
Clip alive (“alive.MPG”);
alive.Play ();
```

Start the playback of the playlist *Hitparade*:

```
PlayList Hitparade (“alive.MPG”, “letstick.MPG”);
Hitparade.Play ();
```

Start the playback of the playlist-file *Hitparade*:

```
PlayListFile Hitparade (“music.pvp”);
Hitparade.Play ();
```

Prepare

This function is intended to prepare a clip. After preparing the clip, the video decoder will be paused just before displaying the very first frame of the clip.

Applies to:

```
Clip, PlayList, PlayListFile
```

Syntax:

```
clipname.Prepare ();
clipname.Prepare (entity);
```

Example:

```
Clip lord (“lord.VOB”);
lord.Prepare ();
lord.Start ();
Clip lord (“lord.VOB”);
lord.Prepare ();
lord.Start (Output4);
Clip lord (“lord.VOB”);
lord.Prepare (Input7);
```

Remark:

In the last example (`lord.Prepare (Input7);`), the output is only set during 250 microseconds and will be cleared automatically. When `Prepare` is called without the *entity* parameter, the function `Start` must be called to start the playback of the clip.

If the entity parameter is used, then the clip is in the *pause state* until the entity becomes *true*. It is not necessary to call the `Start` function in this case. The entity can be an *input* or a *trigger*. In the first case, `Prepare` waits for the assorted hardware input. In the latter case, `Prepare` waits for the assorted trigger command on the *SSC*.

Prepare&Show

Prepares a clip and shows its first frame.

Applies to:
```
Clip, PlayList, PlayListFile
```
Syntax:
```
clip.Prepare&Show ()
clip.Prepare&Show (entity)
```
Example:
```
Clip myclip (“alive.MPG”);
myclip.Prepare&Show (); //show first picture of clip
WaitEvent (Input1); //wait until Input1 is triggered
```
Remarks:

When `Prepare&Show` is called without the *entity* parameter, the function `Start` must be called to start the playback of the clip.

If the entity parameter is used, then the clip is in the *pause state* until the entity becomes *true*. It is not necessary to call the `Start` function in this case.

The entity can be an *input* or a *trigger*. In the first case `Prepare&Show` waits for the assorted hardware input. In the latter case, `Prepare&Show` waits for the assorted trigger command on the *SSC*.

Previous

Applies to:
```
PlayList, PlayListFile
```
Syntax:
```
object.Previous ();
```
Example:
```
PlayList List (Clip1, Clip2);
List.Number (2); //Start playing the second clip
WaitEvent (Input1); //wait for Input1 to start playing Clip1
List.Previous ();
```

Pulse

The pulse function makes it possible to set (to the "active low" state) a particular output trigger or a combination of output triggers (cfr. output masks) during a given period of time, entered in miliseconds.

Applies to:
```
Output1, Output2, ..., Output8
```
Syntax:
```
OutputN.Pulse (<time_in_miliseconds);
```
or
```
<NameOfOutputMask>.Pulse(<time_in_miliseconds);
```
Examples:
```
Output MyOutput(1,0,*,1);
Output1.Pulse(100); // Output1 will be pulsed to (active low)
during 100 mliseconds
MyOutput.Pulse(5000); // Output1 and 4 will be pulsed during 5
seconds, the status of
Output3 will remain unchanged
```

Put

Scripts can also be used to automatically perform the download (copying files from the removable disk drive to the non-removable disk drive) or upload (copying files from the non-removable disk drive to the removable disk drive). For downloading files we can use the Put function.

Applies to:
```
System
```
Syntax:
```
System.Put (“filename + file-extension”);
```
Example:
```
System.Put (“lord.VOB”);
```
Copy the clip “lord.VOB” from the directory of the removable disk drive containing the MPEG clips to the homologous directory of the internal hard disk drive.

Remark:

The location of the files on the disk drives is implicitly determined by the file-extension (see examples here after). The following file-extensions are recognised by the system:

- MPEG files:“.MPG”, “.VOB”, “.MP1”, “.MP2”, “.MPV”, “.MPA”, “.M1V”, “.M2V”,

".M1p", ".M2P" (\DVP\clips)

- Script files:".SVP", ".PVP" (\DVP\scripts)
- Image files:".GIF" (\DVP\images)
- Binary files: (executables) all other extensions (\DVP)

Random

This function will "random-select" a clip from a playlist object or a playlist file.

Applies to:

```
      PlayList, PlayListFile
```

Syntax:

```
      object.Random ();
```

Example:

```
      PlayList MyList ("PLST.PVP");
      MyList.Random (); // a random chosen clip from
      the external playlist file
      PLST.PVP will start playing
```

Recall

Recall will load the status of a flag which was previously written to disk, using the Store function.

Applies to:

```
      Flag
```

Syntax:

```
      Flag MyFlag();
      MyFlag.Recall();
```

ReplyOnChar

When set to *true*, the *Generic Serial Driver* will respond with the *reply character* on each character received.

Applies to:

```
      SerialDriver
```

Syntax:

```
      SerialDriver.ReplyOnChar(<true,false>)
```

Resume

Resumes a clip after it has been paused.

Applies to:

```
      Clip, Any
```

Syntax:

```
      object.Resume ();
```

Example:

```
      Any.Resume (); //Resumes the currently
      active clip, playlist or
      playlist-file.
```

Remarks:

When the object is *Any*, then the currently active clip, playlist or playlist-file will resume playback. The *Resume* function can only be called on a previously paused clip.

RToL&R

The *RToL&R* function will redirect the right channel of the original stereo sound (when applicable) to both the left and right audio output channels. Thus, both both audio output channels will provide exactly the same sound signal, being the original right channel.

Applies to:

```
      Audio
```

Syntax:

```
      Audio.RToL&R();
```

Scroll

Scrolls the text defined by mytext from right to left. The delay is the time needed to move the text one character. When no delay is specified 500 ms is assumed.

Applies to:
```
      OsdText
```
Syntax:
```
      myText.Scroll(<delay in ms>);
```

SeamlessModeOff, SeamlessModeOn

Set the seamless append mode on or off. Seamless append allows smoother transitions between the clips of a playlist or a playlist-file but is known to cause problems in some occasions.

Applies to:
```
      PlayList, PlayListFile
```
Syntax:
```
      object.SeamlessModeOn ();
      object.SeamlessModeOff ();
```
Example:
```
      PlayList List (Clip1, Clip2);
      List.SeamlessModeOff ();
      List.Play ();
```
Remarks:

The default mode is turned *on*. Seamless append is a 'best effort' service, it does not guarantee a smooth transition. Both functions do not affect PlayListFiles that contain one or more pauses.

Search

The Search function makes it possible to locate to a given frame number in a given clip. In order to do so the clip needs to be preprocessed first. The existence of the associated PIC file (preferably version 4) is therefore mandatory ! The search function will search and locate the clip to the requested position, pause the clip while keeping the video output black. Playback at normal speed can be started by using the resume function.

Applies to:
```
      Clip
```
Syntax:
```
      clipname.Search (<framenumber>);
```
Example:
```
      Clip alive ("alive.mpg");
      alive.Search(250); // search & locate to frame 250 (10 seconds from the start
                    of the clip)
      WaitEvent(Input4);
      Video.On(); // the video output is turned on (was black)
      alive.Resume(); // starts playback from frame 250
```

Search&Show

The Search&Show function differs from the basic Search function by the fact that the video output is not kept to black. After the requested position has been reached, the clip will be paused and the picture will be shown.

Applies to:
```
      Clip
```
Syntax:
```
      clipname.Search&Show (<framenumber>);
```
Example:
```
      Clip alive ("alive.mpg");
      alive.Search&Show(250); // search & locate to frame 250 (10 seconds from the
                    start of the clip)
      WaitEvent(Input4);
      alive.Resume(); // starts playback from frame 250
```

Search&Start

The *Search&Start* function differs from the basic *Search* function by the fact that the clip will not be paused after the requested position has been reached.

Applies to:
```
      Clip
```
Syntax:

```
      clipname.Search&Start (<framenumber>);
```
Example:
```
      Clip alive ("alive.mpg");
      alive.Search&Start(250); // search & locate to frame 250 and start playback
                    immediately
```

Send

This function is used to send a serial code to an *External Device*. Do not forget to enable the *Generic Serial Driver* and disable SSC or PPP.

Applies to:
```
      SerialOut
```
Syntax:
```
      code.Send():
```

Set

This function is intended to activate the selected trigger output (electrically, the active level is low).

Applies to:
```
      OutputN, Flag
```
Syntax:
```
      outputname.Set ();
```
Example:
```
      Output1.Set (); //activate output trigger 1
```

SetDefaults

Reset attributes that were assigned to a clip.

Applies to:
```
      Clip
```
Syntax:
```
      object.SetDefaults ();
```
Example:
```
      myclip.FROut(100); //Finish at frame 100
      myclip.SetDefaults(); //Oops. my mistake
```
Remarks:

The *SetDefaults* function resets the attributes that were updated by the *FRIn*, *FROut*, *Set-Loops*, etc. functions. Playback will start from the beginning and proceed until the end. The *SetLoops attribute* is set to *zero*.

SetEndOfText

The *End-of-text* character is one out of two methods used by the *Generic Serial Driver* to detect the end of a valid serial command. See also *EndOfText()* and *SetTimeOut()*.

Applies to:
```
      SerialDriver
```
Syntax:
```
      SerialDriver.SetEndOfText(<ASCII character or escape sequence>);
```
Example:
```
      SerialDriver.SetEndOfText("\r");
```

SetLB/SetLetterBox

This function is used for letterbox conversion of video clips provided in 16:9 aspect ratio. This way, it is possible to show the 16:9 content on a regular 4:3 screen.

Applies to:
```
      Clip, Video
```
Syntax:
```
      clipname.SetLB ();
```
Example:
```
      Clip BarTime ("TIMEG6E.M2V");
      Bartime.SetLB (); // sets the video output to the letterbox converted format
      Bartime.Play (); // Bartime will start playing in letterbox format.
```
Remarks:

When applied to *Video*, the setting affects all *Clips* that are not set individually. So to play all Clips in *16/9 format* except for clip2, call: Video.SetLB() and clip2.ClearLB().

SetLoops

This function sets the number of repeat cycles for a clip.

Applies to:

```
Clip
```

Syntax:

```
object.SetLoops (<number of loops>);
```

Example:

```
Clip alive ("alive.MPG");
alive.SetLoops (3); //repeat the clip three times alive.Prepare ();
alive.Start ();
```

Remarks:

This function only has effect on the `Prepare`, `Prepare&Show` and the `Start` function. This function is designed to be used together with the commands `TCIn`, `TCOut`, `TMIn`, `TMOut`, etc. Its behaviour is similar to the `Loop` function but it does not start the playback of the clip.

SetReplyChar

This functions sets the reply character. The default value is the character with ASCII value 6.

Applies to:

```
SerialDriver
```

Syntax:

```
SerialDriver.SetReplyChar(<ASCII character or escape sequence>);
```

Example:

```
SerialDriver.SetReplyChar("a");
```

SetTimeout

The timeout is one out of two methods used by the Generic Serial Driver to detect the end of valid serial command. If no timeout is specified, 10 seconds is used.

Applies to:

```
SerialDriver;
```

Syntax:

```
SerialDriver.SetTimeout(<timeout in miliseconds>);
```

Example:

```
SerialDriver.SetTimeout(200);
```

Show

This function displays a selected image on screen.

Applies to:

```
Image, OsdText
```

Syntax:

```
imagename.Show ();
myText.Show();
```

Example:

```
Image logo ("logo.GIF", 1, 60, 70, 80);
logo.Show (); //Display the image "logo"
```

Remark:

It is possible to simultaneously display several images on the screen. Although the whole width of the screen can be used, there is a limitation on the *vertical* area. The display of GIF images must indeed take place in a defined vertical area on the screen that is approximately *half a screen* (294 lines). The area where the user can place GIF files is defined by the `ypos` of the first image drawn on the screen (lines ranging from `ypos` up to `ypos + 293` are usable).

Secondly, the DVP uses a *16-colour range*. If an image consists of more colours, the DVP will automatically transform the image into its allowed colour palette. When several images are displayed simultaneously, the color palette of the first one is applicable for all the images.

The *black colour* will automatically be shown as *transparent*. To prevent this, the user can define *black* as a *darker shade of grey* (e.g. by changing the RGB colour values to 1+1+1).

Slow

Plays a clip in *slow-motion.*

Applies to:

```
      Clip
```

Syntax:

```
      object.Slow (<speed>);
      object.Slow ();
```

Example:

```
      Clip.Prepare&Show ();
      Clip.Slow (3); //Plays a clip three times slower than normal
```

Remarks:

The *speed* parameter indicates the *playback ratio* (ranging from 2 to 10). For example: when the speed parameter is 5, the clip will be played back *five times slower*.

The *Slow* functionality can only be called on a previously prepared, paused or a playing clip.

Start

This function is intended to start a clip after it has been prepared with the Prepare function. The function can be used with a parameter. When used with a parameter, the output gets pulsed as the clip starts. This way, several DVPs can be started synchronously. The DVP, executing the Start function, is master. The slave-DVPs should have one of their input triggers wired to the master's output.

See the examples of the `Prepare` function. The second example should be running on the master DVP. It's *Output4* gets pulsed as the clip starts when executing the `Start` function. The third example should be running on one or more slave DVPs. All their *Input7* input triggers must be connected to the master's *Output4* for the clip to be started on the slaves.

If this function is used with a `Timer`, the function sets the selected `Timer` to the initial status as defined in the declarations and starts the countdown.

Applies to:

```
      Clip, Timer
```

Syntax:

```
      clipname.Start ();
      clipname.Start (output-entity);
      timername.Start ();
```

Examples:

```
      Clip alive ("alive.MPG");
      alive.Prepare (Input7);
      alive.Start (Output2);
      Timer oneminute (60);
      oneminute.Start ();
      Timer 10seconds (10);
      10seconds.Start (15);//resets the timer "10seconds" to 15 seconds
      //and starts the countdown
```

Remark:

Alternatively, the user can *reinitialise* the *duration* of the timer and start this timer with a single statement (e.g. 10seconds.Start (15)). Note that if later the syntax `10seconds.Start();` would be used, the timer would be again initialised to *15 seconds* and not to *10 seconds*.

Status

The following message will be sent to the DVP Server Reply log:

```
"Status of 'clip' (Clip): true."
```

It shows the *name* of the object on which the function was called, its *type* or *classname* and its associated *logical status*.

Applies to:

```
      All classes
```

Syntax:

```
      object.status();
```

Step

This function steps through the clip on a frame-base.

Applies to:

```
Clip
```

Syntax:

```
object.Step (<step-size>);
```

Example:

```
Clip myclip ("alive.mpg"); //Show the clip in slow motion
myclip.Prepare ();
while (true)
{
myclip.Step (3); //step three frames at a time
}
```

Remarks:

The clip has to be prepared (using `Prepare` or `Prepare&Show`) before calling this function. The default *step size* equals to *1*.

Stereo

The *Stereo* function will reset the audio output channels to the original stereo sound (when applicable) in case the audio outputs were previously converted to mono (*Mono* function) or redirected by means of the *LToL&R* or *RtoL&R* function.

Applies to:

```
Audio
```

Syntax:

```
Audio.Stereo();
```

Stop

This function is intended to cancel the execution of a selected object.

Applies to:

```
Clip, PlayList, PlayListFile, Timer, Any
```

Syntax:

```
clipname.Stop();
listname.Stop();
playlistfilename.Stop();
timername.Stop();
Any.Stop();
```

Example:

Stop the playback of the clip "lord":

```
Clip lord ("lord.VOB");
lord.Play();
lord.Stop();
```

Stop the playback of the playlist "Hitparade":

```
PlayList Hitparade ("alive.MPG", "letstick.MPG");
Hitparade.Play ();
Hitparade.Stop ();
```

Stops the playback of the playlist-file "Hitparade":

```
PlayListFile Hitparade ("music.pvp");
Hitparade.Loop ();
Hitparade.Stop ();
```

Stops the execution of the timer "10seconds":

```
Timer 10seconds (10);
10seconds.Start ();
10seconds.Stop ();
```

Stops the playback of the clip that's currently playing:

```
Any.Stop ();
```

Remarks:

The way the playback was initiated (`Play` or `Loop`) does not matter.

Store

The *Store* function will save the current status of a flag to disk. The value of the flag will be written to an .ini file. The status can be loaded from the disk using the *Recall* function, even when the script and/or DVP was restarted.

Applies to:

```
Flag
```

Syntax:

```
Flag MyFlag();
MyFlag.Store();
```

TCIn/TimeCodeIn

This function will start the playback of the clip on a position determined by a time-code specifying hours, minutes and frame number.

Applies to:
```
Clip
```
Syntax:
```
object.TCIn (<hours>, <minutes>, <seconds>, <frame number>);
```
Example:
```
Clip alive (“alive.MPG”);
alive.TCIn (0,5,0,100); //start playback from the 5th minute
alive.Prepare&Show ();
alive.Start ();
```
Remarks:

This function requires a pre-processing step where a .PIC file will be created. This preprocessing must be started manually by using the OSD or the DVP server (see appropriate sections in this manual). This function only has effect on the `Prepare`, `Prepare& Show` and the `Start` function. This function has an alias called `TimeCodeIn`, that has the same functionality.

TCOut/TimeCodeOut

This function will stop the playback of the clip on a position determined by the time-code in hours, minutes, seconds and frame number.

Applies to:
```
Clip
```
Syntax:
```
object.TCOut (<hours>, <minutes>, <seconds>, <frame number>);
```
Example:
```
Clip alive (“alive.MPG”);
alive.TCOut (0,5,0,100); //start playback untill the 5th minute
alive.Prepare&Show ();
alive.Start ();
```
Remarks:

This function requires a pre-processing step where a .PIC file will be created. This preprocessing must be started manually by using the OSD or the DVP server (see appropriate sections in this manual). This function only has effect on the `Prepare`, `Prepare& Show` and the `Start` function. This function has an alias called `TimeCodeOut`, that has the same functionality.

TextSize

The text size ranges from Small (0) to Largest (3). Just as *Transparency* and *BorderColor* this function has effect on both the *OsdText* objects and the *TimeCode Display*.

Applies to:
```
Osd
```
Syntax:
```
Osd.TextSize(<0-3>);
```

TMIn/TimeIn

This function will start the playback of a clip on a position determined by the time in hours, minutes and seconds.

Applies to:
```
Clip
```
Syntax:
```
object.TMIn (<hours>, <minutes>, <seconds>);
```
Example:
```
Clip alive (“alive.MPG”);
alive.TMIn (0,10,0); //start playback from the 10th minute
alive.Prepare&Show ();
alive.Start ();
```
Remarks:

This function requires a pre-processing step where a .PIC file will be created. This preprocessing must be started manually by using the OSD or the DVP server (see appropriate sections in this manual). This function only has effect on the `Prepare`,

`Prepare& Show` and the `Start` function. This function has an alias called `TimeIn`, that has the same functionality.

TMOut/TimeOut

This function will stop the playback of the clip on a position determined by the time in hours, minutes and seconds.

Applies to:
```
      Clip
```
Syntax:
```
      object.TMOut (<hours>, <minutes>, <seconds>);
```
Example:
```
      Clip alive (“alive.MPG”);
      alive.TMOut (0,5,0); //start playback until the 5th minute
      alive.Prepare&Show ();
      alive.Start ();
```
Remarks:

This function requires a pre-processing step where a .PIC file will be created. this preprocessing must be started manually by using the OSD or the DVP server (see appropriate sections in this manual). This function only has effect on the `Prepare`, `Prepare& Show` and the `Start` function. This function has an alias called `TimeOut`, that has the same functionality.

TrackModeOn, TrackModeOff

The track mode determines if the DVP stops after each clip in the playlist or playlist-file.

Applies to:
```
      PlayList, PlayListFile
```
Syntax:
```
      object.TrackModeOn ();
      object.TrackModeOff ();
```
Example:
```
      PlayList List (Clip1, Clip2);
      List.TrackModeOn ();
      List.Play (); //start playing and stop after first clip
```
Remarks:

The default track mode is set to *off*. Both functions do not affect PlayListFiles that contain one ore more pauses.

Transparency

Used to set the transparency level (ranging from 0% to 87.5%) of all *OsdText* and the *Time-Code Display*. The chosen level will still be in effect when the script has terminated.

Applies to:
```
      Osd
```
Syntax:
```
      Osd.Transparency(<0-7>);
```

Volume

This function sets the audio volume of the DVP.

Applies to:
```
      Audio
```
Syntax:
```
      Audio.Volume (<volume level>);
```
Example:
```
      Audio.Volume (6);
```
Remarks:

The volume is set using a number between zero and ten, where the levels correspond to the following decibels (dB):
Setting the volume to *zero mutes* the audio.

VOLUME	DECIBEL
10	+3
9	0
8	-3
7	-6
6	-9
5	-12
4	-15
3	-18

2	-21
1	-24
0	Mute

Wait

This function is intended to delay the execution of the next statement in the script till the end of the object the wait function is initiated for. This can be till the end of a clip, till the end of a timer or till the end of a loop, etc....

Applies to:

```
Clip, PlayList, PlayListFile, Timer, InputN, Time, Date, Day of the
week, TimeInterval, BarCode and SerialIn
```

Syntax:

```
clipname.Wait ();
listname.Wait ();
playlistfilename.Wait ();
timername.Wait ();
inputname.Wait ();
timename.Wait ();
datename.Wait ();
day_of_the_week.Wait ();
timeInterval.Wait();
barCode.Wait();
serialIn.Wait();
```

Examples:

Wait to execute next statement until the clip “*alive*” played till the end:

```
Clip alive (“alive.MPG”);
alive.Play ();
alive.Wait ();
```

Wait to execute next statement until the end of the playlist ”*Hitparade*”:

```
PlayList Hitparade (“alive.MPG”, “letstick.MPG”);
Hitparade.Play ();
Hitparade.Wait ();
```

Wait to execute next statement until the end of the playlist-file “*Hitparade*”:

```
PlayListFile Hitparade (“music.pvp”);
Hitparade.Loop ();
Hitparade.Wait ();
```

Wait to execute the next statement until the *end of the timer*:

```
Timer 10seconds (10);
10seconds.Start ();
10seconds.Wait ();
```

Wait until *Input trigger 4* is activated (electrically the active is low):

```
Input4.Wait ();
```

Wait until *noon* to execute next statement in the script:

```
Time noon (12, 0, 0);
Noon.Wait ();
```

Wait until *Christmas* to execute next statement in the script:

```
Date Christmas (25,12,*);
Christmas.Wait ();
```

Wait until *Saturday* to execute next statement in the script:

```
Saturday.Wait ();
```

Wait&Play

This function waits for a bar code to be scanned. When the scanned bar code exists in the playlist file, the associated clip is started.

Applies to:

```
BarCodePlayListFile/BCPlayListFile
```

Syntax:

```
myBarCodePlayListFile.Wait&Play();
```

Associated logical status

During the execution of scripts, a logical status *True* or *False* is given to any of the objects and classes. This permits the user to execute conditionally given statements of the scripts as a function of the status of these entities (see the section “Control structures” here after).

Status of the “BarCode” entity:

The status given to a “BarCode” entity becomes “True” when the associated code has been scanned. After evaluation in the script, the status auto-resets to “False”.

Status of the “Button” entity:

The status given to an “Button” entity becomes “True” when the associated screen area is pressed. After evaluation in the script, the status auto-resets to “False”.

Status of the “Clip” entity:

The status given to a “Clip” entity is “True” during the playback of this clip (“False” otherwise). The status of a “Clip” entity is also “True” after it has been prepared with the Prepare function, even if it hasn’t really been started yet.

Status of the “Flag” entity:

Since a Flag is a boolean variable, it is “True” when it is set and “False” when it is cleared.

Status of the “InputN” entities:

The status given to an “Input” entity is “True” while the associated input trigger is active (“False” otherwise). The same applies to combined input masks (see Input class).

Status of the “OutputN” enities:

The status given to an “Output” entity is “True” while the associated output trigger is active (“False” otherwise). The same applies to combined output masks (see Output class).

Status of the “PlayList” entity:

The status given to a “PlayList” entity is “True” during the playback of the clips in the playlist (“False” otherwise).

Status of the “PlayListFile” entity:

The status of a given “PlayListFile” entity is “True” during the playback of the clips in that entity (“False” otherwise).

Status of “SerialIn”:

The status given to a “SerialIn” entity is “True” if the associated code was received by the Generic Serial Driver. When evaluated in the script the status becomes “False”.

Status of “TimeInterval”:

The status given to a “TimeInterval” is “True” while the current time has passed the start time of the interval and has not yet passed the end time of the interval. The status is “False” otherwise.

Status of the “Timer” entity:

The status given to a “Timer” entity is “True” while this timer is counting down (“False” otherwise).

Status of the “Trigger” entity:

The status given to a “Trigger” entity becomes “True” when the associated message is received. After evaluation in the script, the status auto-resets to “False”.

Status of the “Time”, “Date” and “day of the week” entities:

The status given to these entities is “True” if the time, date or day of the week corresponds to the current time of the VGP dLite system and “False” otherwise.

Pay particular attention to the use of wildcards. Some examples with “Time”:

- ❑ (11, 20, 0) is only True on 11:20:00 AM (no wildcards used, so the 3 numbers are checked)
- ❑ (*, 0, 0) is True during one second at the start of every hour (the wildcard means: all hours are valid)
- ❑ (*, *, 29) is True during one second, every 30th second of every minute (the wildcards mean: hours and minutes are not checked)
- ❑ (08, *, 29) is True during the 30th second, every minute between 8 AM and 9 AM (only hours and seconds are checked)
- ❑ (20, 15, *) is True during one minute, i.e. 08:15 PM (the wildcard means: all seconds are valid)
- ❑ (13, *, *) is True during one hour, i.e. 1 PM (minutes and seconds are wildcards and hence not checked)

Control Structures

The control structures permit the conditional execution of some tasks, a task being defined as the sequence of statements include between {} of the conditional structure statement. For all of the following control structures, it is first proceeded to the evaluation of a condition. Depending on the status of the condition ("True" or "False") the system will perform (or not) a given task
The condition is a logical expression involving any of the above-defined entities. The allowed operators are "And", "Or" and "Not". For those who are familiar with "C/C++" programming these operators can be replaced by, respectively "&&", "||" and "!". For the syntax please refer to the examples here after.

If

If the condition is "True" the script will execute the statements belonging to "task" once. Otherwise the script jumps immediately to the next instruction.

Syntax:

```
if (<conditon>)
{
<task>
}
```

Examples:

```
if (Input1)
{
Output1.Set ();
}
```

If `Input1` is activated the output trigger `Output1` is asserted. Otherwise `Output1` stays as it is.

If … else …

If the condition is "True" the script will execute the statements belonging to "task1" once. Otherwise the statements belonging to "task2" are executed once.

Syntax:

```
if (<condition>)
{
<task1>
}
else
{
<task2>
}
```

Examples:

```
if (Input1)
{
alive.Start ();
alive.Wait ();
}
else
{
letstick.Start ();
letstick.Wait ();
}
```

If `Input1` is activated the clip `alive` is played once entirely. Otherwise the clip `letstick` is played once entirely.

Repeat

The `Repeat` structure can be used to repeat a section of code for a defined number of times.

Syntax:

```
repeat (<number>)
```

```
{
<task>
}
```

WaitEvent

The script waits until the assertion of the “condition”, then jumps to the next statement.

Syntax:

```
WaitEvent (<condition>);
```

Examples:

Wait till input trigger 2 or 3 is activated:

```
WaitEvent (Input2 || Input3);
```

Activate output trigger 2 when condition is true:

```
Output2.Set ();
Oneminute.Wait ();
Oneminute.Start ();
Output2.Clear ();
```

While

The statements between the {} are executed *iteratively* as long as the condition is verified (“True”). As soon as the condition is “False” the script goes to the next statement.

Syntax:

```
while (<condition>)
{
<statement1>
<statement2>
<statementN>
}
```

Examples:

```
while (Not Input1 And Not Input2)
{
alive.Play ();
alive.Wait();
}
```

The clip `alive` is played in *loop mode* till the activation of any of the input triggers 1 or 2. Note that the status of `Input1` and `Input2` is taken into account only at the end of each playback of the script.

Chapter 8: Input/output triggers

Input triggers

The execution of the scripts can be synchronized on the activation/deactivation of the socalled *input triggers*. Actually the *input triggers* are input *signals* delivered to the DVP by an external unit on the pins of a dedicated connector located on the back panel.
When instructed so, the script will sense the status of these input signals and will react according to their status.
In the scripts, reserved keywords (Input1, …, Input8) are used to represent the *eight* input triggers. An input trigger is said *inactive* when the input voltage on the homologous pin is comprised between 2.0 V and 5 V. Electrically, the input triggers are *TTL compatible*. An *internal pull-up resistor* (10 kOhm) is connected between each trigger input and the internal +5 V. The *floating input triggers* are thus *inactive*. Any of the input triggers can be activated by *directly* connecting (*short-circuit*) the related pin to the *GND pin* of the same connector.
A typical example for the use of an input trigger follows: at the main entrance of a building one wishes to start the playback of a *welcome clip* each time a visitor enters. A switch is coupled to the main entrance door and shortens the pins INPUT1 and GND each time someone opens the door. The script senses INPUT1 and starts the playback on the activation of the trigger.

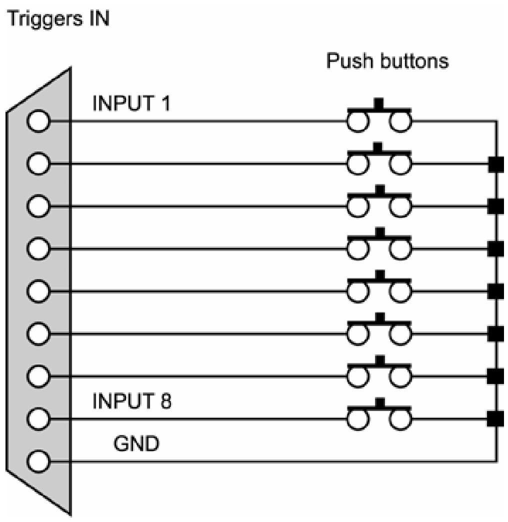
Output triggers

Conversely, up to *four output signals* (called *output triggers*) can be activated/deactivated under *software control*. These signals are available on the pins of a dedicated connector located on back panel.
In the scripts these outputs are represented by reserved keywords (Output1, …, Output4). The output triggers are *active low signals*. Electrically, due to an internal 10 kOhm pull-up connected to the +5 V, they are *TTL compatible*. In the active state, they can sink up to 10 mA each (open collector mode). When operated in *open collector mode*, the maximum voltage is limited to +5 V when the output trigger is inactive.
Typically these outputs can be used to drive a *relay* or to activate the *input triggers* of another DVP.

Chapter 9: Keypad guidelines for input triggers

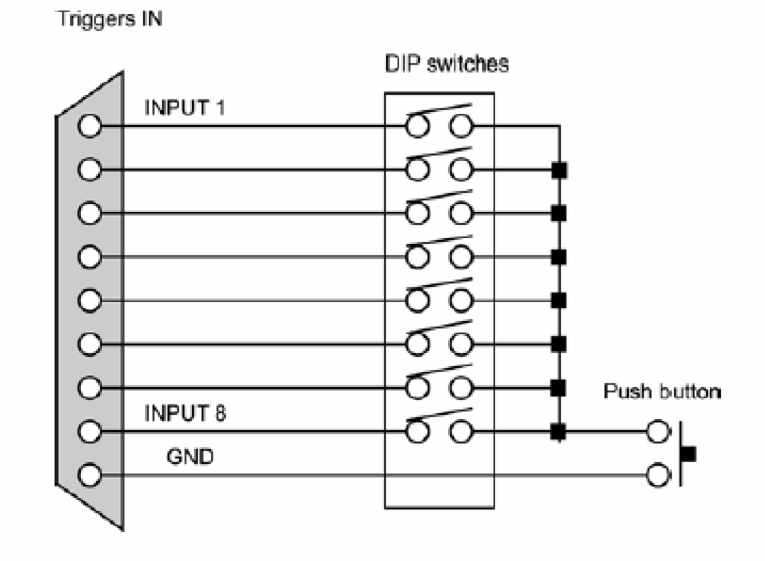
The goal of this section is to give some guidelines about how to implement simple control devices able to generate the input triggers required by given applications.

1. Let us first assume that the application makes use of max. 8 different external events. The simplest way to proceed is then obviously to associate the activation of a well given input trigger to each of these events. In this case simultaneous activation of several triggers must be discarded by the script as being not relevant.

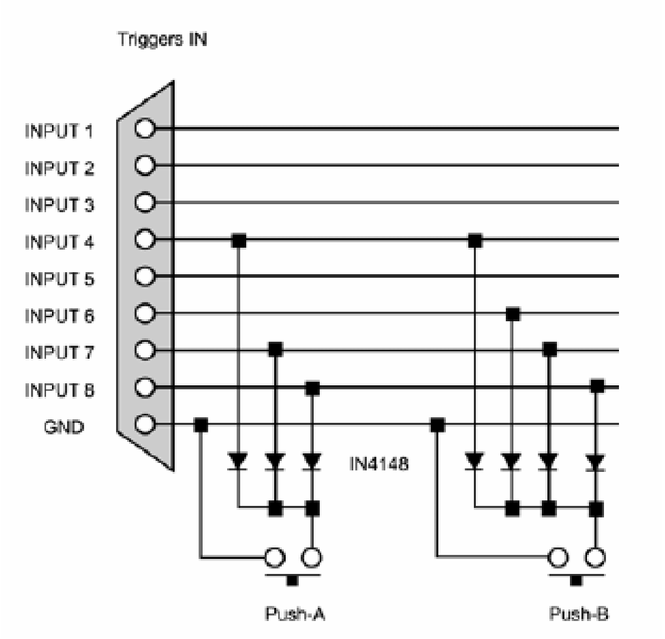


2. Let us now assume that more than 8 external events (but 255 at max) are required by the application. We have to distinguish here between the two following approaches.

- ❑ The code to be presented on the triggers pins is first prepared (by means of DIP-switches or by means of a thumbwheel) then presented on the triggers inputs. Practically, the presentation occurs when the push button is depressed. This way to proceed guarantees the simultaneous (de)assertion of all the inputs.

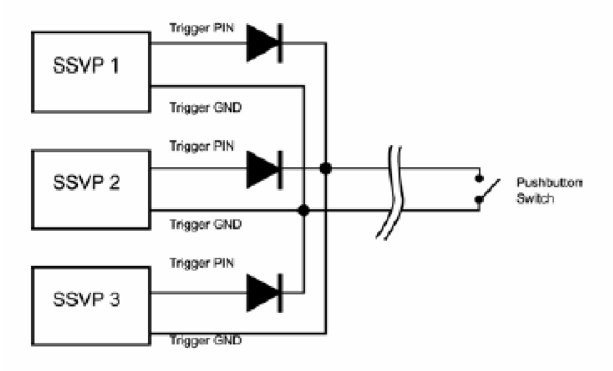


❑ The control device makes uses of one dedicated button/key for each of the external events. The suggested implementation is illustrated here after.



The diodes are usual 1N4148 (not critical). As above, this way to proceed guarantees the simultaneous (de)assertion of all the inputs. In this example the inputs 4,7,8 are asserted when the button "A" is depressed, while the activation of button "B" asserts the inputs 4,6,7,8. If several buttons are depressed simultaneously, the code presented on the triggers inputs is a bitwise "OR" of the codes corresponding to each of the involved buttons.

3. The following schematics demonstrate the use of input triggers on multiple DVP's. The first figure is the connection diagram for triggering a number of DVP's with one single pushbutton. It's recommended to apply a cabling diagram with blocking general purpose diodes as shown in the figure below. The reason for this security measure is to prevent unwanted triggering when one of the units is switched off and on again (accidentally or for maintenance reasons). When this particular unit or its replacement unit boots up, a short *active low* status is generated on its input trigger terminal, which might cause the other units to start an event in case they are waiting for an input trigger.



Chapter 10: Simple Serial Control

This section will give you more specific information about SSC.

The DVP may be controlled via a simple serial protocol, using messages sent via RS-232.

The data-format should be 8 data bits, no parity and 1 stop bit. The baudrate can be selected in the General Communication settings of the OSD menu. In the same submenu, make sure to enable the SSC Protocol.

The protocol is ASCII-based. Commands should be upper case characters.

Any message consists of the address of the DVP, followed by a @-sign, by the command and its optional parameters. Each message ends with a carriage-return, indicated here with <CR>. In the description of the commands below, the address is written as <address>. You only have to type the number itself, not the "<" and ">" characters. The address is used to distinguish between messages meant for different players. Each DVP has its own address which can be set in the General Communication settings in the OSD menu.

Select File (SE)

This command lets you select the filename of the clip to be played or the script to be executed. The filename must be enclosed between quotation marks. This command has no effect on the playback of the current clip, if any.

```
<address>@"<filename>"SE<CR>
```

The DVP responds with the letter R followed by <CR>.

Play (PL)

This command lets you play the previously selected clip or start the previously selected script.

```
<address>@PL<CR>
```

This command stops the current playback.

The DVP responds with the letter R followed by <CR>.

Pause (PA)

This command halts a playing clip. The picture goes black if the monitor is connected to an analogue video output. The RE command will resume a paused clip. If the clip is already stilled, the DVP will ignore the Pause-command.

```
<address>@PA<CR>
```

The DVP responds with the letter R followed by <CR>.

Note that this command should not be used when a script is selected (instead of a clip).

Still (ST)

This command halts a playing clip. The stilled picture remains on the screen. The RE command will resume a stilled clip. If the clip is already paused, the VGP dLite will ignore the Still-command.

```
<address>@ST<CR>
```

The DVP responds with the letter R followed by <CR>.

Note that this command should not be used when a script is selected (instead of a clip).

Stop command (RJ)

This command causes the DVP to stop the playback and to freeze the screen.

```
<address>@RJ<CR>
```

The DVP responds with the letter R followed by <CR>.

Active Mode Request (?P)

This command causes the DVP to report its operating mode.

```
<address>@?P<CR>
```

The DVP responds with P0n where *n* indicates the mode as follows:

1 = stopped (after a Stop-command or if the end of the clip was reached)

4 = playing

5 = stilled

6 = paused

Frame number (?F)

This command returns a five digit number. This number is the number of the frame actually shown.
`<address>@?F<CR>`

Time (?T)

This command returns a five digit number, indicating the actual time of playback. The time is in the following format: HMMSS.
`<address>@?T<CR>`

Time code (includes the frame number) (?TC)

This command reproduces a 8 digit number indicating the actual playback time and the actual frame number. The format is HHMMSSFF. The FF stands for the actual frame displayed within the running second.
`<address>@?TC<CR>`

Attention!
Commands with syntax errors are ignored by the DVP (although "R<CR>" is returned)!

Loop Play (LP)

The Loop Play-command plays the selected clip an infinite number of times.
`<address>@LP<CR>`

Resume (RE)

The Resume-command resumes a previously paused or stilled clip.
`<address>@RE<CR>`

Volume (VO)

The Volume-command controls the output volume ranging from 0 (muted) to 10 (maximum volume).
`<address>@VO<volume><CR>`

Language (LA)

The Language Command-selects the audio stream to be used when decoding a clip. The language ranges from zero to fifteen.
`<address>@LA<stream><CR>`

Volume Request (?V)

The Volume Request-command returns the current volume level.
`<address>@?V<CR>`

Language request (?L)

The Language Request-command returns the currently selected audio stream.
`<address>@?L<CR>`

Trigger (TR)

The Trigger-command allows a script to listen to the SSC connection. This is done by supplying an identifier that corresponds to the identifier that is used in the script (see the Trigger-class, in the programming section of this manual).
`<address>@TR<identifier><CR>`
The identifier is always a three digit number. The number must be *zero-padded* if its size is smaller. For example, the command "1@TR005" is valid but "1@TR5<CR>" is not, because the latter is lacking two extra digits.

Time Code (TC)

This command starts from – or plays until a position in the clip. The position is specified by the time code. The command takes two parameters:
`<address>@TC<time code><mode><CR>`

- ❑ Time codes have following format: `HHMMSSFF`. `HH` is hours, `MM` are minutes, `SS` are seconds and `FF` is frame count. The field must be exactly 8 characters in size. The minutes, seconds and/or frame count are to be zero padded if necessary.
- ❑ The mode can be:
- ❑ SE: Search
- ❑ SM: Stop Marker
- ❑ SL: Search & Play

Time (TM)

This command starts from – or plays until a position in the clip. The position is specified by the time. The command takes two parameters:
`<address>@TM<time><mode><CR>`

- ❑ The time field has following format: `HHMMSS`. `H` is hour, `MM` are minutes and `SS` are seconds. The field must be exactly 5 characters in size. The minutes and/or seconds are to be zero padded if necessary.
- ❑ The mode can be:
- ❑ SE: Search
- ❑ SM: Stop Marker
- ❑ SL: Search & Play

Frame (FR)

This command starts from – or plays until a position in the clip. The position is specified by the frame number. The command takes two parameters:
`<address>@FR<frame number><mode><CR>`

- ❑ The frame number field has following format: `FFFFF`, where `F` is one digit. The field must be exactly 5 characters in size.
- ❑ The mode can be:
- ❑ SE: Search
- ❑ SM: Stop Marker
- ❑ SL: Search & Play

Marker Clear (MC)

lear the stop marker. This command will remove the stop marker that was previously nserted by the TC, TM or FR command.
`address>@MC<CR>`

Video Black (VB)

urns the video output black.
`address>@VB<CR>`

Video Off (V0)

urns the video output off.
`address>@V0<CR>`

Video On (V1)

urns the video output on.
`address>@V1<CR>`

Slow (SP)

lays the clip in slow motion. The speed parameter can range from 2 to 9 and indicates he playback ratio. The clip has to be started or paused before this command can be used. he default speed parameter is 2.
`address>@SP<speed><CR>`

Step Forward (SF)

teps forward through the clip. The 'frame count' parameter indicates the number of rames that will be shown.
`address>@SF<frame count><CR>`

File Listing (LS)

he LS command followed by a directory name (clips, scripts or images) will produce an SCII dump of the contents of the directory. When no directory is supplied, the *clips* irectory is chosen. Every filename is followed by a <CR> code. The list is terminated by R<CR> code.
`address>@LS<directory><CR>`

- ❑ LSclips
- ❑ LSscripts
- ❑ LSimages
- ❑ LSpages
- ❑ LSfonts

Chapter 11 Troubleshooting

Syntax error in script

This error message indicates that there is a mistake in the script file. The most common mistakes are made by forgetting a ';' or a '}'.
recheck the script and undo every syntax mistake.

A script command is ignored

This is possibly due to a typing error. The DVP will not show a 'syntax error' message if a typing error occurred in the script or a specify clip does not reside on the DVP.
You can trace the execution of a script by turning on the 'user logging' feature in the OSD menu. A file with the current date will be created in the \DVP\logs directory. It contains a list of all the actions the DVP has done since logging was turned on.

Changing OSD settings takes some time to have effect or reboots my DVP

This is expected behaviour of the DVP. The DVP can need some time or needs to reboot in order to let the changes take effect. This is especially true when changing the communication settings: changing the IP address of the DVP can cause external network devices (like routers) to get confused. It may require specialized help from a system administrator for the network to adapt.
Make sure not to change major OSD settings (like network- and videostandards) when the DVP is playing a script.

Setting the audio language freezes the playback of the current clip

This is a known limitation of selecting the audio language. It occurs when the language setting is updated after it has past a boundary between two clips with different properties. This is shown in the following diagram:

```
Set Language→Play Clip1.MPG→ Play Clip2.VOB → Set Language
```

In this case, Clip2 has to be restarted to restore the DVP.

Some script functions (like FRIn and FROut,…) cause unexpected behaviour on a clip

To use these functions, the clip has to be preprocessed (a .PIC file has to be created). You can use the OSD menu or DVP Server to preprocess a clip.

Touch screen, bar code reader or serial controller don't work

To use any external device connected to the RS-232 port of the DVP, the correct driver needs to be selected in the OSD menu. Follow these steps to resolve any configuration errors:

- ❑ In Communication-General Settings, the SSC Protocol should be disabled.
- ❑ In Communication-External Devices, the correct device (touch screen, bar code reader, ...) should be selected.
- ❑ For the selected device, make sure its additional parameters (driver, logging, baudrate, ...) are set as required.