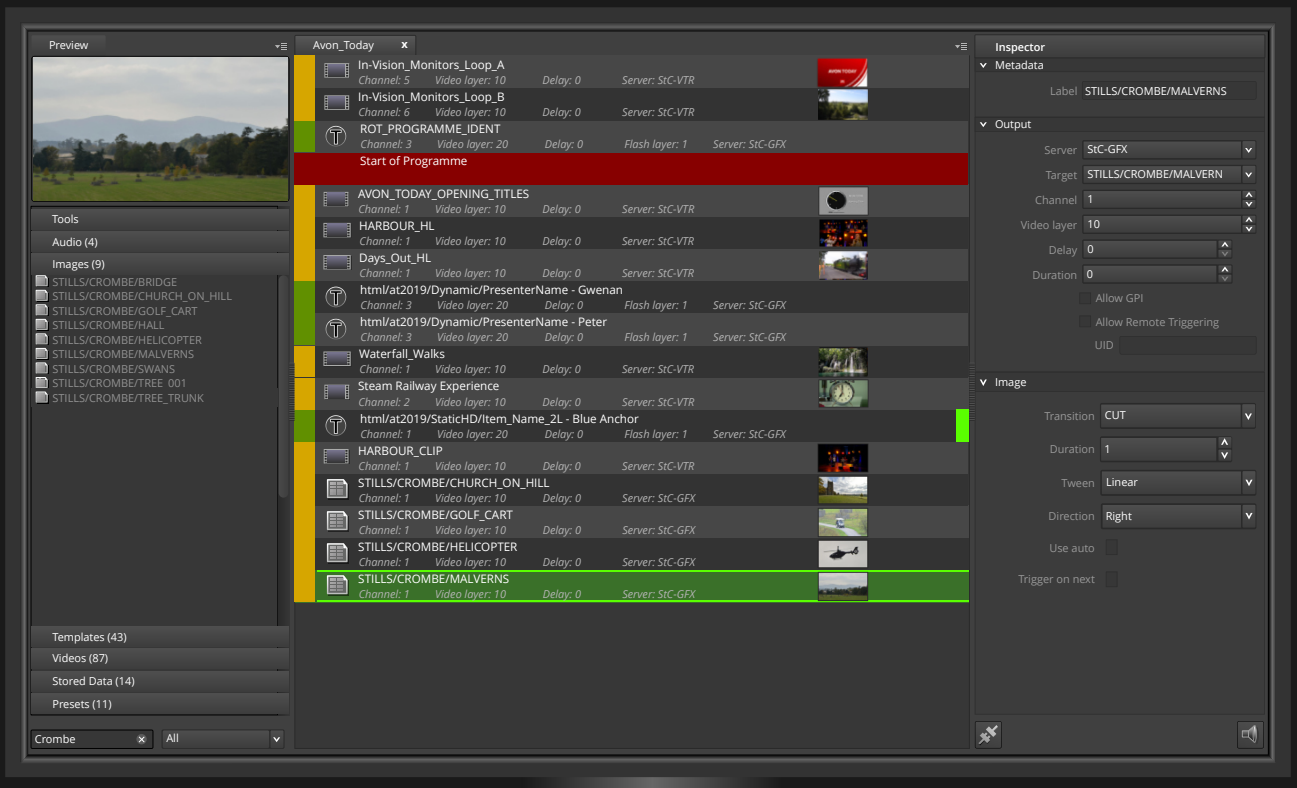
An Introduction to CasparCG Graphics System



Andy Woodhouse

About this document

This document was conceived, written, drawn and typeset by Andy Woodhouse.

The document was typeset using Affinity Publisher using Inria Serif as the primary typeface. Graphics were created in Affinity Designer and/or Affinity Photo.

Document Version 1.1
Document Date August 2022

Email: andy@amwtech.co.uk

Document Use

This document is distributed free of charge in the hope it may help others learn about the CasparCG System.

Anyone obtaining the document may pass a copy or copies to others, use it for training others and make multiple copies for their own use. The author requires that no charges are made for such copies other than those to cover any physical media used in creating the copies (for example printing costs, paper costs, magnetic media costs). This statement of document access must be included verbatim in any copies made.

Disclaimer of Warranty.
There is no warranty for the document or its content, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holder and/or other parties provide the document “as is” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance and use of the document is with you. Should the document prove defective, you assume the cost of all necessary servicing, repair or correction.

Limitation of Liability.
In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who modifies and/or conveys the document as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the document (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties), even if such holder or other party has been advised of the possibility of such damages.

Acknowledgements

My thanks go to the development team in Sveriges Television (SVT) for creating CasparCG, and to SVT managers at who agreed to publish the software under an open source licence.

I also want to thank the members of the CasparCG Support Forum who provide ongoing support for all CasparCG users. The support forum is accessible at https://casparcgforum.org/

There is a CasparCG Wiki accessible at https://github.com/CasparCG/help/wiki

Preface

I discovered CasparCG some years ago through a posting in the Blackmagic Design User Forum. Since that discovery I have installed multiple instances of CasparCG and introduced many individuals to its potential applications. I have created various flash and HTML templates, and used the deployed systems in many training environments.

My aim in publishing this document is to assist new users learn the capabilities of CasparCG and speed them to their own further investigations.

There has been no formal publication edit process, so I apologise in advance for any spelling or grammar errors that have slipped through into the published document.

The primary distribution for this document is through Github at:

https://github.com/amwtech/CasparCG_Documentation

Please use the Issues feature in Github to log any problems.

CasparCG server can be deployed on both Windows and Linux operating systems. For simplicity this document only covers the operations on a Windows host system using a Windows Client..

I have also decided to concentrate the descriptions predominantly using server version 2.3.3 Long Term Support working with client version 2.2 as these are the newest versions available at the time of writing.

Terminology note:
I apologise to any readers who are confused by the spelling I use for an instance of a typeface with set weight and size - **fount**. This spelling was the standard in UK English when I first met typography and TV character generators. At that time a **font** was located in a church and used for baptism. The standard dictionary (Oxford English Dictionary or OED) now recognises both spellings for typography applications.

Andy Woodhouse.

Contents

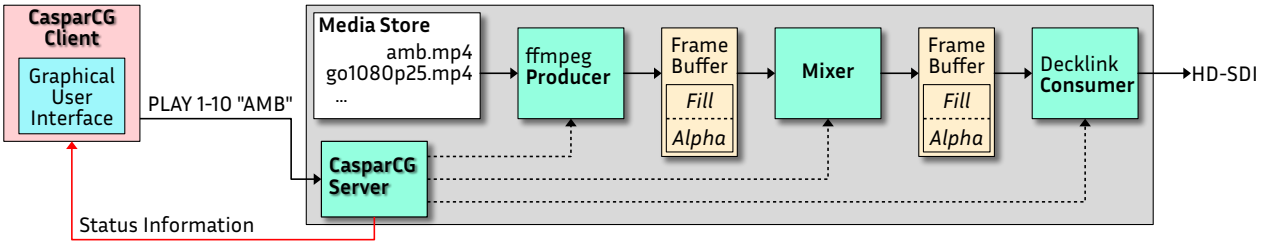
System Overview

CasparCG is a broadcast quality graphics and video playback system created by Sveriges Television (SVT). It was initially developed for use within SVT, and subsequently released as open source software under version 3 of the GNU General Public Licence.

CasparCG uses a client-server architecture. CasparCG ***clients*** send text-based control commands, known as Advanced Media Control Protocol or AMCP, to a CasparCG ***server***.

The server software manages the replay of audio clips, video clips, rendering of templated graphics and combining multiple graphical elements into a composite bitmap. Templated graphics, such as lower thirds, combine a design layout (the template) with instance data.

An example process flow for video clip replay is illustrated below.



The client software has a graphical interface enabling the user to select a file, in this example called amb.mp4, and to designate the output channel number and layer for the content. More detail about channels and layers is given later in this chapter.

When the user presses the Play key the client sends the AMCP command string `PLAY 1-10 "AMB"` to the selected server. This command requests the server locate video media whose stem name is AMB, playing the content in layer 10 of channel 1. Note the command does not require the target file extension.

The server parses the received command, looking for a matching file name in the media store. A status report is sent to the client indicating if the play process could be started (the file was found), or if there is some error condition.

The file name is passed to the ffmpeg producer that reads and decodes the source content, rendering the media into a frame buffer. The frame buffer has storage for both the fill content and for any associated alpha content. A full frame alpha channel is generated for files that do not have a stored alpha signal.

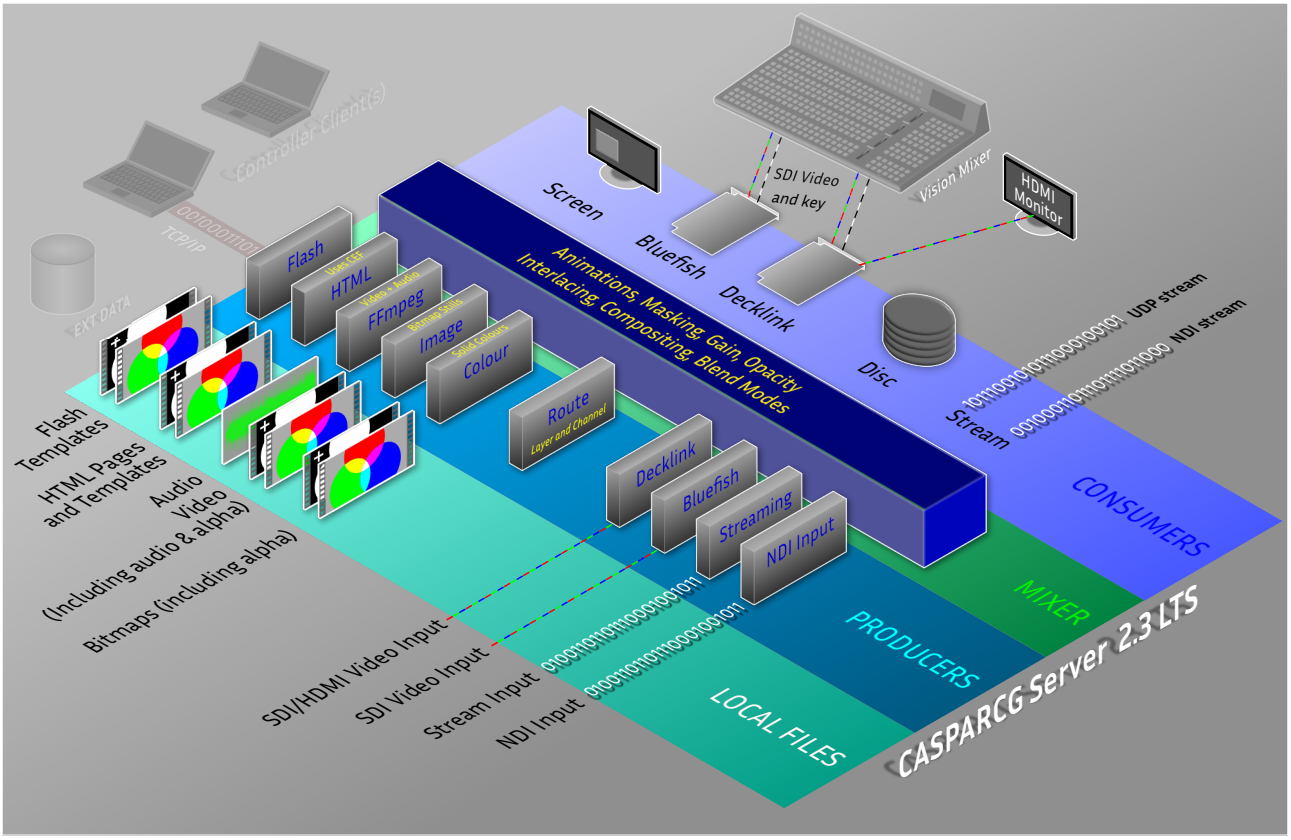
The frame buffer passes the decoded content to the Mixer process. The mixer applies any user specified size changes, position offsets or other manipulations such as opacity, combining the layer 10 content with any other active layers of the assigned channel, for example overlaying a lower third name strap. The composited output from the mixer is placed in a frame buffer.

The output of the mixer passes to one or more consumers for the channel. A consumer converts the bitmap from the mixer into a chosen output format, for example SDI video and key. Multiple consumers can be used on a channel enabling the output as SDI whilst simultaneously making a recording of that output.

In the above illustration, the consumer uses a Blackmagic Design Decklink card to output an HD-SDI output signal with embedded audio.

The server sends ongoing status about the active channels and the progress of the file replays to all connected clients. This status information enables the client software to display a progress meter showing the percentage of the clip played and a countdown to the end of media playout.

There are multiple producers and consumers. The number of producers and consumers may differ between versions of the server software. The following illustration shows the facilities in SVT version 2.3 server.



A server creates one or more logical ***channels***. Each channel is the output of a compositing process for multiple ***layers*** executed by the ***mixer*** module. Logical channel outputs are converted to physical outputs, such as SDI video, by one or more ***consumers***.

Media files are processed by the relevant ***producer*** module rendering a bitmap and associated alpha/key information into a frame buffer. The producer output is targeted at a user-specified ***layer*** of the final composite.

When the software was initially developed ***Flash producer*** was the main mechanism used for rendering templated graphics. Most new applications of CasparCG now use HTML templates. The latest server version disables the Flash producer by default as Flash technology has reached end of life.

The ***HTML producer*** uses an embedded Chromium engine to display web pages or to implement rendering of templated graphics. The Chromium engine includes support for WebSocket technology, enabling a web server or web browser to push content directly to a display page. Specialist web-servers can be created using server side processing technologies, enabling full page graphics that continuously update, for example sports scores or weather information. A CasparCG client can still send update requests to a suitably authored HTML page, the page code in turn requests data from the web-server (for example to display a different football league).

HTML templated graphics pages can be created using any combination of manual coding, GUI created SVG data that is manually coded for animation, or using a GUI design and animation tool such as Adobe Animate, Google Web Designer or Tumult Hype (macOS only). Some manual coding is used to adapt the inherent animation facilities to support triggering by CasparCG commands. There is a dedicated GUI template generator for CasparCG called loopic. See https://loopic.io for more detail.

Video and audio clips are processed by the ***ffmpeg producer*** supporting a wide range of file wrappers and codecs such as ProRes codec in .mov wrapper, DNxHD in a .mov wrapper, DNxHD in .mxf wrapper, AVC Intra codec in .mxf wrapper. Audio WAV files from a CD with 44.1 kHz sampling are sample rate converted by ffmpeg to 48kHz for use in a video environment.

The ***image producer*** renders bitmaps from still images stored as PNG, BMP, TGA, TIFF or JPEG. It recognises embedded alpha channels, using the fill and alpha in the compositing operations executed in the mixer module. The image producer auto-scales stills to fit the target channel size, enabling an Ultra-1 resolution source image to be shown in a channel operating in HD 1080i.

The ***scroll producer*** is part of the image producer function. It uses images which have one dimension that matches a channel output dimension. If the matched dimension is image width, the scroll producer operates the image as a roller caption. If the matched dimension is image height, the scroll producer uses the image as a crawler caption.

The ***colour producer*** creates full-screen solid colour displays. The colour property is specified by both the red, green and blue amplitudes with an overall opacity (alpha) amplitude. Using the mixer blending facilities allows complex tinting operations of stills or moving video.

The ***Decklink producer*** supports the input of live SDI video via a Decklink card. This supports record and picture-in-picture operations.

The ***Bluefish producer*** supports input of SDI via a Bluefish444 video card. This supports record and picture-in-picture operations.

There are multiple IP ***Stream producers***. Native Newtek NDI inputs are supported on server version 2.3_LTS. Several MPEG-based streams can be used as live inputs using the ffmpeg producer.

The ***Route producer*** sends either a selected layer or the composited channel output to a video layer in another channel.

CasparCG allows the use of Virtual Channels that have a defined spatial and temporal resolution, but that are not routed to a consumer. Virtual channels are used with the route producer to support various production requirements. Examples include:

1. sending a moving graphics background to multiple output channels ensuring temporally aligned outputs for downstream vision mixing;

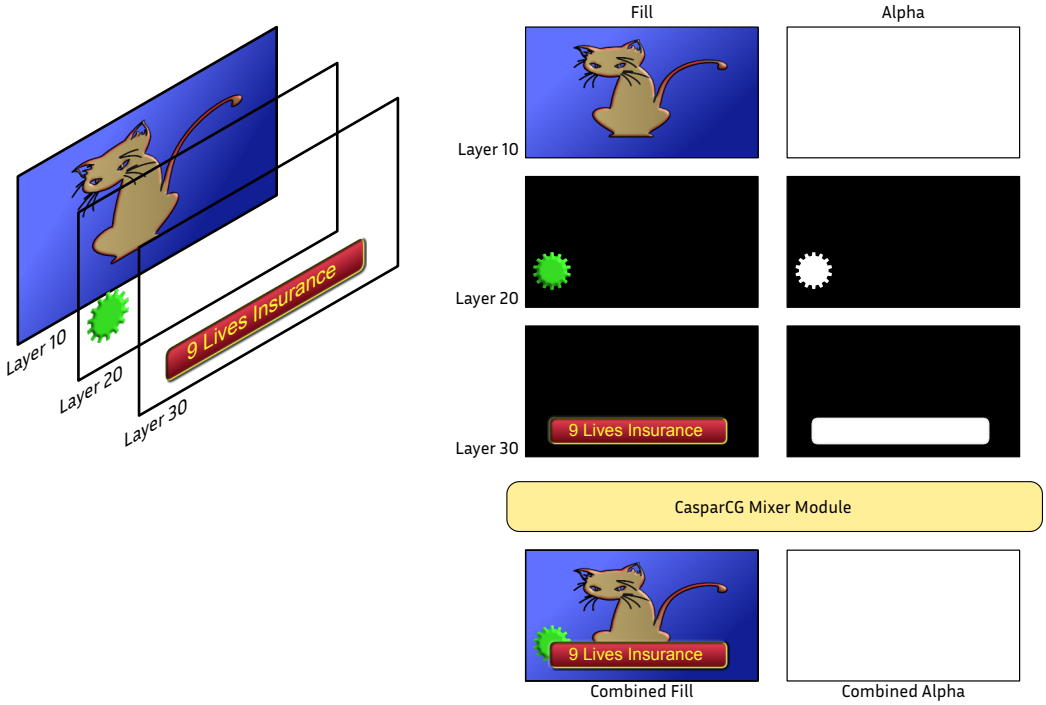
2. viewing a live input source on a designated channel;

3. sending an extra high resolution image to multiple output channels that spilt the image across the channel outputs to provide a video wall operation.

A producers frame buffers pass content to the ***mixer*** module. The mixer implements various image manipulations including size, position, brightness, contrast, opacity and saturation. The transforms are applied independently to each active layer of the channel. The manipulated layers are combined to create the composited output. The mixer module implements the combination using both alpha data for a layer and user-set blend modes (multiply, darken etc).

The mixer module includes an audio mixer enabling complex mixing processes for the various layers of a composited output.

Layers are numbered from 0 to 9999. Layer 0 is at the lowest level of the combining stack and 9999 at the top. The layering process is illustrated below.



The left side shows a graphic made from 3 layers. Layer 10 contains the picture of the cat, layer 20 has the cog-wheel, and a layer 30 contains a strap with overlay text. The graphics in layers 20 and 30 contain large transparent areas defined by their alpha channels. The fill and alpha contents for each layer and the resulting output are shown at the right-hand side of the illustration.

Although this example uses static elements, each layer can be a dynamic element, such as video, with an associated alpha channel. The alpha channel controls both overall visibility and transparency of elements in its application layer.

The outputs of the mixer are converted to channel outputs by ***consumers***.

The ***screen*** consumer creates a video display on the computer graphic display of the server computer. The display can be created in a moveable window, or set borderless full-screen on a defined output screen. This style of consumer can be used to drive a display embedded in a piece of scenery.

The ***system audio*** consumer directs the channel audio to the audio driver in the computer. This is convenient for development servers, but less use in a production installation. An operational hazard is system alert and alarm noises are also sent to the same audio output.

The ***Bluefish*** consumer is associated with a Bluefish444 SDI output card. There may be two outputs for a channel, one with the fill video and the second with the key enabling a lower-third caption to be keyed in a downstream vision mixer. A single card can provide the output for multiple channels.

The ***Decklink*** consumer is associated with a Blackmagic Design Decklink SDI board. There can be multiple SDI outputs for a single channel producing fill and key signals. Decklink cards commonly offer an HDMI output for direct connection to a video display. A single Decklink card can provide multiple channels of SDI output, or the card may be operated in split mode with some BNC connectors operating in output mode, and other BNC connectors used for inputs.

The ***file*** consumer sends the channel output to a disk file. The resulting file can be used in an edit session, or become the master recording for the studio.

The ***stream*** consumer supports output of a channel in an MPEG transport stream encapsulated in a UDP transport layer. This option must be used with care as MPEG coding, especially H264, is processor intensive.

The ***NDI*** consumer allows CasparCG to output a Newtek NDI stream to other equipment. The data rate for such an output is quite high, perhaps 150 Mbit/sec per output channel for a 1080i HD channel. Using the NDI stream output may require extra TCP/IP interfaces to support this operation.

Multiple consumers can be assigned to a single channel. For example, using a file consumer and a Decklink consumer allows the channel to be viewed whilst a recording is created. The consumers can be added and removed by AMCP commands from the CasparCG client.

A CasparCG server computer may be fitted with multiple physical outputs. The server must be informed which card output is used for a given CasparCG channel. The resource allocation uses a reserved file, **casparcg.config**, stored in the same folder as the server software. This file has sets of XML tags that define the locations for the media, the allocation of hardware to channels, the channel resolution, as well as the TCP/IP ports used for various operations.

```
<channels>
  <channel>
    <video-mode>1080i5000</video-mode>
    <consumers>
      <decklink>
         <device>1</device>
         <key-device>2</key-device>
         <embedded-audio>true</embedded-audio>
         <latency>default</latency>
         <keyer>external</keyer>
         <key-only>false</key-only>
         <buffer-depth>3</buffer-depth>
      </decklink>
    </consumers>
  </channel>
</channels>
```

A segment of casparcg.config for a single channel is shown in the box to the left. The section between the `<channel>` and `</channel>` tags defines the properties of the channel. This tag pair are the first entry inside the overall channel information container (<channels> </channels>) making this the channel definition for CasparCG channel 1.

The video mode tags define the resolution for the channel, in this example high-definition operating as 1080i/25. The number after the interlace definition letter uses four digits so that the NTSC related scan standards can be defined using an integer parameter.

The `consumers` tags surround the specification of all the consumers used for the channel. In this example a Decklink card is assigned to the channel using card number 1 for the fill and card number 2 for the key. The card numbers are enumerated in the Blackmagic Design Decklink configuration tool.

A ***CasparCG Client*** is any computer programme that sends ***AMCP control commands*** to one or more CasparCG servers.

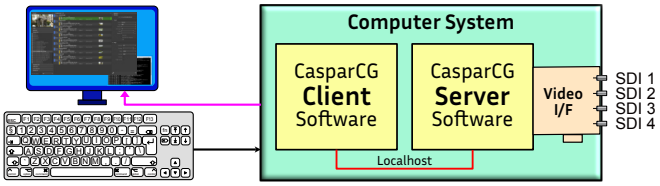
The server returns a status report to the client indicating success or failure of the command, or providing status information requested by the command. The server also outputs a stream of status data wrapped into Open Sound Control (OSC) messages and transmitted over UDP.

SVT provide an 'official' or 'standard' client for three operating system hosts - Windows, macOS, and Linux. Users can also develop their own control solutions known as ***custom clients***.

The SVT client uses the name ***rundown*** for a sequenced list of CasparCG commands. Rundowns can be stored on the client computer or in a shared network folder. Multiple rundowns can be loaded in a client instance, switching the active rundown is achieved by a single mouse click.

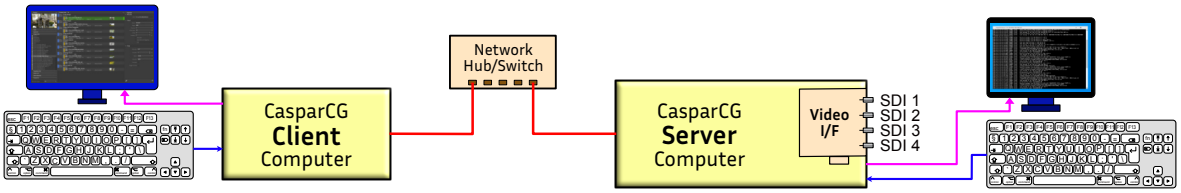
The operator steps through the elements in the rundown, issuing each command by selecting it in the list then pressing a function key such as F2 to start a playout operation, or F1 to stop the playout.

A single client instance can connect to multiple servers, and multiple client instances can connect to a single server. The combination of connections is established as needed to deliver the operational flexibility required.

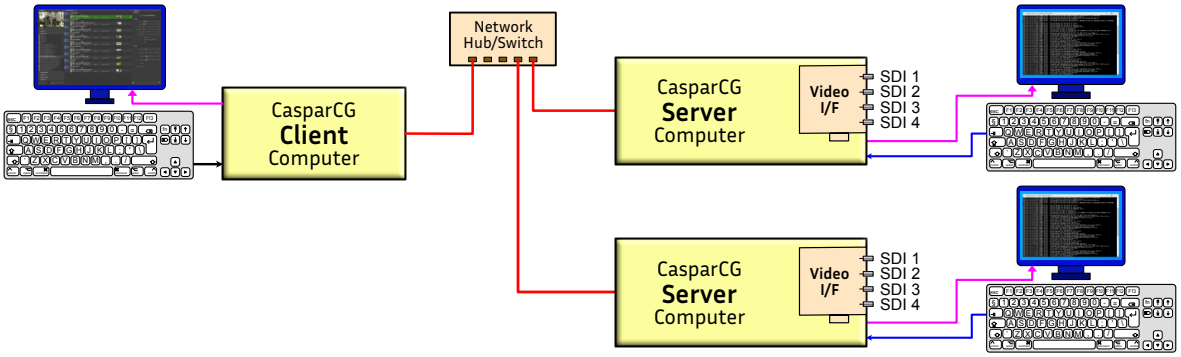


The simplest system uses a single computer running both the sever and the client as shown at the left. The control connection between the two software blocks uses localhost (127.0.0.1) networking.

A functionally equivalent operation can be implemented with the server software one one computer and the client software on a second machine as shown below.

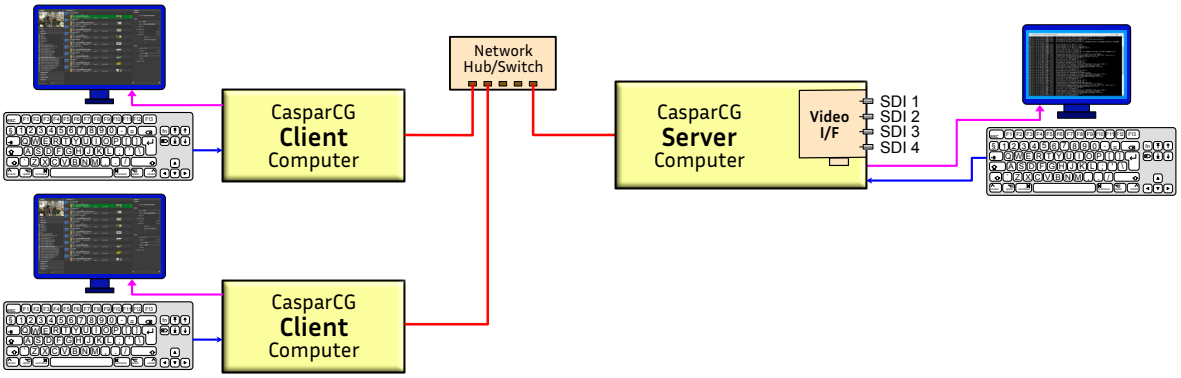


A single client connected to multiple servers is shown below. Using multiple servers can provide more output channels than a single computer, or servers can be paired to provide a main and reserve output.



Where servers operate as main and reserve, users must make suitable arrangements to ensure the media is mirrored to both servers.

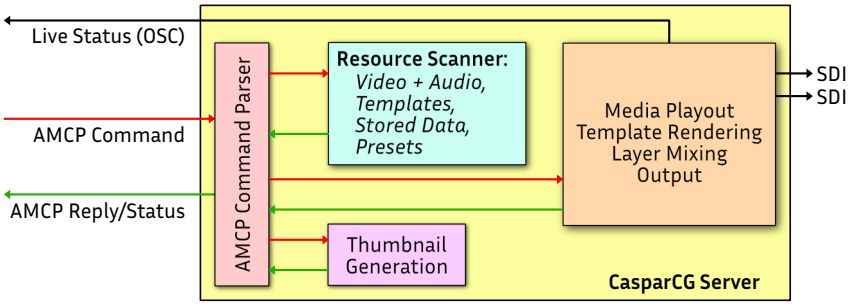
Multiple clients can connect to a single server. Supporting multiple connections to a single server allows one operator to manage graphics displays or lower thirds whilst a second operator manages clip replays.



For some news and sports applications one client can provide manual control of selected channel outputs, whilst a second custom client pulls data from a sports information server to update event statistics on a chosen channel output.

CasparCG Software Versions

CasparCG is open-source software. Multiple code branches exist because individuals or organisations have modified the code to provide the specific facilities they need. Many of the proven modifications are added to the core branch.



The overall server function is implemented by several processes. One process parses commands from clients, invoking the relevant processing. One process scans the media paths, returning lists required by the client. One process creates and manages the storage of thumbnail images used in the SVT client preview window. The primary playout process emits status reports to the client in response to user commands. It also emits a stream of status data using OSC (Open Sound Control) messages wrapped in UDP packets.

When this document was written there were two major code branches. Key differences between the branches are the processing of the thumbnails, the management of media lists, and the detailed structure of the OSC messages.

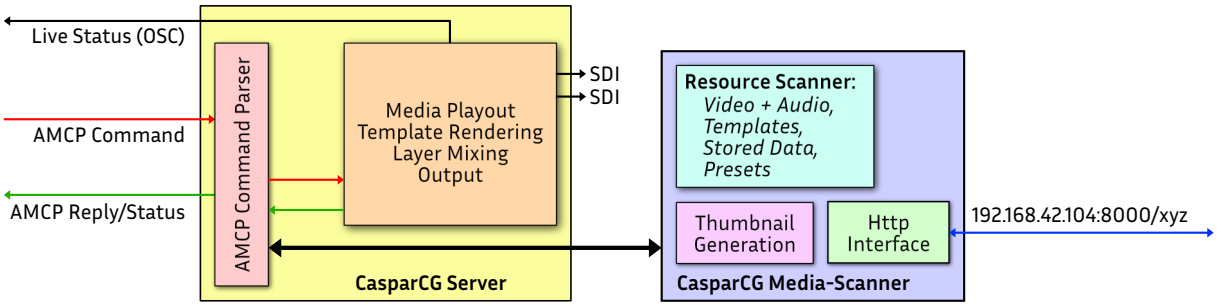
SVT Code Branch

When this document was written the SVT software versions were:

server: 2.3.3_LTS (LTS = Long Term Support)

matched client: 2.2

This version requires a computer graphics card supporting OpenGL version 4.5 or higher. SVT recommend *n*VIDIA Quadro graphics cards, although other graphics cards work.



The server software has been divided into two programs, one program provides the media scanning processes and thumbnail generation leaving the core server program to manage the playout processes. The core server uses the partner program to return lists of media requested by the client software. The media scanning element also supports user queries via a restful http interface. In the above diagram the computer running the CasparCG server system has a tcp/ip address of 192.168.42.104, and the media scanner program listens for traffic on port 8000. A limited set of http get requests are supported on this port including:

/tls return a list the templates

/cls list the media files

/cinf/<name> return information about the named file.

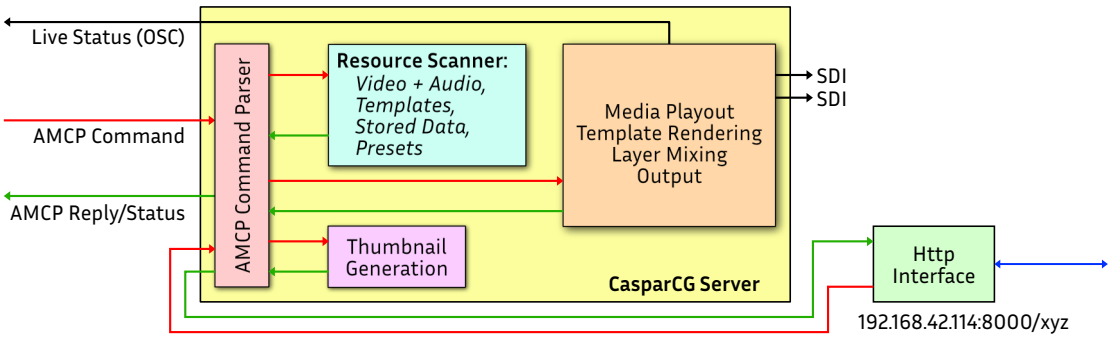
See https://github.com/CasparCG/media-scanner for a more detailed list of supported http endpoints.

NRK Code Branch

NRK created this code branch because they required time-based triggers that allow them to use CasparCG in their open-source *Sofie* news automation playout system. At the time of writing the current software versions were:

server:	2.1.12_NRK
matched client:	2.0.9

This version requires a server computer graphics card supporting OpenGL version 3.0 or higher, again *n*VIDIA Quadro cards recommended. There is a single executable that provides the media scanning and server functions. A media scanner program is available that hooks into the server and provides a restful http interface for media scanning. The process flow is illustrated below.



The NRK scanner module offers more http get endpoints than the LTS version of the server, mostly added in support of automation operations.

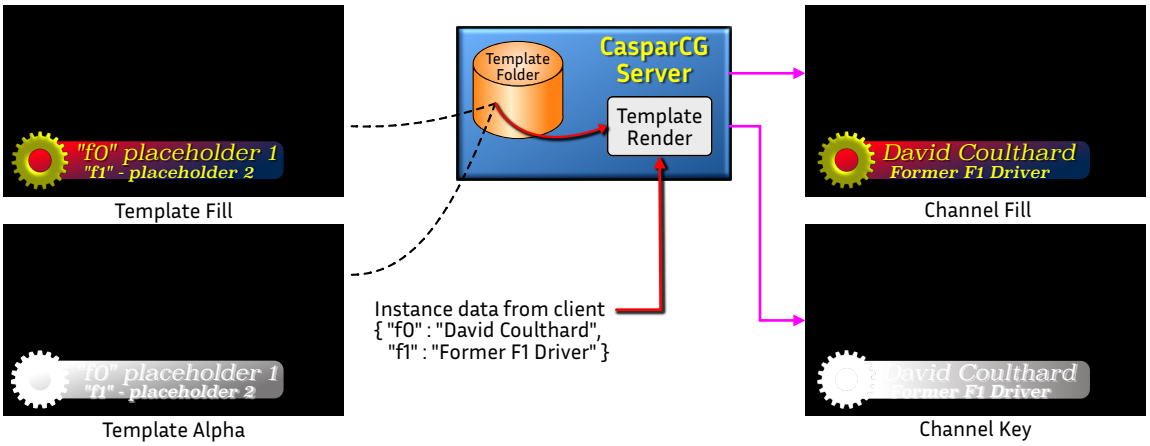
The list of endpoints is available at https://github.com/nrkno/tv-automation-media-scanner.

Templated Graphics

A ***graphics template*** defines a preset format for the presentation of graphic and text information. A template is converted into a rendered bitmap by combining the template format with ***instance*** data. The instance data replaces named placeholder fields within the template.

The rendered results can be designed as static or dynamic. Static templates are commonly used when an operator pushes a button on a vision mixer to control the times when graphics inserts occur. Movements in a dynamic graphic draw the viewers attention to the updates or new information. Both static and dynamic templates require a scripting language that manages the insertion of new or updated instance data into the placeholders within the template.

Early versions of CasparCG server used Adobe Flash with ActionScript (AS3) as the scripting language. Subsequently HTML template support was added using an embedded Chromium engine using Javascript (ECMA script) as the scripting language.



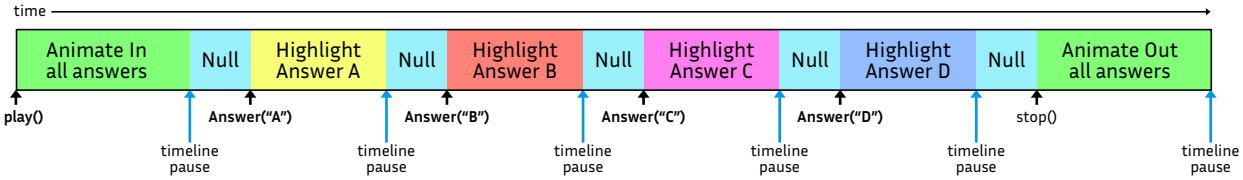
The placeholder fields in the above example are called **f0** and **f1**. Instance data is sent by the CasparCG client as part of a ***CG Play*** or ***CG Update*** command. Instance data is encapsulated in either an XML or JSON wrapper. In the above example the instance data is in a stringified JSON wrapper. A well-designed template auto-detects the encapsulation method and extracts the included instance data.

The template producer creates two bitmap outputs - fill and alpha (key). The template fill and alpha are used by the CasparCG internal mixer combining the template layer with other active layers in the channel, such as a video clip or further template layers. A composited fill and composited alpha/key are created for the channel. The two signals can be output as SDI signals via a Decklink or Bluefish consumer, enabling a keyer in a vision mixer to combine the graphics with live sources such as a camera output.

Each HTML template requires user-provided Javascript functions that process the commands and instance data from the client. Four Javascript functions are required:

Function Name	Action
play()	Called when a CG PLAY message is received from a client. The play function initiates the display of the template content. The action may be a cut or a timeline transition.
stop()	Called when a CG STOP message is received from a client. The stop function initiates the removal of the template content from the display.
update()	Called when either a CG PLAY or CG UPDATE are received from a client. When a CG PLAY command is received the update() function is called before the play() function. The CG UPDATE allows new instance data to be inserted whilst the static graphic content remains on display.
next()	Called when a CG NEXT command is received from a client. Moves the timeline playout point from the current position to the next keyframe position.
myFunctionA() myFunctionB() *[optional]*	The CG Invoke command allows a user-defined function to be called by the client. For example the movement speed of a graphic could be set via a a function named **rollspeed(new_speed)** where new_speed is a user provided value. The name of the function to call is passed as a parameter of the CG INVOKE command. Multiple invoked functions can be used in a template.

An example of the invoke function in operational use is a quiz show that displays a list of four possible answers. When the contest announces their choice, the invoke function is used to highlight the correct answer. An example timeline block for the quiz example is shown below. The null blocks are just spacing that permits later adjustment of animation durations.



When the operator presses the CasparCG standard client Play key (F2) the server process first calls the update() function to place the answers in their display elements. It then invokes the play() function to animate the question and answers onto display. At the end of the initial animate process the timeline auto-pauses playout.

When the contestant gives their answer, for example answer B, the CasparCG operator sends an invoke Answer("B") command to the server. The invoked Answer() function jumps the timeline playhead to the start of the Highlight Answer B segment and starts the animation, playing until it reaches the next pause mark.

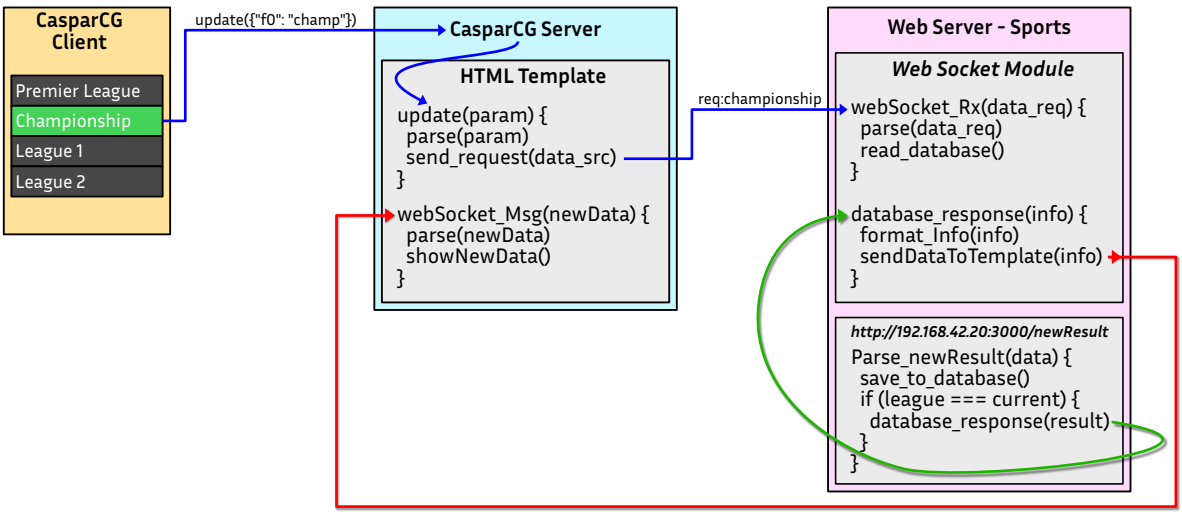
Finally, the operator sends a stop command to the server which invokes the Javascript stop() function. This animates the question and answers off the screen.

Template placeholder fields require identifying names. A news programme with a lower-third story ident may require two placeholder fields identified as ***Name*** and ***Designation***. When an operator instructs the SVT CasparCG client to add a identifier field name to the instance edit form it auto-inserts placeholder field names ***f0***, ***f1***, ***f2*** etc. These default field names can be over-typed with the user specified placeholder names. Many templates use the default idents to speed entry of instance data into the client entry form. The selected field names need documenting so that the graphic operator knows the field names to enter.

There are multiple methods available to design and code a template. These methods include manual coding of the HTML page, free-to-use visual layout tools such as ***Google Web Designer***, and commercial design tools such as ***Tumult Hype*** (macOS only), ***Adobe Animate***, and ***loopic***.

The visual editing packages support insertion of user-defined Javascript libraries or code blocks such as the CasparCG interface functions. Detail of how a template is authored is outside the scope of this document.

One feature of the Chromium HTML engine is built-in support for *WebSocket* technology. WebSockets allow the template to communicate directly with a web server. The data channel can be used both to request data from the web server or to push data from the web server. An example data flow is illustrated below.

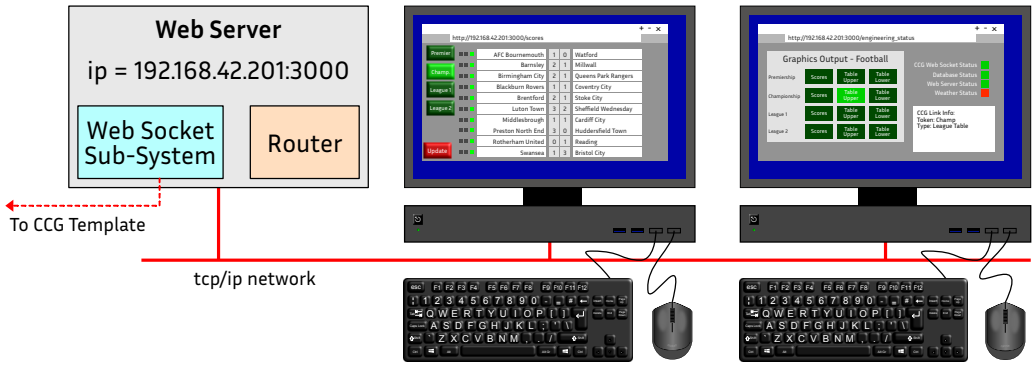


The illustration shows how a template can receive a request from a client, using the web socket system to request a large data set from a web server. This example fetches the football scores from the web server.

The CasparCG client issues a standard update command requesting a template data refresh. In this operation the instance data passed to the client is used as an index to request the actual data from the web server.

In the example, the parameter passed from the client has a field name of f0 and a value of "champ" (shorthand for championship). The CasparCG software receives the update command and passes the name and value to the update function. The update function uses the value to create a request for the actual data, sending the request to the web server via the web socket connection.

The web server reacts to the message initiating a request to fetch data from the results database. A database read operation is normally programmed as an asynchronous process. When data is returned from the database the processing code calls function database_response(info). This callback function formats the information ready for the template to use, using the web socket connection to send the data to the template. The template updates the placeholder fields with the new results data.



The web server includes a conventional html page server. A computer browser can connect to web server to load a page that allows manual entry of data via a form, for example by browsing to an html page at ***http://192.168.42.201:3000/scores***. The template and the web server can be programmed such that an updated result is pushed to the template for immediate display. The web server can also connect to other data sources via database queries or http *get* requests.

Other standard browser pages, for example ***http://192.168.42.201:3000/engineering_status***, can be configured to monitor the state of the template and web socket connection.

Some programmes, for example News, need a wide range of templates providing a consistent look and feel to all forms of name straps, story straps, live location identifiers etc. Rather than use lots of individual templates it is possible to use the power of the Javascript system to have a single template file that shows the element or elements required. Before the programme starts the template is played, providing an empty output.

All templated graphics are then controlled by the newsroom automation system. It sends a single command to the update process in the template, but that update can have several sub-fields separated by a printable character or character sequence that is never required in the graphic text. The template manages the displays such that if a two-line lower third is active and a single line display is needed the template does the remove old, display new transitions. Assume the field split id is set to use two up-arrow characters **^^**. The newsroom computer can request a two-line strap sending a command such as:

```
cg update 1-10 "{\"f0\":\"story^^Dick Whittington^^Lord Mayor\"}"
```

The Javascript update function checks the field id is "f0" then splits the value into sub-strings using **^^** as the divider. The key requirement for such control is a careful definition of the protocol that is used to select display elements and how the whole display or a sub-set can be cleared (for example removing a name strap but keeping the live location text).

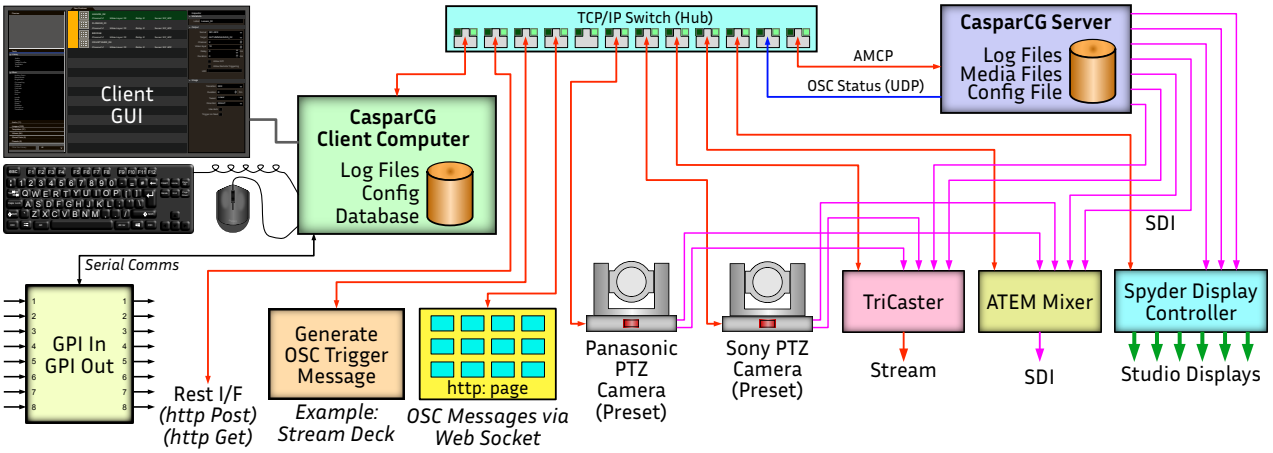
The control operation can be extended to use web socket connections between a controller and the template. When the template is initialised from a CasparCG client it connects to another process that also has web socket support. Web socket libraries are available for many programming languages facilitating custom control solution development.

Such a custom solution can enable a weather presenter to use a hardware push button to sequence through a selected set of displays, or a presenter to use a browser on a tablet computer to summon a wide range of graphics.

A simple, static displays, template is listed in Appendix H.

CasparCG SVT Client

The SVT client enables an operator to create and edit rundowns, and manage their replay. The client can control multiple CasparCG servers and also interact with other items of broadcast equipment. This control connectivity is illustrated below.



The client has a multi-column graphical user interface (GUI) where the centre column shows the list of items in the rundown. Playing items uses the keyboard function keys and cursor movement keys. It is operational best practice to set the client GUI to run at maximum screen size, ensuring that the client screen retains input focus for keyboard events.

The client software uses an SQLite database to store the configuration properties, such as the tcp/ip addresses of controlled items, and the thumbnail pictures of the video and stills media on connected CasparCG server units.

The client generates plain text logs of actions executed by the client, each entry is time stamped using the host PC clock time.

Controllable Hardware

CasparCG Server

The client establishes and maintains links to the CasparCG servers listed in the client configuration. The client continually monitors the link status and shows error indications on the client GUI if communications are not operating.

The client requests the list of media and template items stored on each linked server. Updates of the media lists can be done using manual requests or set to run at user-defined intervals. The client requests thumbnail pictures of any new media elements.

The client issues AMCP control messages to connected servers. The server checks the command syntax returning brief status data about the command (OK, Fail etc), or sending more extensive data such as the list of media file names.

The server emits a UDP data stream describing the current operational status of all channels in the server. The status is encoded into OSC (Open Sound Control) address structures. The OSC messages are sent to all registered clients, plus external recipients defined in the server configuration. The client uses the OSC data to show media progress bars for video clip playout. External OSC status processors can create status displays, for example on a web page, enabling a programme director to see the names of current media and their timeline progress.

GPI Control

Any item in the rundown can have GPI triggering enabled. The SVT client uses a serial connection to a GPI hardware interface, often implemented using an Arduino microcontroller. The client interface supports eight input contacts that are allocated to command functions as defined in the client configuration. The configuration sets the triggered action and active GPI transition edge for each input. The triggered action options include Stop, Play, Pause, Load, Next and Clear.

The rundown command set includes a GPI output instruction. This enables the client to trigger an external hardware unit such as a lighting desk or control an element on a vision mixer such as the DSK enable. The duration of the GPI output pulse is set in the client configuration. The GPI output data is sent via the serial link to the hardware interface.

OSC Control

When the client is installed or the configuration is reset Open Sound Control, OSC, message support is enabled. The client listens for OSC ***control*** messages, matching the address in the OSC message with the remote control enabled element address fields in the active rundown. An example control message from an external unit is:

```
/control/clip01/play 1
```

The user enters **clip01** into the UID address box of the rundown item to trigger. When the control message is received the CasparCG client will select the item with UID containing `clip01` address and issue the play command for that item.

This message address capability allows an external “Shot Box” interface to be used as the control device. Example hardware devices are the Elgato *Stream Deck* and the P.I. Engineering *X-Keys* units.

The client listens for OSC messages via multiple IP ports. One port, typically 6250, is for general server status messages. A second port, typically 3250, is used for control messages and a third port, typically 4250, is used for Web Socket transport. This Web Socket transport support enables a web page on a computer or tablet to send the OSC control messages direct to the client.

Pan Tilt Zoom Cameras

The SVT client has limited control of Pan Tilt Zoom cameras from Sony and Panasonic. The manufacturers PTZ control software or hardware is used to create multiple presets in the camera memory, including the speed of movement from current positions to the preset values. The SVT client instructs a camera to recall a numbered preset.

TriCaster

NewTek make a range of Vision mixers called ***TriCaster*** which make extensive use of software control. The SVT client can interact with a TriCaster to select sources on the preview or transmit banks, select a network video source, play a macro, and trigger a transition on the background or DSK keyers. These control facilities enable keying of lower thirds to be managed by the CasparCG operator when the vision mixer operator is busy on other programme demands.

ATEM Mixer

Blackmagic Design manufacture a range of ATEM vision mixers from simple desktop units to four mix/effect processors. The SVT client has controls for source selections, keyer action, playing macros and control audio properties such as gain and pan.

Spyder Videowall Processor

Christie Spyder units are processors that manage displays of pixel based graphics onto a linked set of displays forming a video wall. Some broadcasters use Spyder units to control the displays of graphic sources sent to video walls and displays in a news studio. Christie tools are used to create presets that the SVT client can recall.

OSC Output Control

The client can issue simple OSC control messages, ones that use a single parameter, to other equipment.

HTML GET and POST

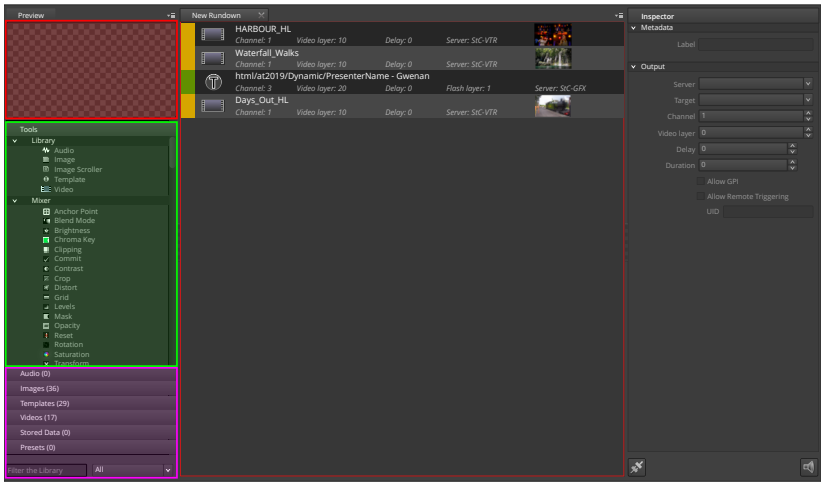
The CasparCG SVT client **POST** and **GET** client commands communicate information to an HTML server. Several equipment manufacturers offer remote control via a web browser. The POST or GET SVT client commands allow CasparCG client users to send remote control instructions to third party devices.

The manufacturer defines which messaging structure is used for the remote control operation of their equipment. Assume a device has multiple inputs, where the input selection can be set by remote control. One manufacturer may choose to use a POST message to select the input, a second manufacturer may choose to use a GET message to select the input.

Information flow is from the SVT client to the equipment. The SVT client has no mechanisms to display any messages returned by the restful control interface, but the response is logged in the client log file.

Client GUI

The SVT standard client is available for three host operating systems - Windows, macOS and Linux. The client uses a graphics support library from *The QT Company* ensuring the same “look and feel” on all operating system hosts. The user interface has three information columns.



The leftmost column displays information about media and playback control tools. This document has added the coloured highlight boxes to the picture to identify the segment functions within this column.

The red rectangle is the media preview display zone. The green rectangle shows the list of media and command tools. The magenta rectangle is the library listing selection and library search filters.

The centre column shows the rundown media and command list or lists. Rundowns can be saved or loaded from the ***client*** disc. Multiple rundown sets can be loaded in the client enabling swift selection of a set of commands by clicking the rundown name tab at the top of the column.

The rightmost column is the information inspector and editor. It displays item or group properties, and is used to edit property values such as the target output channel and video layer.

The left and right columns may be enlarged to expose more information edit space, or fully collapsed to increase the screen area available to the rundown column. The entire right side bar of the leftmost column, or left side bar of the rightmost column are dragged to alter the column width.

Hover over the column edge to be moved and the edge drag cursor appears (see icon at left).

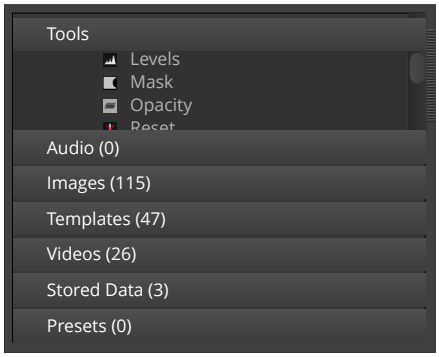
There is a minimum column display width, ensuring all core information is visible. Continuing to drag towards the edge of the screen causes the column width to collapse into a hidden mode. Even when collapsed, there is a knurled patch visible at half the client window height. The looks like the closely spaced lines shown to the left of this text. Minimising the library and inspector columns during playout is good operational practice. This aids keeping focus on the rundown and hence keystrokes are directed to the rundown control processing.

Preview for a Still or Video Clip

The **Preview** panel in the left-hand column is optional. Display of the preview window is controlled by a selection box in the general tab of the client configuration. When a user selects a still or video in the one of the library tabs, the preview window shows either a small version of the still image, or the poster (first) frame of the video clip as illustrated to the left.

The server system creates and stores the preview picture. The client requests a copy of each preview, which can be stored in the client database.

There is a small triangle adjacent to four horizontal lines in the top-right corner of the preview window. Clicking the arrow reveals a drop-down menu enabling the user to show or hide the preview window, and to choose fill or alpha channel preview mode.



The **Tools** tab lists all commands and playout control tools. The example to the left shows some of the layer mixer tools. Tools are available to load video clips, load stills, load html pages, load templates, adjust the mixer properties such as opacity, and control external units such as a Tricaster or an ATEM vision mixer.

The **Audio** tab lists the filenames for any audio files (such as *sting.wav*) in the CasparCG server media store. The preferred audio sample rate is 48kHz, but CasparCG server auto resamples 44.1 kHz CD audio to 48kHz enabling the simple replay of audio extracted from a CD.

The **Images** tab lists filenames recognised as still pictures. The commonly used image file extensions are BMP, PNG, TGA, TIFF and JPEG.

The **Templates** tab shows the graphic template names of files stored in the server templates folder. CasparCG supports two mechanisms for dynamic displays - Adobe Flash and HTML. Flash technology is end-of-life, and the flash producer is disabled when the server is installed. It is possible to download some code and re-enable flash where older templates are still required. HTML templates used in CasparCG include several javascript functions called when a template is played or stopped, and when the text content requires updates.

The **Videos** tab shows files in the CasparCG server media folder that are recognised as video clips. The name display does *not* include the file extension. Any file recognised by ffmpeg can be played in CasparCG provided it uses standard broadcast sizes and audio sample rates. If the audio is not sampled at 48 kHz, clicks, pops, bangs or silence may occur.

CasparCG supports use of alpha (key) channels in channel compositing and output. If a video or stills file inherently contains an alpha channel it is used by the layer mixer, and the alpha content is available to output on an SDI connection to a vision mixer. CasparCG also supports the alpha channel presented as a second file. It recognises the association via a file name convention requiring **_a** added to the name section of the alpha file. For example if the fill video file is called *myTitles.mov* the alpha channel file must be called *myTitles_**a**.mov*.

The **Stored Data** tab lists any data sets stored on the server. A lower-third name strap combines the design layout (the template) and instance data (the name to display). The instance data is sent as part of the CG_*play* or CG_*update* command to the server. It is easy for the client operator to enter and check small amounts of text such as a name or a headline.

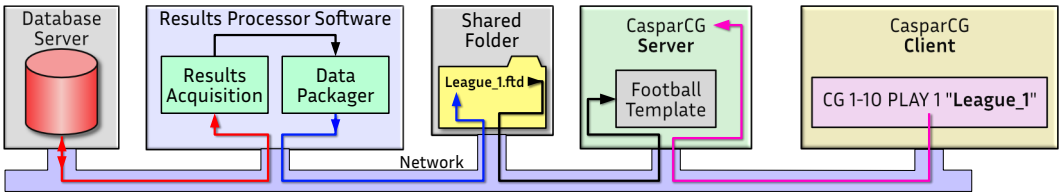
The Stored Data mechanism supports graphics that need large amounts of dynamic information, such as a set of football match teams and scores, and late delivery caption data. The caption text can be sent to a server using an AMCP command, which stores the data in the file name given in the AMCP command. For example:

```
DATA STORE news/headlines/worldcup "{\"f0\":\"WORLD CUP FINAL\"}"
```

Creates a file called **worldcup.ftd** in a sub-folder of the data store. The named sub-folder is created if it does not exist. The file contains the text:

{"f0": "WORLD CUP FINAL"}

A larger data set delivery is illustrated by the diagram below.

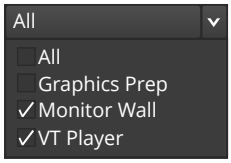


An external computer process gathers the data, often from a database, packaging it in a text file using the same wrapping process used for inline template instance data transfers. The packaged data is stored in a folder shared by the external computer and the CasparCG server, or transferred to the server using a DATA STORE command. The client sends a template "play" command to the server, passing the external data set name to the server.

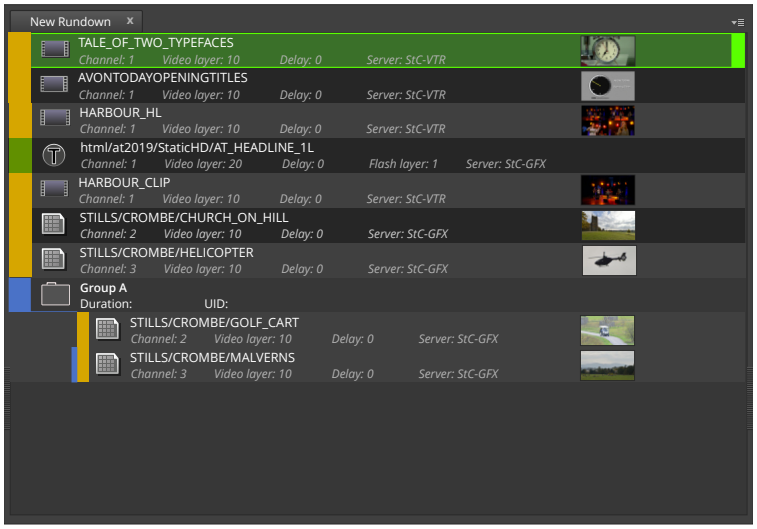
The **Presets** tab lists the names of user-saved command sets on the **client** computer. The presets feature supports recall of sets of AMCP commands, for example a headline sequence, for use in a rundown. Instance data, such as the clip name, can be updated in the rundown, but the proven sequence of commands in the preset is re-used to minimise potential entry errors.

Below the library content tabs is a **library filter** box. The example at the left of this text shows the box before filter string entry. When a string is typed into the filter box and the return key pressed, the displays of files in the tabbed lists show only file names that contain the filter text letters.

Assume there is a saved a set of files about rivers stored in a sub-folder called ***river***. The search text **river** will match that folder name, limiting the name lists to just that folder plus any other files containing the filter string. Case is ignored in the searches. To remove the filter either delete the string entry or click on the cross in the filter box, ***then*** press the return key to refresh the library listings.



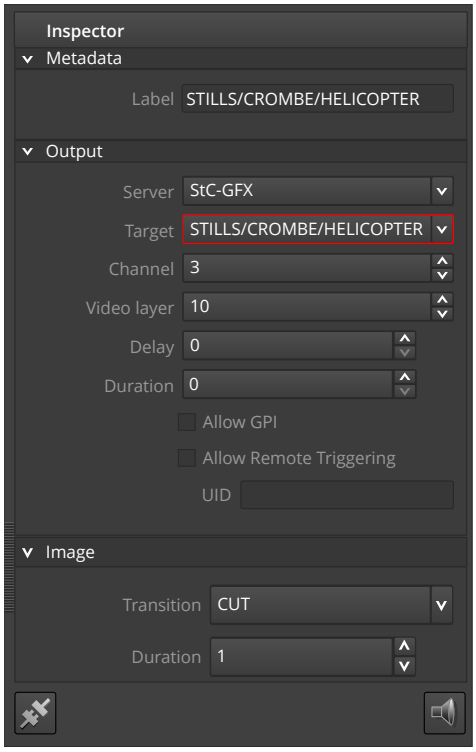
To the right of the filter string entry box is a second filter control. A single client can connect to multiple server units. The example at the left shows a client with three named servers. Currently the server filtering is set to show the library content of both the Monitor Wall server and the VT Player server.



The central panel of the client screen shows the contents of the currently active rundown.

A **Rundown** is a sequence of videos, stills, templates and control commands to play out under user control.

Rundown lists can be stored on the *client computer* disc for later recall and/or modification. Multiple rundowns can be open in the client, enabling the user to quickly switch between sequences via the named tabs at the top of the rundown list area.



The **Inspector** panel at the right-hand side of the client window is used to view and change item properties. The inspector shows the properties of the currently selected item in the rundown.

The **Output** section of the inspector shows the target CasparCG server name, channel and layer. The server is selected from a drop-down list of server names.

The server names shown in the drop-down list, and their associated IP addresses, are defined in the client configuration settings.

A stored rundown file includes the ***server name*** in the item data. Using identical server names in ***all*** client configuration files within an operational area allows the rundown file to be copied and run on a different client computer with no additional editing.

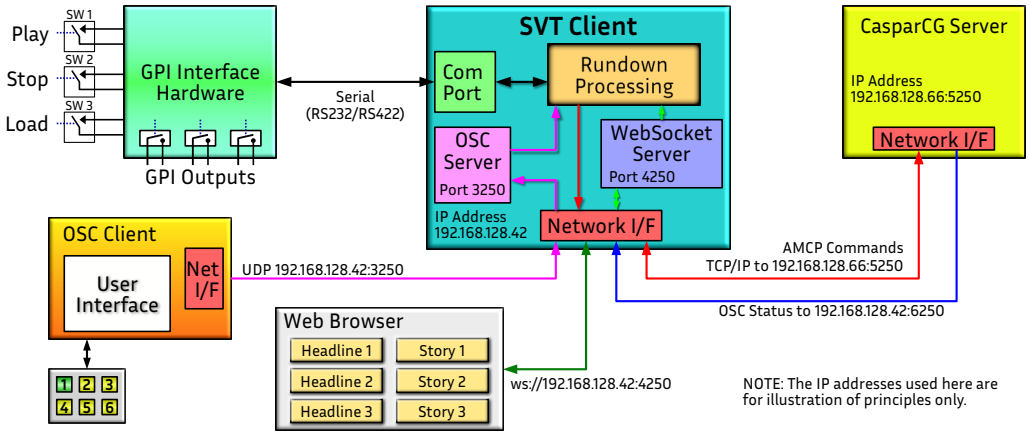
The **Target** box shows the name of the media item. It is a drop down box that displays a media list. The library filter controls affect the content of that displayed list. The **Channel** and **Video layer** properties were described earlier in this document. **Delay** sets the time interval between pressing an action button and when the command is sent to the server. This delay feature is used when building timed sequences of events within a group.

If the **Duration** is set to a non-zero value the media is removed from display after the time set in the duration box. Delay and duration entries can be configured to display either milliseconds or frames.

An event, such as play or stop, can be triggered using a keyboard function key on the client computer or by using an external control. The SVT client supports two external control mechanisms:

- Contact closure (GPI) - affects the current selected rundown item.
- OSC (Open Sound Control) messages. These can target a specific item for shot-box style operations, or remote control item selection and set the play/stop/pause state.

The external control options are illustrated in the diagram below.

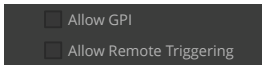


GPI contact closures are sensed in an external hardware unit such as a microcontroller. The contact close information is encoded into a short ASCII message sent across a serial connection to the SVT client. The client configuration defines the mapping between the GPI input number, the active transition edge, and the action initiated on the selected rundown item. The serial protocol is described in Appendix A.

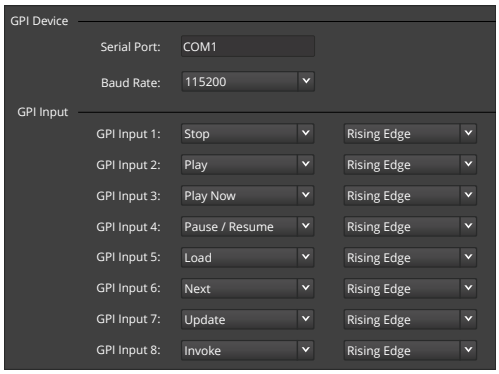
An external OSC client can send messages to an OSC server hosted by the SVT client. The default client configuration sets the UDP port number for OSC control traffic to port 3250 (client version 2.2). This port number can be changed by the client configuration. When a recognised message is received the message is passed to the Rundown Processing software for action. The OSC system and the CasparCG OSC message address schemes are described in Appendix B.

An Elgato Stream Deck LCD button unit combined with open-source BitFocus Companion software can be used as an OSC client. This supports both shot-box operations and sequential playout via the rundown.

Modern web-browsers support ***WebSockets***, a technology that provides a full-duplex high speed communication between client and server. The SVT CasparCG client includes a WebSocket Server enabling a browser and html page to connect to and send OSC messages to the client via the WebSocket connection. The OSC operation via websockets is described in Appendix B.



Rundown items that are intended to receive GPI triggers must be enabled by ticking an Allow GPI selector box in the Item Inspector.



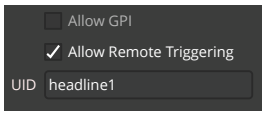
The default settings for both the GPI device and input actions are shown in the screen grab to the left.

The GPI configuration defines the active level transition and the action triggered by that transition. This allocation is universal on the client computer.

A simple microcontroller, such as an Arduino, can be used to convert physical button presses into the serial bit stream sent to the client computer.

There is a rundown command that triggers a GPI output bit. This provides a simple control trigger to external equipment such as a lighting chaser controller used on a quiz programme set, or actioning something on the vision mixer such as enabling the DSK.

The output GPI is connected via the same serial port used for GPI input triggering. The duration of the GPI output bit is configured in the client configuration, GPI tab.



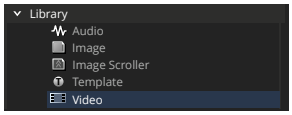
OSC control of an item is enabled by a tick in the Allow Remote Triggering box in the inspector. In the UID box enter the control name that the external OSC client will send. The software that sends the OSC command includes the chosen trigger name (headline1 in this example) as part of the control message:

```
/control/headline1/play 1
```

The 1 in the example command is an integer parameter. For many day-to-day operations it may be advisable to use very simple UID's such as a number. External trigger units do not then need reprogramming as different rundowns are selected for replay, just allocate the trigger ID in the SVT client item properties.

Adding Media and Commands to the Active Rundown

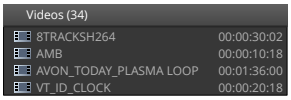
There are two primary methods to add media or other control commands elements into the rundown.



Method 1

Insert the base command from the Tools tab. The screen grab at the left shows the list of library media selection tools.

There are three options to add the command: drag and drop the item/tool into the rundown; double-click the item/tool; right-mouse click and select action from the context menu. Use the Target field in the item inspector to choose the media item. Set the target channel, target layer, and Metadata label.



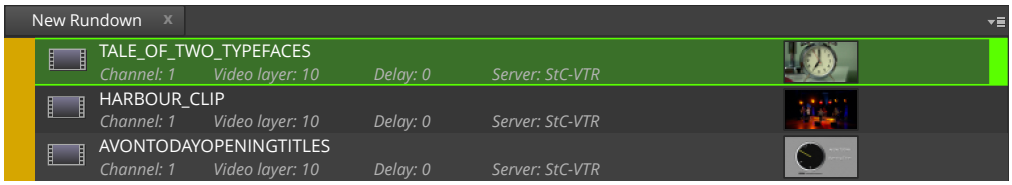
Method 2

Use the Library browser to locate the media item, checking the preview display to confirm the media choice.

Drag and drop the item/tool into the rundown; or double-click the item/tool; or use the right mouse click and context menu. Use the Target field in the item inspector to set the target channel and layer.

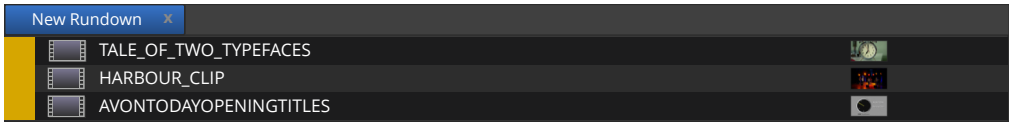
If items inserted at the wrong position in the rundown, their position can be moved by either mouse click and drag, or by key strokes (**Ctrl**+**Cursor Up Arrow** or **Ctrl**+**Cursor Down Arrow**).

Selecting a rundown item from the library automatically fills in the property inspector's **Target** field. The screen grab below shows a rundown after three items have been added. The topmost item 'TALE_OF_TWO_TYPEFACES' has been clicked to make it the active selection, showing it's properties in the inspector.



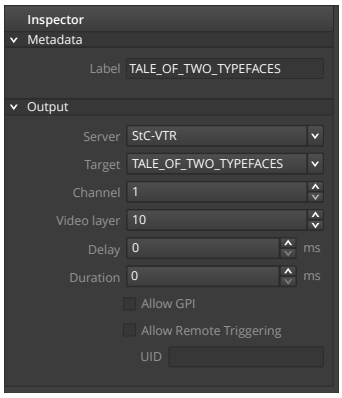
Summary properties are available in each item in the rundown list. As fields are edited in the inspector, information changes are reflected in the item rundown display. When a library double-click or drag and drop is used to add the item to the rundown the label display in the top line of the rundown entry is set to the clip name. When a raw tool is inserted to the rundown the tool type is shown in the label (Video, Image, Template etc). Use the Inspector Metadata Label field to change to an operationally more relevant name.

If the properties are not shown in the items of the rundown, the client is operating in *Compact View* as show below.



The viewing mode is toggled by the Toggle Compact View entry in the Rundown menu section of the client application menu.

The Item Inspector for a rundown item shows the file name, the allocated output channel, the video layer, the delay between pressing the command button and the command being issued, and the server that will receive the command.



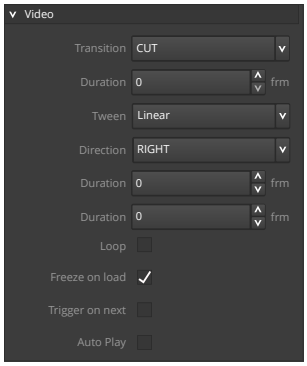
The Inspector display for the Tale_of_Two_Typefaces clip is shown at the left. The **Label** field is set as the clip name because the item was dragged from the library into the rundown. This field can be edited as may be required, for example providing an operator hint such as USE F6 (UPDATE) KEY.

The **Server** field is a drop down box. Select the target server from the list. The **Target** field is the item file name. For clips and images this is the file name to play, for templates it is the template file name to use.

The **Channel** and **Video layer** define which output channel and layer will display the media. Video and stills default to Channel 1 Layer 10, templates default to channel 1 layer 20.

The **Delay** value adjusts the time offset between the play command button being triggered, and when the command is sent to the server. When an individual item is played the delay field is normally set to 0. The client can group commands together (more details later) enabling an action such as Play to be sent to multiple items using a single keystroke.

An example of using the delay property is playing a clip and overlaying a lower-third caption credit. The clip item and the caption item are grouped, playing the clip in layer 10 and playing the caption in layer 20 of the chosen output channel. The video clip Delay is set to 0, and lower-third Delay is set at the time the caption should be shown, e.g. 5000 ms. When the group Play command is activated, the clip will start immediately, but the caption display is delayed by five seconds. The caption layer can also be set to stop after a user-selected time (Duration field).



The item properties section of the inspector changes slightly with item type. The example shown at the left is the property display for a video clip.

The **Transition** field has a drop-down list of the available transitions, including Cut, Mix, Push, Slide and Wipe. The **Duration** of the transition is set in ***frames***, not milliseconds. **NOTE**: with server version 2.3 LTS a fade duration of 0.5 seconds is entered as a value of 25 for a 50Hz interlaced or progressive channel.

The **Tween** field controls the transition profile. There is an extensive list of options, but the most used are linear, EaseInOutQuad and EaseInOutSine, the latter giving the flattened S shaped profile commonly used in vision mixer T-bar transitions. The available tweens are illustrated in Appendix C.

The **Seek** field allows the clip to be loaded with the chosen frame displayed as the first replay frame in the clip. A non-zero value in this field is commonly used to skip over a clock or ident at the start of a clip. The **Length** controls the number of frames that are played before the clip shows a freeze frame or re-loads (loop mode play). Server version 2.3 LTS requires numbers are entered as the number of *frames* for a progressive mode output channel, or as the number of *fields* for an interlaced mode output channel. Thus a seek of 1 second is entered as 50 on a 1080i/25 channel.

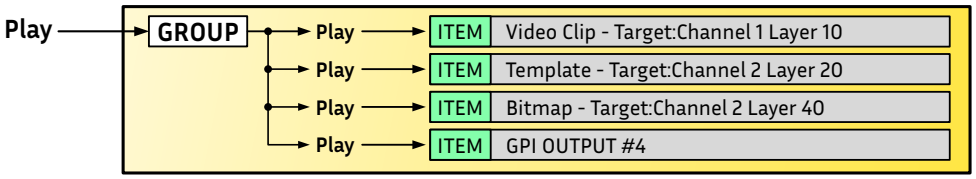
The **Freeze on load** tick box default setting is defined in the client configuration. When enabled, a video loaded with the F3 key will show the initial frame of the video. If the freeze on load is not active a blank frame is shown, with the video loaded in the background ready for play. Having the frozen still is helpful to many directors as they can see the clip is the one they want to play next.

The **Trigger on next** tick box changes the function key used to initiate play from the F2 key to F5 key.

The item **Auto play** tick box is disabled, unless the item is part of a group that has the group Auto play enabled.

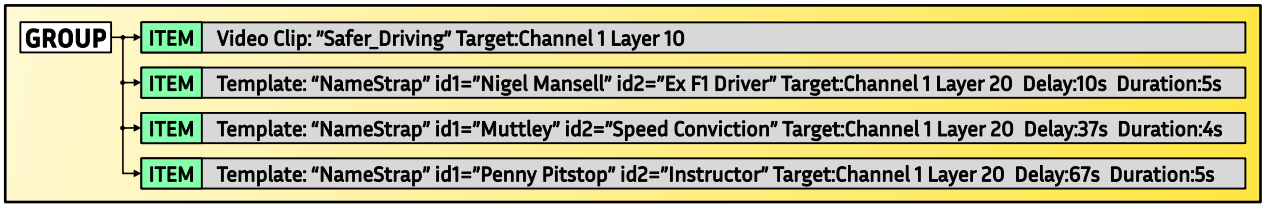
Groups

The group tool, as it's name implies, allows multiple commands to be treated as if there was just a single entity. A CasparCG command, such as **play [F2]**, directed at the group container is sent to all members of that group, as illustrated below.



The group-level command is sent sequentially to group items, with minimum delay between each command. Layer mixer commands have a **defer** tag. Commands with the defer flag asserted are stacked in the server until an **Execute Deferred** command is received. All deferred commands then start operation in the same blanking interval.

Combing a group container with delays set on the component enables sequenced playout of items or actions. Consider a clip replay that needs some lower-third credits shown during playout. An example of such a group command is illustrated below.

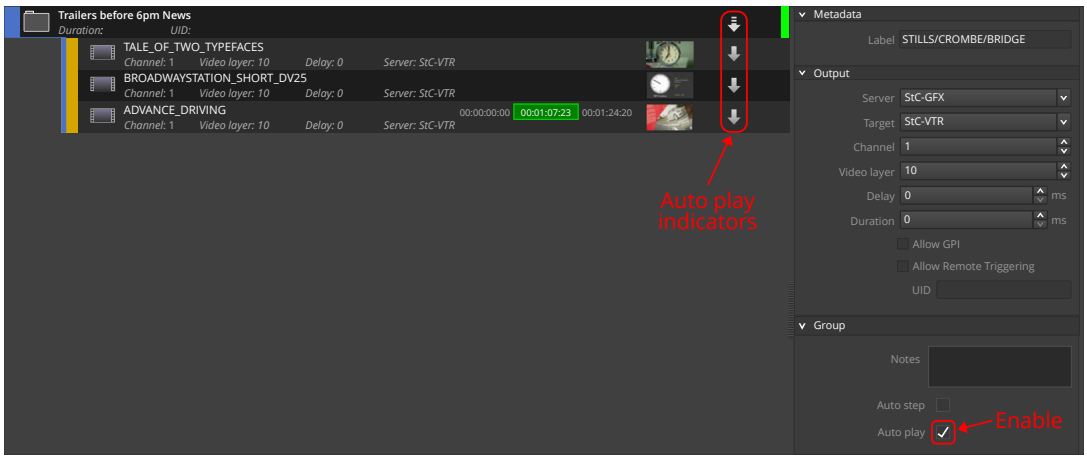


The video clip, item 1, starts playing as soon as the group play command is issued. The client waits 10 seconds before it issues a play command to item 2, a template called NameStrap that displays two lines of text. The text to display is sent as part of the play command. Five seconds later the client sends a remove (Stop) command to the same template. The same play and stop process is applied to items 3 and 4 at their specified offset times from the start of the clip.

The items in a group can be audio-visual commands, or device control triggers sent to external devices. The CasparCG client can issue GPI output commands, send an OSC command string, instruct a Panasonic or Sony Pan-Tilt-Zoom cameras to recall a numbered preset, and issue commands to ATEM and TriCaster vision mixing devices.

Group Auto Play

A group can be set to auto run a series of video clips using **Auto play**. An example of an auto-play enabled group is shown below.



All video clips are allocated to the ***same*** server and channel. The videos are added to the group, then Auto play mode is enabled for the group via the tick box in the Group Inspector. This sends the Auto play enabled flag to the members of the group.

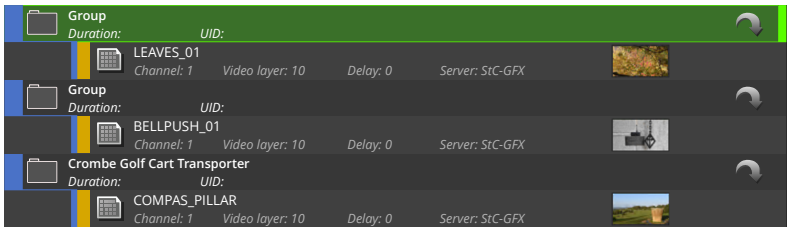
The SVT rundown client display shows Auto play mode is active by adding a large downward pointing arrow to the group box and to each item in the group.

When the group is selected and a Play action initiated, the client loads the first video in the list. The second video in the list is loaded in the background section of the channel. When the currently playing video reaches the end of file or reaches the set length, the background clip auto transitions to become the foreground clip. The transition settings on the background clip define the transition - Cut, Fade Wipe etc.

The client loads the next clip in the group list into the background section of the channel, and the auto play operations continues to the last clip in the group.

Group Auto Step

The Group facility also has an **Auto step** mode. The group is selected by mouse click or keyboard. When an action command, such as play, is issued on the group the command is executed by all elements in the group, then the next item in the rundown is selected. An example of the client with 3 groups set for Auto step is shown below.



The auto step enable is indicated by the looping arrow in the group header.

This mode allows the operator to step through a sequence of display commands without manually selecting the next item in the play list.

Creating a Group

Add one or more items to the rundown, either by double clicking or dragging a tool or media item. Select the items that are to be in the group using the mouse click and standard selection modifier keys (Shift and Alt).

Method 1

With the mouse above one of the selected items use a right click to show the context menu, selecting Group.

Method 2

Use **Ctrl**+**G** key combination.

The client creates a Group identifier (folder icon at the left) and shows the group members by adding a left-side indent to each item. It is best practice to name the group immediately after it has been created. Select the group header, then use the inspector to set the **Metadata Label** to a meaningful name.

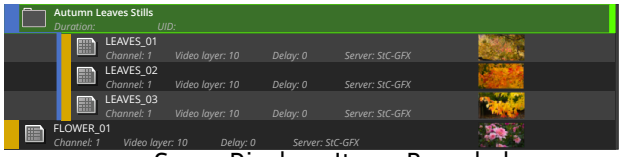
Deleting a Group

To delete the group and all items in the group, select the group then either use the Delete Key or right-click the group and select Remove from the context menu.

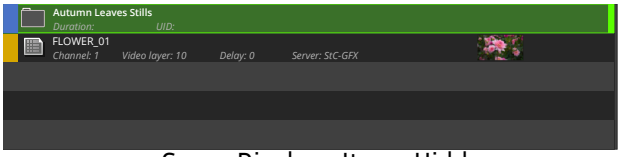
To delete *just* the group identifier and keep the items in the rundown, select the group then either use **Ctrl**+**U** key combination or right-click and select **Ungroup** from the context menu.

Expanding and Collapsing a Group Display

A group display can show the group header with all the grouped items, or just the group header. These display options are illustrated below in a rundown containing a group with 3 items followed by a stand-alone item. On the left we see all of the group contents, on the right we see the group collapsed to make maximum screen display space.



Group Display - Items Revealed



Group Display - Items Hidden

The expand and collapse operations are actioned using either the mouse or a keystroke. A mouse double-click over the group header *toggles* the expand status. With the group selected, **Cursor Left Arrow** collapses the display, **Cursor Right Arrow** expands the display.

Editing a Group

Add Item to Group

Method 1 - Using a mouse double click

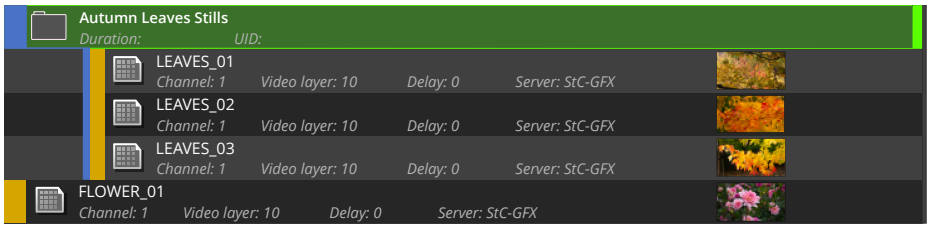
Expand the group if it is currently collapsed. Use the mouse to select an item in the group. Added content inserts below the currently selected item. Locate the item to add by browsing the tool or library list. Double click the item to add it to the group. Repeat the process if more items are to be added.

Method 2 - Using Drag and Drop

Expand the group if it is currently collapsed. Locate the item to add by browsing the tool or library list. Click and drag the new item over the current group item display. The new item is inserted immediately *after* the group element under the mouse pointer when the left mouse button is released.

Method 3 - Add the item Immediately below the Group

Select the item to add, such as FLOWER_01 in the example below, then use **Ctrl**+**Cursor Right Arrow** to add the item to the group.



Post-Addition Editing

The newly added item is auto-selected by the insert process, showing the item properties in the inspector. Set the item properties such as the channel, layer and transition mode.

Remove Item from Group

Method 1

Select the item using the mouse, drag the item below the group and release the mouse button.

Method 2

Select the item with the mouse cursor, right click to display the context menu and select Ungroup. The item moves to the line below the group.

Method 3

Select the item then use **Ctrl**+**Cursor Left Arrow**. The item moves to the rundown line below the group.

Moving items within the Rundown or a Group

Items can be moved to a new position using either the mouse or the keyboard.

Mouse: Select the item holding the left mouse button pressed whilst dragging the item to a new position. When the mouse button is released the item moves to the slot below the item under of the cursor pointer.

Keyboard: Select the item using mouse or cursor movement keys. Then use **Ctrl**+**Cursor Up Arrow** to move the item upwards, or **Ctrl**+**Cursor Down Arrow** to move the item down.

After any item is selected in the rundown the selection highlight can be moved up or down the rundown or group using the **Cursor Up Arrow** or **Cursor Down Arrow** keys.

Presets

Some programmes have complex video and graphics replay sequences used on almost every episode. Entering the media items into the rundown and editing their transition properties can be a relatively complex operation with all the testing needed. The ***Preset*** allows a proven item or block of items to be stored as a named entity. The preset is recalled for subsequent episodes, and the media names or template data fields edited to meet production requirements. But all of the transition properties, channel and layer allocations are retained from the recording of the preset.

Creating a Preset

Use the standard tools to create the list of media to playout, external triggers to be output, and instance data for templates. Prove the operations by rehearsing the sequence. Use the mouse to select the first item of the block that will be recorded as a preset. Move the mouse over the final item of the block, press the shift key then click the left mouse button to highlight the block.

Right-click the mouse over the block, selecting **Save as Preset...** from the context menu. Enter a name in the pop-up box and click the OK button.

Using a Preset

Highlight an item in the rundown. The recalled preset actions will add to the rundown immediately following the highlighted item.

Open the presets tab in the library window column. The list of names, filtered by the current filter text pattern, is displayed. If the required preset name is not shown check the filter fields, both the text and the list of servers.

Use your preferred method to copy the preset from the library list into the rundown - double click, drag and drop, or right-click and select **Add Item** from the context menu.

Select each item in the recalled preset, setting new targets and updating template data.

Deleting a Preset

Select the preset in the library. Either press the Delete key, or right-click and select Delete from the context menu. **Take care - there is no undo mechanism available**.

Archiving a Preset

Use the New Rundown item in the client file menu or use the Ctrl+N shortcut. The new rundown auto selects as the focus for new item entry. Recall the preset from the library.

Use the Save As... entry in the client file menu or the shortcut Shift Ctrl+S key combination. Select a folder then name the rundown and click Save. Local practice dictates the folder and rundown naming convention.

Copy a Preset to Another Client

Follow the process to archive the preset to be copied. Either use a shared network folder or a USB pen drive to copy the rundown content to the computer running the second client. Open the copied rundown on the target client computer, select all the elements of the rundown then create the named preset.

Playing Items in a Rundown

The selected rundown items can be activated by a mouse click in the client menu or by pressing the keyboard function keys. The function keys operation changes very slightly between media and templated character generation items. The key actions are summarised in the tables below.

Still and Video Replay

Key	Key Name	Description
F1	STOP	Immediately stop replay. Remove the video or still from the layer of channel.
F2	PLAY	Load the video or still if needed. Start playback actions using set transitions.
F3	LOAD	Load the video or still producer and await further instructions. When Freeze on load is set the target first frame is shown.
F4	PAUSE/ RESUME	Pause or resume replay.
F5	NEXT	When the **Freeze on load** and **Trigger on next** flags are active in a clip, this key starts playback.
F6	UPDATE	-
F7	INVOKE	-
F8	PREVIEW	If a preview channel is defined in the client configuration pressing the preview key shows the first frame or still on the designated preview channel.
F9	unused	-
F10	CLEAR	Clear layer as set by output control segment of media item.
F11	CLEAR LAYER	Remove and reset parameters on selected channel. Mixer parameters are **not** changed.
F12	CLEAR CHANNEL	Remove all sources of video from all layers in the channel. Reset mixer properties to default values.
Shift F2	PLAY NOW	Forces the delays entered into an item of a group to be ignored.

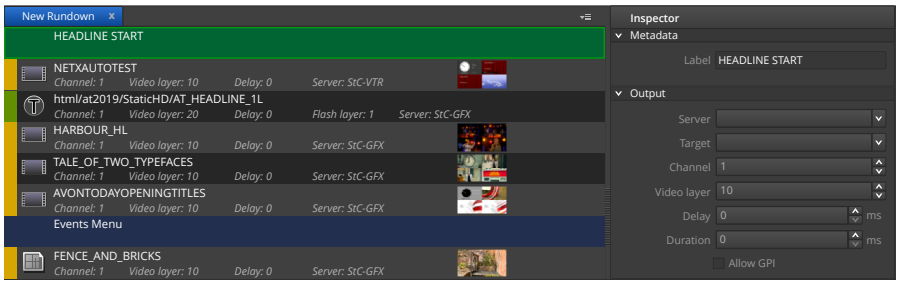
Templated Graphics

Key	Action	Description
F1	STOP	Fires any exit transition then removes the producer from the layer when out transition is complete.
F2	PLAY	Load character generation producer and template, then call the routine that runs opening animation.
F3	LOAD	-
F4	PAUSE/ RESUME	-
F5	NEXT	Start next stage of animation.
F6	UPDATE	Send new text to character generation template without using an input animate. It may use an animation to remove old text and add the new text.
F7	INVOKE	Calls a function in the character generation template. User provides function name and any parameters as part of the command.
F8	PREVIEW	-
F9	unused	-
F10	CLEAR	Clear
F11	CLEAR LAYER	Clear Video Layer
F12	CLEAR CHANNEL	Clear Channel
Shift F2	PLAY NOW	Forces the delays entered into an item of a group to be ignored.

Visually Divide a Rundown into Blocks

It is helpful to have a rundown divided into identifiable blocks - such as headlines, interview 1, music item etc. The CasparCG client supports named blocks via a coloured separator bar.

The separator bar is part of the **Tools Other...** command group. It can be accessed in the library Tools window, or via a right-click in the client rundown area. The separator bar defaults to a solid red background with no metadata. When the bar is added to a rundown any Metadata label text is shown in the separator.

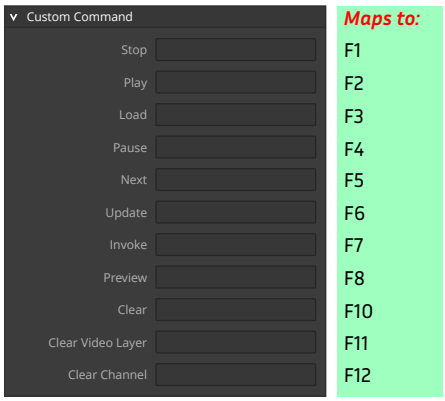


The example rundown at the left has two separator bars visible, one at the top and one near the bottom (dark blue background). The text in the first bar is a mirror of the Metadata label text. There is a limited set of colours available for the separator. The colour can be selected from the list available in the right-click **Colorize Item** context menu. The colour may also be applied to media or control items in the rundown.

Custom Commands

One of the challenges when playing through a rundown sequence is the need to use different keys to action some sections of the rundown. The **F2** function key can be used to load and immediately play a video clip, or play an already loaded clip. Many programme directors like to see the first frame on an insert clip before calling for it to be cut to the output of the vision mixer. This can require the CasparCG operator load the clip using **F3**, subsequently playing the clip using **F2**. Whilst this should not be a major issue for an experienced operator, new users may find swapping command keys more challenging.

The **custom command tool** provides a mechanism that enables a rundown sequence played using just one function key. The complexity of this solution is the rundown creator has to know some fine detail of the AMCP control protocol used by CasparCG.



The monochrome screen grab at the left shows the properties inspector for a newly added custom command. The dark boxes immediately to the right of the white text are where the AMCP command text is inserted.

The light green box to the right of the screen grab shows how the function keys map to the boxes. Function key F9 is **not** used.

The mapping enables any function key to be used to issue any AMCP command - thereby enabling the F2 key, normally used as a **play** command, to issue a **load** command.

Details of the AMCP language are available in the CasparCG online wiki at https://github.com/CasparCG/help/wiki/AMCP-Protocol

Each command description contains a mandatory element - the command verb. Required elements are shown in rectangular brackets **[ ]** and optional elements are enclosed in curly brackets **{ }**.

The parameter descriptions include the type of information that is expected in that field such as an integer or a string. An abbreviated version of the **play** command is:

```
PLAY [video_channel:int]{-[layer:int]|-0} {[clip:string]}
```

The commands are case *insensitive*, so play and PLAY are treated identically. The minimum version of the play command is:

```
play 1
```

which would cause a video file previously loaded on layer 0 of channel 1 to start playing. Why layer 0? The command syntax shows the verb and video channel must be present in the command to the server. The second element in the command is shown in curly brackets, and hence is optional:

```
{-[layer:int]|-0}
```

The vertical bar, `|`, just before the `-0` is shorthand for an **OR** function. If there is a layer number present, for example `15`, that value has precedence.

The second optional parameter is a string type and is the core name of the file to play. It is possible to use file names that contain space characters, but the spaces do occasionally cause issues. Better practice is to use underscores or hyphens in file names. This is because the command parser in the CasparCG server uses whitespace characters to detect the tokens present in the command line, so spaces in a file name might be interpreted as parameter separators, causing the file to be not found.

Bad	*Better*	*Best*
`Monday Story 001`	`Monday_Story-001`	`Mon20201207_StationOpening`

When a file name does contain spaces that file name in the command ***must*** be surrounded by quotation marks:

```
play 1-10 "Monday Story 001"
```

The play command can have extra parameters such as an instruction to LOOP the clip, seek a start point that is not the first frame etc. The extra parameter structure is common to three commands:

`loadbg` Loads a producer in the background layer and prepares file for playout.

`load` Loads a producer in the foreground layer and prepares file for playout.

`play` Moves clip from background to foreground and starts playing it.

The full syntax of the play command is:

```
PLAY [channel:int]{-[layer:int]} [clip:string] {[loop:LOOP]}
{[transition:CUT,MIX,PUSH,WIPE,SLIDE] [duration:int] {[tween:string]|linear}
{[direction:LEFT,RIGHT]|RIGHT}|CUT 0} {SEEK [frame:int]} {LENGTH [frames:int]}
{FILTER [filter:string]} {[auto:AUTO]}
```

To start a looping playback of file AMB.mp4 in channel 2 layer 10 starting with a 12 frame linear mix the command is:

```
play 2-10 amb loop mix 12
```

Using the same video mix in time, but starting at frame 100 then playing for 250 frames before restarting the loop the command becomes:

```
play 2-10 amb loop mix 12 seek 100 length 250
```

Appendix C shows graphical representations of the available transition **tween** functions.

The filter element of the command is a reference to a supported ffmpeg filter. Detail of filter functions is available at https://ffmpeg.org/ffmpeg-filters.html. An example of a simple command with a filter is:

```
play 2-10 amb vf hflip
```

The **vf** initials define the filter properties as video processing, and the single video filter applied is called **hflip** and this has no control parameters. The video is left-right reversed by the filter action.

The output picture can be both horizontally and vertically flipped using the command:

```
play 2-10 amb vf hflip,vflip
```

The comma between hflip and vflip delineates the filter elements where multiple entries are used. In the above example there is no white space between the filter elements. If space is used to visually separate the elements on the line the filter string must be enclosed in quotation marks:

```
play 2-10 amb vf "hflip, vflip"
```

Some ffmpeg filters support parameter values. An example is the average blur filter that requires the number of pels to use in the average operation. To set the filter at 6 pels the command is:

```
play 2-10 amb vf avgblur=6
```

Use a colon to separate multiple elements of a filter. The following example command draws a grid on top of the video, setting the separation of grid lines to 100 samples horizontally (w=100), 100 samples vertically (h=100), with a line thickness of 3, and selecting best contrast colour (c=invert):

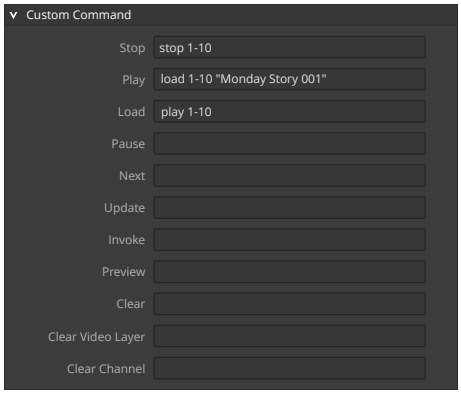
```
play 2-10 amb vf drawgrid=w=100:h=100:c=invert:t=3
```

Some filter commands can be very long strings. The custom command entry box scrolls if the entered string exceeds the screen display box width.

Audio filters are available, enabling various bandwidth control and pan operations. For example converting a mono source to a dual channel source:

```
play 2-10 myMonoVideo af stereotools=mpan=0
```

The name of file to play may be quite long, especially when logical named subfolders are present in the name. Rather than re-keying the string, use a copy and paste technique. Click on the wanted media item in the library listing, then use the Ctrl C shortcut key to copy the name. Paste the content of the target field into the filename position in the custom command.

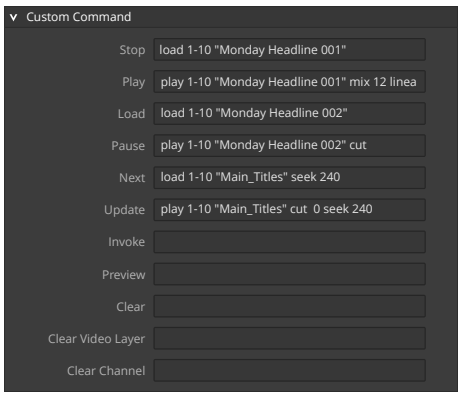


The example custom command show at the left illustrates how commands can be set in multiple slots.

In this example pressing the F1 function key will stop playout in channel 1, layer 10.

Pressing the F2 function key loads file “Monday Story 001” into the foreground of channel 1 layer 10.

Pressing the F3 function key plays any media loaded into channel 1 layer 10.



Custom commands can be entered into all of the available fields.

Hence it is possible to have a command that contains the load and play commands for a sequence of clips, each actioned by the associated function key, as illustrated at the left.

Custom commands are used when starting and stopping recordings of a channel. Setting up a recording is described in more detail in a later section.

Media and Template Storage

The CasparCG server configuration file includes three XML tokens whose values define the storage locations for audio-visual content, graphics templates and stored template instance data. The storage tokens are called media-path, template-path, and data-path. An illustrative example of the token values is shown below:

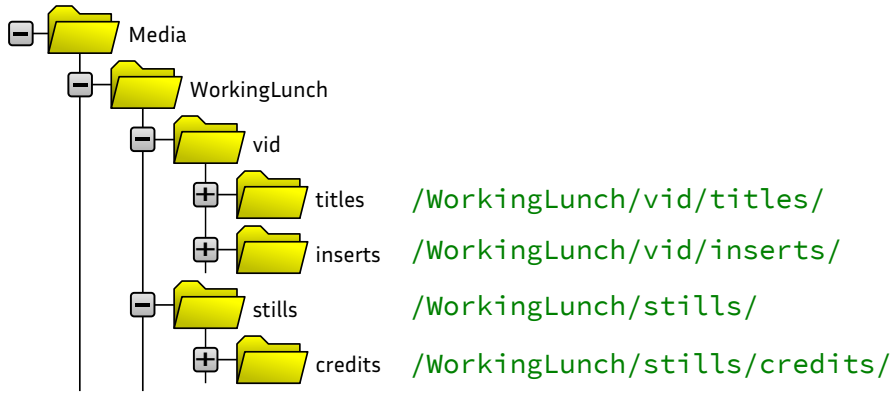
```
<media-path>m:/ccg/media/</media-path>
<template-path>t:/ccg/template</template-path>
<data-path>u:/ccg/data/</data-path>
```

The token values ***must*** end with a path separator. The above example shows different drive letters allocated to each type of information, but a modern computer using fast SSD devices can store all user media on the same drive.

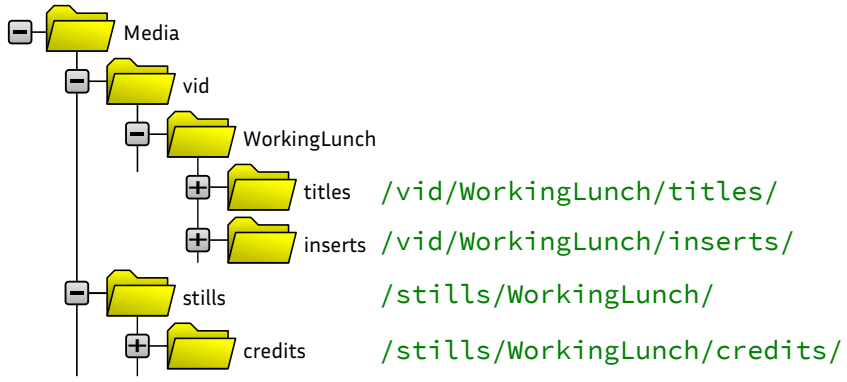
Audio clips, video clips and still images are accessed from the same base path defined in the <media-path> token. Video media needs a high-speed, low-latency store. The higher the resolution of the replayed content, the faster the read and/or write access required to the disc store. A Raid 0 array can be used to deliver the required speed and capacity. Solid State Drive units provide a relatively low-cost, high speed, high capacity store. Whilst the bus access speed is important, the actual read and write speeds to the disc elements are the critical elements in designing the store.

It is also important to have long-term archive storage. This archive must use robust back-up and management processes, and may require the use of a Media Asset Management (MAM) system. The archive storage capacity, technology, and data backup processes depend on user requirements. A small community television channel doesn't require the complex arrangements of a national broadcaster. When a CasparCG server is deployed in a large desktop or tower style case, a local archive drive may be available inside the chassis.

Locating and managing media elements in a CasparCG server is simplified by using sets of sub-folders within the main media store. Each sub-folder can contain further sub-folders grouping media according to functions or programme genre. Consider a multiple-channel CasparCG server in use for a daily magazine programme called ***Working Lunch***. One possible folder set is:



An equivalent folder set placing the media type at the highest level is:



Both example folder organisations enable simplified media searches in the SVT client by entering a phrase such as WorkingLunch/inserts or vid/WorkingLunch into the library search box. It may also be appropriate to use abbreviated folder names, for example WL, in place of WorkingLunch. The file names of media are shown in the client rundown list, so minimising the text length can prove helpful. Similar sub-folder organisation of the template folder simplifies archiving and searching operations.

Media transfer between archive and working storage can be implemented manually or use an automated process. A manual transfer process can use either a pull or a push transfer model. In a pull model, media is copied from the archive under the control of the CasparCG server host operating system. No folders in the CasparCG need to be shared on the network, but the computer console must be easily accessible to configure the transfer. In a push model, the working store folder structure is exposed as a network share, with the transfer of media controlled by the archive computer host.

A further option uses a watch folder on a shared drive. Specialist software on the CasparCG server runs a daemon that checks the shared folder contents. When media is found, it is queued for transfer to the correct store on the CasparCG, then the shared drive is instructed to move the copied source into a Copy Completed folder. The source file can either be deleted when the copy is complete, or moved to a transfer archive folder for later manual delete.

Media deletion in the CasparCG stores is normally a manually controlled operation, and may use a local archive for programme strands that are in an extended production break.

When using manual media transfers, how can a user find the root of the folder tree used for storage? Where the user has physical access to the CasparCG server computer, the information can be read from the casparcg.config XML file for that server. Alternately, if the user can access the CasparCG server console window, the command `INFO PATHS` lists the paths wrapped as XML data:

```
<paths>
   <media-path>m:/ccg/media/</media-path>
   <log-path>c:/ccg/log/</log-path>
   <data-path>u:/ccg/data/</data-path>
   <template-path>t:/ccg/template/</template-path>
   <initial-path>C:\CasparCG\ServerV2d3d3_LTS/</initial-path>
</paths>
```

If the CasparCG server computer is not local, the same info paths command can be run using a telnet client to connect to the server IP address on port 5250.

The mechanism used to copy new media into the server media folders varies to meet the needs of local computer and network security practice. It is important to maintain both data integrity and data security, for example running anti-virus software on all devices and files attached to the network.

The template store can use part of the main media drive, or be held on a separate storage volume. If the templates are predominately text based output, a slower storage medium may be adequate. Templates that use a large number of bitmaps or webm movies may require a high-speed store.

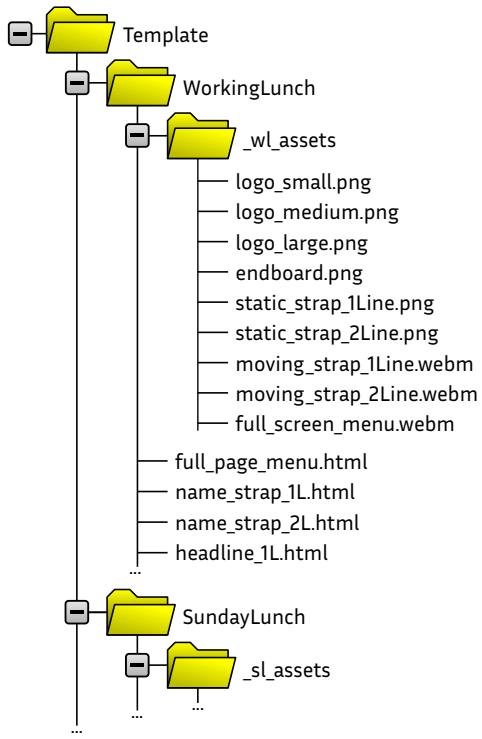
HTML Template Storage

Most new templates are now coded using HTML with Javascript. Very simple templates, such as a fixed colour strap with plain text overlays, can be coded to have all markup and script elements inline within the html file. Transferring such templates between a development system and the active playout systems is simple because there is just a single file per template. As stated above, it is still often desirable to collect templates for a single programme into an appropriately named folder.

As the complexity of the templates increases, it becomes more complex to keep the content within a single file. A template may need multiple bitmapped graphics, such as PNG or JPEG files, and webm encoded movies. As the javascript code increases in complexity users must decide if they want to maintain an integrated markup and scripting approach, or if they wish to use dynamic loading of scripts enabling code sharing between template instances.

When using multiple files it is still important to group resources into relevant named folders to create easy-to-implement transfer of templates between systems. One possible working practice is to use a folder structure that groups all resources for a set of templates into a sub-folder of the template folder.

Some template authoring mechanisms, such as Google Web Designer, include code libraries when the page or template is published by the authoring software. Such libraries are normally sharable between multiple templates, and all resources can be grouped into a single /assets sub-folder by a manual copy process.



The example at the left shows a programme name oriented set of storage folders for the media assets and html templates.

System implementers decide if the deployed CasparCG systems are allowed access to external internet web services. An example external resource is a content delivery network (cdn) service that delivers frequently used code libraries and typefaces. Providing external access minimises the local storage requirement, but has network security, loading latency variation, and library version control issues.

Access to internet resources is relatively easy to engineer at a fixed programme making site, but access may be much more complex and the access-speed somewhat variable when the CasparCG system is in use at an outside broadcast location.

Network access may also be required where a third-party data service, for example sports results are accessed for information that is formatted and passed to CasparCG for graphical display.

For some deployed systems it is more appropriate to deny external internet access, hosting copies of the code libraries on the CasparCG server as files, or running a local web service.

Still Images

CasparCG Server can replay many types of still images. Five commonly used types are PNG, JPEG, BMP, TIFF and TGA.

The CasparCG Image replay software scales an input still to fit the active channel output dimensions. Ideally the source graphic resolution matches the channel resolution. The re-sizing of images in the producer can create aliasing artefacts, so check the still output quality on a transmission grade monitor, or use a conform process to create a still with ideal sizing for the channel.

A standard definition output channel has two options for still sizes – the “square pixel” sizing or the native channel resolution. A 16:9 PAL output channel can use 1024 by 576 (square pixel) or 720 x 576 (native). A 4:3 PAL output channel can use 768 x 576 samples (square pixel) or 720 x 576 (native).

High resolution images are usually fast to recall. Images with very high resolution, for example more than 4000 samples per line, add visible delay between pressing the play key on the keyboard and seeing the image on the output of a standard or high-definition channel.

Best workflow practice is to conform a source still, such as one from a DSLR camera or phone camera, to the target shape (e.g., 16:9) using cropping, scaling the image to the channel target resolution. Save the file as a PNG or JPEG image, but do not use spaces in the file name.

CasparCG and Alpha channels

The CasparCG system is designed to support alpha channels in all replay media compositing operations. The alpha channel is activated by either of two mechanisms:

1. Using a file format that supports an embedded alpha channel. Example formats are ProRes codec in a MOV wrapper for video, and PNG and TGA for still images.

2. Using a second file with **_a** added to the primary name. Given two files, one called wilty_titles.mov and the second called wilty_titles_a.mov, CasparCG auto-selects wilty_titles_a.mov file as the alpha channel for wilty_titles.mov. The file name association also works for still images that do not include an alpha channel.

Media Scanner Operations in Server 2.3.3_LTS

CasparCG server version 2.2 and later use a media scanning tool that runs as a separate programme. The server and scanner processes communicate via localhost network connection. The media scanner maintains a database that is physically stored in the CasparCG server distribution folder.

When new media files are copied to the store whilst the scanner is offline, that media is normally found and added to the local database when the scanner process is started. For this detection process to work, the program startup order is important. The scanner must be started first, followed by the CasparCG server. This is the sequence set in the batch file distributed as part of the server code. A thumbnail picture is also created for the new audio-visual media and stored in the database. When an SVT client connects to the server, it copies new thumbnail(s) into it's own client database.

If the scanner is running when a Windows file delete is used to remove media, the scanner detects the delete operation, and updates the database accordingly. If a client is connected, it is informed about the file delete, and it removes the file and the thumbnail from the client database.

Unfortunately, media that is deleted from the media folder when the scanner is ***not*** running might not be removed from the media database. In such instances a CasparCG administrator or user must follow a simple procedure to rebuild the server and client databases. The procedure is documented in Appendix F.

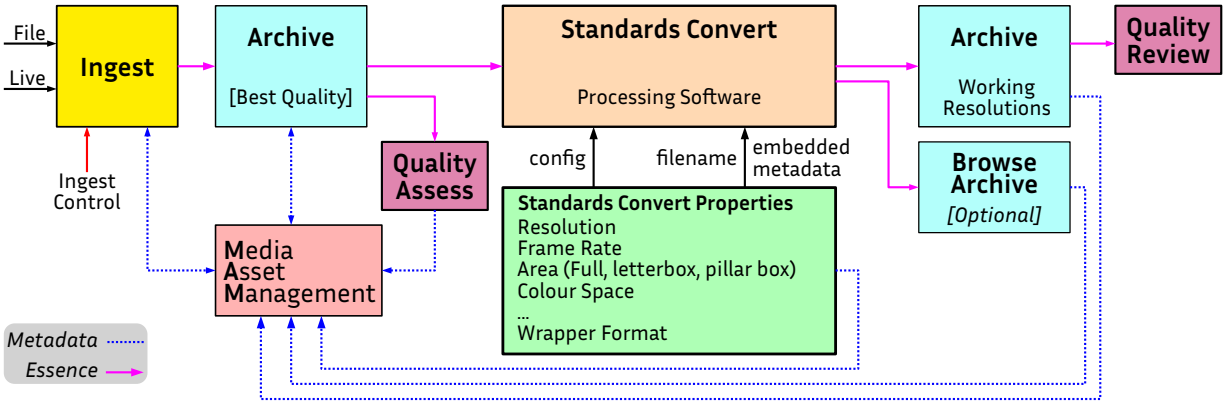
Media Conforming

The number of television scan standards has grown significantly in recent years, adding new standards for spatial and temporal resolution. The ffmpeg process used in CasparCG server automatically performs some conversion processes "on-the-fly" as media is replayed. High-definition high-frame rate source material, such as 1080 line 50 frames per second content, can be directly played in a standard definition PAL channel. CasparCG server has to balance ultimate engineering signal quality against the available time for re-processing, and uses a set of options that achieve a real-time result.

Production picture aspect ratios have also changed. Film used *Academy Format* for many years, a 4:3 ratio that was also adopted by television. TV used both film media licensed from producers, and as a direct production format for both full programme and programme inserts. As film stock capabilities improved, movie production studios experimented with other aspect rations targeted at enhancing viewer immersion, including ultra-widescreen options such as cinemascope.

Playout computational load can be minimised and transcode quality maximised by using an ***off-line conform*** process.

The best quality version of the media is acquired and stored in an archive ***master store***. If a MAM (Media Asset Management) system is used, the technical and production metadata for the master copy is entered into the MAM.



The above diagram illustrates a generic workflow for a media conform process. The best quality version of the media is acquired from either a file or via live-capture. Any technical or production metadata in the master is copied to the MAM system.

The complexity of the MAM is chosen to be appropriate to the playout operation. This may vary between using carefully chosen file names through to a full model relational database incorporating a large set of data fields, for example implementing the EBU SMEF model (see https://tech.ebu.ch/docs/tech/tech3293.pdf for SMEF information).

A quality assessment of the master file is run, using "eyeballing", specialist analysis software, or both. Many properties need to be validated, including video level ranges, consistency of colour (grading), audio audibility and audio loudness, and potential to trigger photo-sensitive epilepsy. Results of the quality assessment are stored in the MAM.

Ahead of first transmission, a software standards convert operation is applied to the master file, creating one or more versions at the working resolutions of the output channels. This conversion standardises all media properties, such as how 16:9 widescreen source material is converted for transmission on a 4:3 transmission channel, the type of file wrapper used, colour space, frame rate etc. The processed output file is stored in the working area of the media archive, and relevant metadata for the version added to the MAM. A further quality review may be commonly required, checking for issues such as aliasing caused by filter quality settings used in the processing.

Some users also find it useful to hold a lower-resolution version of the media for off-line browse operations, including finding shots or sequences to be used in programme promotion, or that are usable for editorial quality checks and sign-off.

There is a wide-range of options for the standards convert software. Many users find *ffmpeg* can deliver their requirements by using a careful selection of command line switches and filter properties. There are also several commercial conversion packages that run as automated background processes. These packages scan an inbox for files, processing any new files according to user-set conversion rules.

Stills media should also use a conform process that maintains the best quality version of an image in the master archive, but creates working resolution stills for the playout channels. This conform operation can be implemented manually through any bitmap edit program. The MAM data for a still should include both the media source and copyright ownership information.

CasparCG server version 2.3 software auto-conforms the audio sample rate to 48 kHz during playout if it is not already at that standard broadcast sample rate. It can still be advantageous to perform the convert operation off-line to allow testing for unwanted aliasing issues.

Day to Day Operations

This section examines building and editing rundowns for day-to-day operational work in a production studio or outside broadcast. The mechanics of adding media to a rundown and editing the properties are described in an earlier chapter about the SVT client.

A typical programme requires both pre-programme events and programme transmission/recording events. The pre-programme phase prepares and starts in-vision moving graphic backgrounds etc., and may output a programme ident slate to identify any recording.

Building a New Rundown

Programmes require a combination of moving video, stills, and templated graphics. CasparCG server supports playout of all these media types. External devices may also need triggering as part of the playback actions.

Creating the first rundown instance for a programme requires at least one copy of the programme script.

Work through the pages of the script, adding notes that identify all video clips, stills and captions. Pay particular attention to the output channel number allocated for each media item. Include that allocation on the marked-up script. Where video clips do not yet exist, for example news items still being edited, select a standard media item as a placeholder in the rundown.

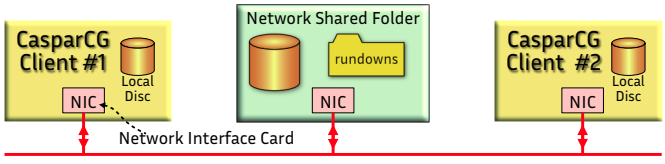
Identify the graphic templates that are needed. If new templates are required, allow sufficient development time for template creation. It is helpful to maintain a document that shows examples of each template in use, including the field names used in the template.

Once all available media elements are stored in the CasparCG server(s), start the SVT client and a new rundown tab is created. If the client is already active, create a blank rundown by selecting **New Rundown** in the SVT client **File** menu, or by using the shortcut keyboard command **Ctrl N**. Refresh the client library list using the menu or **Ctrl R** shortcut.

Most of the detail of how to add media items to the rundown were described earlier in this document in the section called “Adding Media and Commands to the Active Rundown”. Work through the marked-up script adding all required elements to the rundown. Most operators find the simplest way to build a video section of a rundown is using the library search facilities where the media preview panel helps confirm the media selection.

If the rundown might be controlled by an external process, such as a web browser or OSC client, use the main menu bar Rundown sub-menu to tick Allow Remote Triggering. A red lightening flash is added to the rundown name tab showing the trigger enabled state.

Finally, save the rundown using the **Save As...** Item in the File menu, or use the shortcut key combination of **Shift Ctrl S**. Select an appropriate name and folder location to store the rundown.



Sometimes more than one client requires access to the rundown. An example is where an integrated rundown was created, but two operators will control the replays - one managing clip replays, the other managing graphics replay. Using a network accessible store enables all interested clients have access to the single resource. Working copies can be stored on the client computer hard drive.

Name	Address	Port	Description
StC_GFX	192.168.42.21	5250	Studio C Graphics Server
StC_VTR	192.168.42.31	5250	Studio C VTR Server

The name of the content source server for each item in the rundown is stored as part of every item in the rundown. Sharing a rundown on multiple clients is thus most effective where the all client configurations use the ***same names*** for the controlled servers. The logical name to IP address lookup table is defined in

the client configuration screens. An illustrative example is shown above. The elements that need to match are the Name column and the associated Address and Port column.

Play through the items, checking the intended media or template are displayed on the allocated target channel using the assigned layer. Store the edited and checked rundown using the **File Save** menu element or use the shortcut **Ctrl S**.

Editing Video Clip Properties in a Rundown

The full list of files on the sever media store can be long, needing much scrolling of the list to find the required item. Using the library filter tools for media name and CasparCG server selections limits the media list size. After the clips are found and added to the rundown, clear the library filters.

When a video clip is added to a rundown, the default selections set the first frame of the clip as the first replayed frame. If the clip includes a countdown ident clock, as commonly specified for network programme delivery, the first replayed frame can be set as the start of the actual content by entering a value into clip item inspector **seek** field.

Using a drag and drop or double click to add the clip to the rundown automatically sets the name of the server in the **Target** field of the inspector.

Set the output channel *number* to match the programme script allocation. Set the channel *layer* to be used. The default layer for video clips is 10, and this works well for simple replays. Enable or disable the Freeze on load box in the Video segment of the item inspector to match local working practices. The client configuration can be set such that freeze on load flag is set true when a clip is added to the rundown.

The clip title is automatically shown in the client item Label field. This label can be changed using the Inspector. It may be helpful to add operator hints into the label field to give a clearer description of the function of the item; for example "Main Title", "Menu Item 1" etc.

It may also be helpful to add labelled colour separation bars that divide the media into logical blocks.

Still Images in a Rundown

Some programmes use sequences of still images to create an 'events' menu, or that form a set of end credit boards. Still images are added to the rundown using the same processes used for video clips, but selecting the Images segment of the library listing.

Some directors like to see a preview of a still before it is added to the transmission output, where other directors trust the graphics operator. Where sufficient channels of CasparCG output are available the still preview function can be incorporated into the rundown listings.

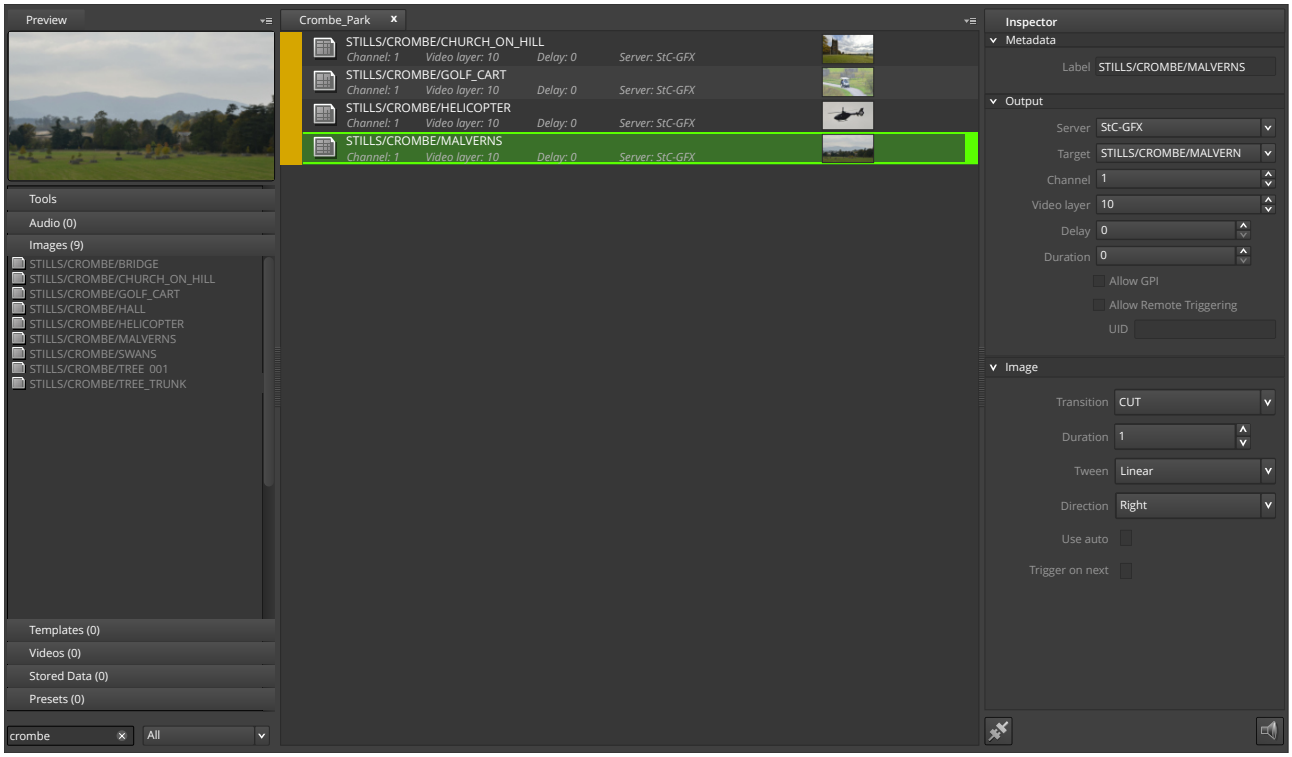
When a transition between stills is used in conjunction with a preview output, it is good practice to leave the old preview image on display until inter-still any transition has completed. Then cut the new preview still into the allocated preview output. Such a change is "seen" by the directors and vision mixers brains, even though they may not be directly looking at the preview display. The director and vision mixer then "know" the next still is ready for use.

Stills Replay using only Transmission Output

In the left-hand column of the client, select the Images tab. Enter or clear the filter string as appropriate. Careful management of the server media sub-folders creates library selection boxes that need minimal scrolling of the library window.

Selecting a still in the Images list usually shows the still in the client Preview window. If the preview picture does not appear this is because the preview for that still has not yet been created at the server and sent to the client. If a still has been added to the media store, but the name is not visible in the client library list this may be because the library listing needs a refresh. Use the Ctrl R key combination or the menu Library Refresh item to initiate a client update.

After some images have been selected the client window may be similar to the example below.



As each still is added to the rundown it is allocated a default channel number, video layer, transition type and tween type. Edit the channel number and layer to the values required for stills output on the active CasparCG server, and set the transition and tween properties.

Images can be added to the rundown in any order. It is simple to re-position a still within the sequence, or to make a duplicate entry, for example when the image is needed in a headline sequence and as a main programme item.

Unwanted still images are removed from the rundown by selecting the image then pressing the **DELETE** key.

Changing the image position in the rundown order is effected by either selecting the item and using the mouse to drag the image to a new position, or by single clicking the still to select it, then pressing and holding the control key whilst using the up or down arrow keys to move the position of the image in the rundown.

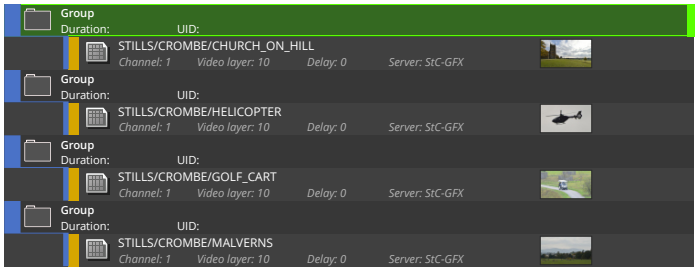
The position change techniques also work for a multi-element selection (click then shift-click or multiple ctrl-clicks) followed by drag or Ctrl Up/Down arrow.

To create a mix between two stills, select the entry for the still that will fade in, setting its transition mode to MIX, then set the duration of the mix and the tween type. Commonly tween modes are Linear (same as EaseNone), EaseInOutSine and EaseInOutQuad. The available tween transitions are illustrated in Appendix C. The same detail edit process is used for other transitions such as Push, Slide and Wipe.

During rundown playout the operator selects the still to load, then presses the F2 key to play it onto the output. The next item is manually selected, either by mouse click or by cursor down arrow.

Auto Advance, Single Channel Output

This mechanism enables an item to be played with the selection cursor automatically advancing to the following item in the rundown. This allows the playout operator to concentrate on issuing the play commands when the director calls for a stills change. Initially build the rundown as described in the preceding transmission output only section.

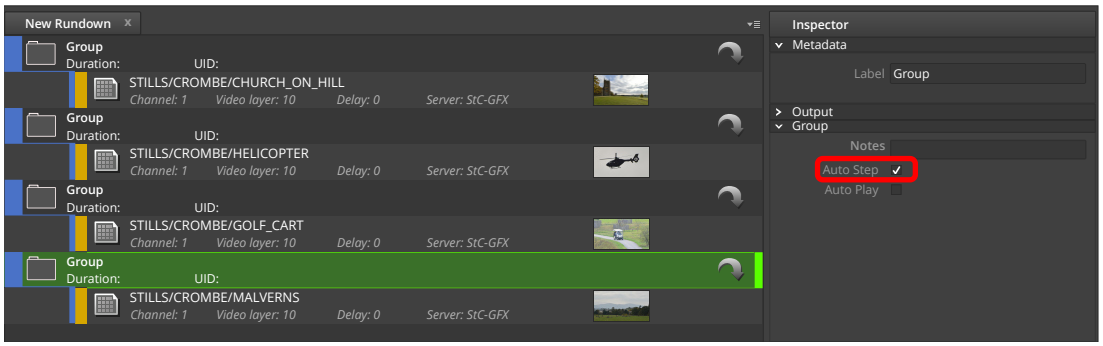


Rundown showing Images in Groups

Select the first still item in the rundown, then press the control and G keys. This puts the still into a Group. Select each still in turn and add it to its own group. After adding several groups, the rundown will look similar to the example shown to the left.

The indent (movement to the right) of each media item beneath the group header indicates the item is a member of that group.

The advantage of the group wrapper for stills replay operations is that the group can have an ***auto-step*** property enabled. The location of the group auto-step property in the inspector is indicated by the red rectangle highlight in the client screen shot below.



Groups with Auto-Step Enabled

Select each group in turn, setting the **Auto step** property. It is good practice to edit each group's Metadata Label property to identify the still name in the group. This metadata label identifies the still when the group display is shown collapsed. A double-click on the group bar collapses or re-expands the group display.

Playout Process

1. Select the first group in the rundown using the mouse.

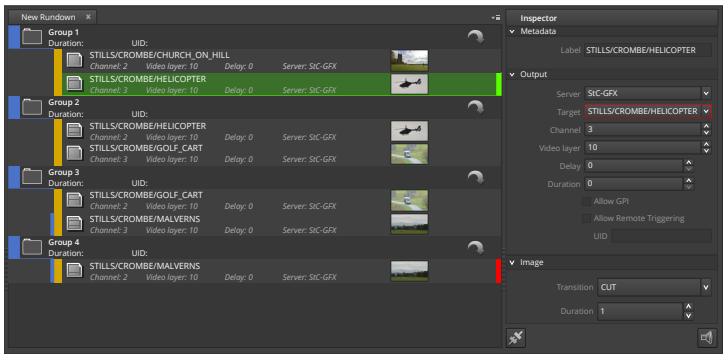
2. Press the F2 key to transition the still to the selected output channel. The Auto step mode selects the following group ready for the next play command.

Auto Advance with Tx and Pv Outputs

As previously noted, it is sometimes desirable to have a preview of the next still in a replay sequence. This requires two CasparCG server outputs. The transmit and preview process is supported using Group mode, each group entry having definitions for the content of the two channels.

Assume for this description that channel 2 is transmission output and channel 3 is the preview output. All groups in the still sequence apart from the last now have two images - the still to output to transmission and the still to output to preview.

The client display is similar to that shown in the illustration below.



Group Mode with Tx and Pv Channels

In this example, Group 1 contains two stills. The first still in the group (CHURCH_ON_HILL) is set to output on channel 2 (Tx channel). The Inspector shows us the second entry in the group (HELICOPTER) will output on channel 3 layer 10.

The second still for each group can be inserted by highlighting the first still of the group, thus setting the target for item inserts, then selecting the second still in the library and dragging or double clicking. Set the output channel using the inspector.

Alternately the second still can be copied from the following group by using the keyboard. There are four steps:

1. Select the still in the second group, then press ctrl D to duplicate the image reference. Leave the duplicated reference selected.

2. Use ctrl left arrow to take the still out of the second group

3. Use ctrl up arrow to move the reference above the second group

4. Use ctrl right arrow to join the reference to the group above the source of our copy.

5. Set the Output Channel to number 3 (still preview).

The above process is repeated for each group in the rundown. The final group needs only the transmission channel still.

Using Mix Transitions

It is good practice to delay the change of the preview output until any transmission output mix is complete, then cut the new preview still onto the preview output. The human visual cortex is adept at detecting changes. It can detect the cut of the preview output even when the observer is not focussed on that section of the vision stack.

This postponed preview output change mechanism is implemented by setting a value for the Delay property of the preview still. Set the preview delay time to the same value as the transition duration for the transmission still.

Playout Process

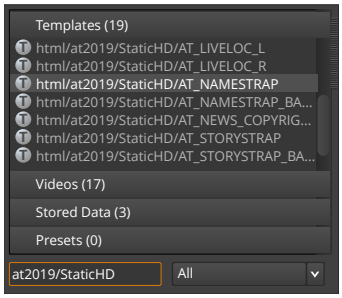
This is the same as used for the "no preview" output operations:

1. Select the first group in the rundown using the mouse.

2. Press the F2 key to transition the image to air. The Auto step mode selects the following group ready for the next play command.

Using Templated Graphics

Templated graphics can be used on any CasparCG server channel, either as overlays shown on top of other content, or output as fill and key for combining with other content in a vision mixer keyer. The following descriptions assume the templated graphics is output on server channel 1.



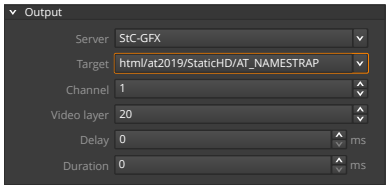
The templates available on the server or servers are listed in the Template tab in the left column of the standard client. If an expected template name is not shown in the library list remove any strings in the filter control and select all servers. Use the library refresh (*Ctrl R* on Widows) if the expected template is still not in the library list.

Best practice is storing templates in folders that are named to match the programme strand. This naming convention enables the folder name to be used as a filter string for library searches, as shown in the example at the left.

There is no preview picture for a template. Template creators *should* be tasked with providing a document, suitable for electronic display and print, showing the name of the template, an example of the template in action, and a list of placeholder names. It is also possible to program templates that show their placeholder names on the selected channel if a suitable named key and value are sent in a play command, or activated via an ***invoke*** function.

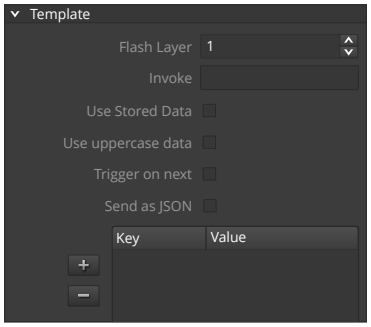
Select the wanted template in the library list. Either double-click the template name, or drag the template into the rundown sequence part of the client window. Either of these item add actions set the source server name and select server channel 1 layer 20 as the output destination.

Select the template instance in the rundown, activating the inspector for that item. In the Output tab, confirm the server name and the channel number are the intended settings.



Templated graphics default to using layer 20. This places templated graphics in front of video replays running in their default layer (layer 10). It may be necessary to change the default layers for some complex graphics applications. Leave the **Delay** and **Duration** fields at zero for captions that will be inserted and removed by the vision mixer operator.

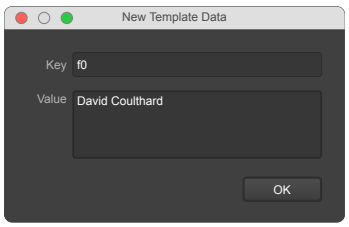
Setting Instance Data



Selecting a template in the rundown shows the Template properties panel in the lowest section of the inspector. An example template properties panel is shown at the left of this text.

There are several control fields followed by an entry box headed with the labels **Key** and **Value**. The instance data is entered in this area.

Click the + button to pop up the New Template Data entry box. The Key field is initialised with the next available ***f*** plus ***number*** (e.g. f0 as shown in the example New Template Data box). The key name can be edited as required by the template, but many templates are created to use keys named f0, f1 etc for speed of entry or editing.



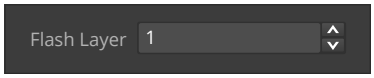
The entry box is closed by clicking on the **OK** button. A shortcut key can be used to close the entry form, or the tab key used to cycle focus between the Key, Value and OK elements.

When a key already exists in the template properties display, the Value is edited by double clicking the Key or Value entry in the list.

As each template is activated on an output channel using the ***Play*** command (pressing function key F2). The instance key names and values are formatted into a string sent to the assigned server as part of the

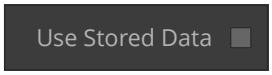
play command. The script code in the template extracts the keys and their values, inserting them into the placeholder fields of the graphic.

There are several control fields that may need editing.

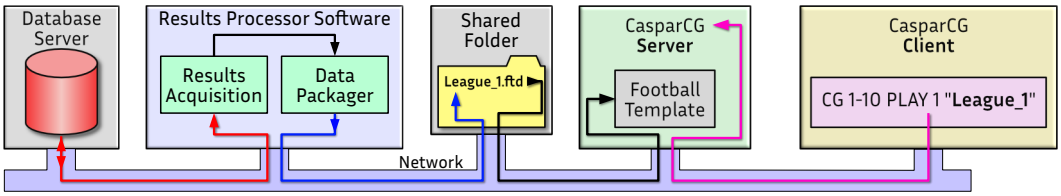


Flash Layer. This field is only relevant to Flash templates and allows the data to be directed at one of several layers within the Flash Action Script code. Leave value at 1 for HTML template use.

Invoke. This field can contain the name of a function in the template code that is called when the template ***Invoke*** key (function key F7) is pressed. Leave this field blank unless the invoke process is in use.



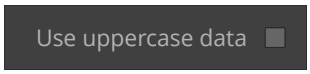
Use Stored Data. When selected, this tick box tells the server to fetch template instance data from the CasparCG data folder location defined in the server configuration file. This process is very useful where a large amount of data must be passed to a template, for example the scores for the entire set of clubs in a specific football league. The external data can be stored in the designated folder by an external process as illustrated below.



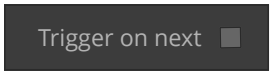
The SVT client shows the name of the external data file in the **f0** field. When the user sends the play or update command the client sends the file name of the stored data to the template. The CasparCG server code fetches the data from the named file, presenting the stored text string to the **update()** javascript function as if the key and value elements had been sent inline as part of the play or update command.

An external data collector programme can also store the data file in the CasparCG server data folder by connecting as a client to the server, moving the data to the server using the AMCP **DATA STORE** command.

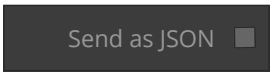
The library inspector lists stored data sets. If the data set name is dragged and dropped over a template in the rundown, the indicator to use stored data is set and the selected data set name are copied into the template instance in the f0 entry.



Use Uppercase Data. Instructs the client to convert all text fields into upper case before sending the command and data to the server.



Trigger on Next. When this flag is enabled the play command is not sent to the server until the Next key (function key F5) is pressed. This allows the graphics operator to prepare the next caption for display, then have an external system send a GPI or OSC trigger to the client firing the **next** operation.



Send as JSON. This flag changes the encoding of the data between the client and the server.

It is only relevant to a CasparCG HTML template. XML is the default encoding used in the data transfer. A single f0 data field with a value of **My Name** would be sent as:

```
<templateData><componentData id="f0"><data id="text" value="My Name"/></componentData></templateData>
```

Setting the JSON flag converts the data transmission into:

```
{"F0":"My Name"}
```

The scripting language for HTML, JavaScript, includes functions for JSON processing. XML support is a relatively complex process in JavaScript, so using JSON can simplify template programming.

In practice, both example transmission modes have to be wrapped in quotation marks to delineate the data segment of the overall command. This requires a backslash escape character used ahead of the embedded quotation marks making the JSON format transmission string to the server into:

```
"{\"F0\":\"My Name\"}"
```

Templated Graphics - Client Function Keys

There are seven function keys used for templated graphics replay control.

PLAY
Used to initiate playback of a template with instance data. May use animations programmed in the template to control the display of user provided data.

STOP
Used to initiate the removal of template from the display. May use animation to remove the data and other graphic elements.

NEXT
Timelines can contain programmed animation pauses. For example, a results graphic can be programmed to progressively reveal sections as the presenter describes the information. The animation timeline pauses at the end of the first phase. The user sends a ***next*** command to instruct the template to play on to the next timeline pause marker.

UPDATE
Used to change some or all instance data fields. Frequently used to update one or more text fields whilst leaving underlying graphic elements unchanged.

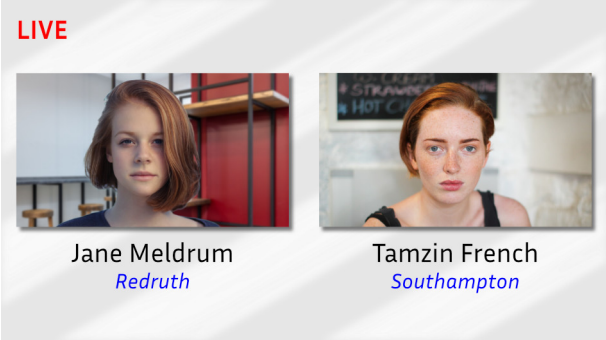
INVOKE
Used to call a specified function in the template. This allows random hops around a timeline or switching of context.

CLEAR LAYER
Clears the layer in which the template is currently active WITHOUT running any exit transitions.

CLEAR CHANNEL
Delete all layers in the selected channel. No output animations are run.

Play Live and Streaming Sources

There are many reasons why a “live” video source is required as part of a CasparCG channel output:



Individuals contributing to a programme via Zoom, FaceTime or Skype session, with their video composited onto a videowall or large studio display.

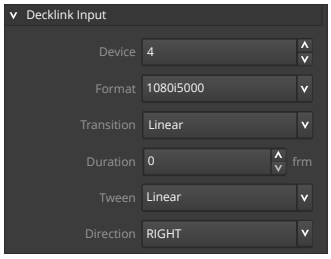
Showing a feed from an outside location whilst textual information displays alongside the picture.

Recording a mixer output or mixer bus output for later editing, or for use as a pre-recorded insert in a programme.

CasparCG server has multiple mechanisms to accept and display live content. Supported sources include SDI video via Decklink or Bluefish interface, live and pre-recorded streaming content, NewTek NDI delivered content, or webcams attached to the CasparCG server.

SDI

Server version 2.3.3_LTS supports SDI capture from Decklink and Bluefish444 interfaces. Server version 2.1 supports SDI capture via Decklink devices only. At least one device connector of the video interface must be available for input. For example, a Decklink Duo-2 interface may be configured with devices 1, 2 and 3 as outputs, leaving device 4 available for use as an input. Some Decklink interfaces support simultaneous input and output on the same Decklink device number.



The SVT client has a pre-built command to use a Decklink SDI input. This command is available in the Tools > Other library menu. The input device properties inspector for Decklink input is shown at the left of this text.

The Device field selects the Decklink device ID used for video input operation, device 4 in the example inspector at the left. The drop-down Format selector **MUST** be set to the scan format of the signal connected to the Decklink input. Failure to set this to the actual signal standard in use is a common cause of no visible input pictures.

The transition, duration, tween and direction boxes list the same options used on other source property forms, but the values set in these boxes are ***not*** sent to the server.

The output section of the Inspector selects the CasparCG server, target display channel and layer. When the command configured as above is activated (played), the AMCP command text to the server is of the form:

```
play 1-100 decklink device 4 format 1080i5000
```

SDI inputs can obviously be played using a custom command. The minimum command takes the form:

```
play 1-100 decklink device 4
```

Note that the above command does ***not*** include the video input format standard. When the command is run, the CasparCG server software interrogates the selected input device to check if it has an input and, if so, what scan standard is present on the input.

The Decklink input command supports filter parameters for audio and video, including the ability to flip the image and apply various blur filter algorithms. For example:

```
play 1-100 decklinnk device 4 vf “hflip, vflip”
```

There is no pre-defined SVT client control for Bluefish cards. Bluefish SDI input uses a custom command.

```
play 1-100 bluefish device 1 sdi-stream 1 uhd-mode false format 1080i5000
```

The SDI stream defaults to number 1 if not provided as a parameter, and the UHD mode defaults to false. When UHD-Mode is true, the card uses multiple input channels as the sub-elements that together deliver a UHD signal.

When the SDI source is required on multiple output channels, play the input on a virtual channel, using routing to copy the virtual channel to a layer on a one or more channels with physical output.

Network Stream Inputs

CasparCG can display content from a network stream. The source stream may be from a camera style device or from a streaming server. Stream inputs are processed and decoded using the ffmpeg producer. Any stream type known to ffmpeg can be used as an input source. The stream producer has four streaming source type headers it directly detects in the filename element of the AMCP play command. The four prefixes are **http://**, **https://**, **rtmp://** and **rtmps://**. **udp://** sources can also be used.

It is common to have multiple information, warning and error reports sent to both the console window and log file when receiving streams. Some errors and warnings only happen when the stream is opened, other streams have continual error, warning, and information reports. Hence the ability to see the server console window is very helpful when checking if a stream is available.

Test Streams

The author used some free-to-access streamed video, links below, to prove the play command syntax. These streams can be played using either a custom command, as shown in the AMCP commands below, or entering the http: url into the Target box of a video clip item. Be aware that live streamed camera feeds do sometimes go offline for various reasons.

Sample Stream 1:
This is from a web camera outside the Abbey Road recording studios in London. The camera looks at the pedestrian crossing seen on the cover of the Beatles 1969 Album “Abbey Road”. The AMCP command line for this stream is:

```
play 1-10 https://videos3.earthcam.com/fecnetwork/AbbeyRoadHD1.flv/playlist.m3u8
```

Sample Stream 2:
This is from a camera looking at Penzance Promenade (UK South Coast).

```
play 1-10 https://camsecure.co/HLS/queenshotel.m3u8
```

Sample Stream 3:
Camera looking at the Strait of Gibraltar.

```
play 1-10 https://videos-3.earthcam.com/fecnetwork/ceuta1.flv/chunklist_w1052202424.m3u8
```

Sample Stream 4:
This stream is a webm encoded movie about the Northern Lights. The AMCP command to show this video is:

```
play 1-10 https://asc-csa.gc.ca/videos/recherche/1_lia7qnwo/1_rlmai0o4.webm
```

This streamed movie can use the load/load and freeze command (F3 key), the play command, and the pause/resume (F4 key).

The above movie is from a Canadian Government web site. Most content relates to the Canadian Space Agency. There are many space themed movies available on that site. The home page for the movies is https://asc-csa.gc.ca/eng/search/video/default.asp. Finding other movies from this web site for use in CasparCG server requires access to a standard computer browser. Navigate to the home address above. Select a movie from the lists, opening the movie page in the browser. Right-click the movie image, and select “Open video in new tab” from the context menu. Switch to the new tab if the browser does not auto select the new content. Copy the url from the browser tab pasting it into the target field of the CasparCG video item.

Using streams where the user configures both the streaming encoder and the CasparCG stream receive command can be complex to debug.

The example streams with a url ending in **.m3u8** generate lots of info level log messages, resulting in huge daily server logs when the stream is played for long periods. Hence when using such streams it can be good practice to change the type of logging messages recorded by using the AMCP log level command. For example:

```
log level warning
```

Captures reports in warning and error categories. To see the messages when testing reception of a previously unproven stream the log level needs to be set at info:

```
log level info
```

How can a user see the live output of log messages when the CasparCG server is in a remote apparatus room? One possible method is via VNC (Virtual Network Computing), using a VNC server installed on the CasparCG and a VNC client on a machine whose display is close to the users operational work position. VNC supports read-only displays, maintaining the integrity of CasparCG server operation.

Temporal jumps may be seen on many live and pre-recorded streaming services when displayed on a CasparCG server channel that is configured for one of the 25Hz related scan speeds. The issues arise because webcam devices tend to operate at either 30Hz or 60Hz, and some lower cost units use even lower frame scan rates (e.g 15Hz). CasparCG at best can just drop or duplicate frames.

NDI Sources

NewTek NDI is a low-latency transport system that moves audio-visual content between terminals on a computer network. NDI uses data-rate reduction to limit network bandwidth whilst retaining good quality. Sources announce their availability on the network using **mDNS** (multicast Domain Name System). Apple commonly refer to mDNS as **Bonjour** technology. NDI sources do not output audio-visual content until a receiver requests that content, but the mDNS broadcasts enable receivers to detect available sources and their names. When a receiver requests access to the content, the source and destination negotiate the detail of the connection to be used.

CasparCG server version 2.3.3_LTS supports direct reception of NDI sources. The negotiation between the source and the receiver in CasparCG commonly exhibits a significant delay between request for use and the signal displaying in the CasparCG channel.

There are two methods available to hide this delay, both require the source to be selected and played in CasparCG before it is displayed on the target output.

1. Use direct reception of the source on the channel and layer that will display it. The target layer is set to have video opacity at 0 and audio muted using AMCP mixer commands. The NDI source is then played. At a later time the NDI source is shown by sending mixer commands that set the video opacity at 1 and the audio volume to 1.

2. Use a virtual channel as the NDI receiver. Play the NDI source in this channel. Use layer route commands to switch the received NDI source to display on a physical channel when needed. An advantage of the routing process is the source reception can be tested using a preview output channel.

An NDI source is activated by a play command of the format below.

```
play 1-20 "ndi://source_computer_name/source_name"
```

The command can be sent from the SVT client by either a video item with **ndi://source_computer_name/source_name** in the target field, or sent as part of a custom command.

NOTE - the first element of the source name (**ndi://**) is **case sensitive**. Using **NDI://** will cause the play to fail. The computer name and source name are case insensitive.

The server console often shows a message stating the source is not available, followed a few seconds later by `202 PLAY OK` after the source and destination NDI negotiation is complete.

NDI installations default to unicast network operation. If the same source is selected on ***N*** devices there will be ***N*** unicast streams in use. An HD signal without alpha/key uses around 100 Mbit/s per device connection. It is possible to use multicast transmission operations for NDI to minimise network traffic as all receivers use the same source stream. See the online NewTek NDI documentation for a description of considerations needed when enabling multicast on a network. NewTek publish an overview of networking practices for NDI. [link].

How can a user find the available NDI sources and their names? Multiple methods are available. There is an AMCP command that lists the sources the CasparCG server has detected. The command format is:

```
ndi list
```

The server response has a format of:

```
#200 NDI LIST OK
1 "Graphics_Edit.01 (macOS AV Output)" 192.168.42.201:5961
2 "MCR.03 (NDI Signal Generator)" 192.168.42.190:5961
3 "MCR.NDIConnect.01 (SDI Src 1)" 192.168.42.191:5961
4 "MCR.NDIConnect.02 (SDI Src 2)" 192.168.42.191:5962
5 "MCR.NDIConnect.02 (SDI Src 1)" 192.168.42.192:5961
6 "CCG.Server.StA.01 (TestSignal Generate)" 192.168.42.101:5961
```

Each line of the replay starts with a simple ident number. The section in quotes comprises the computer name followed by the source name in brackets. The final element of the line is the sources IP address port number. NDI defaults to starting to allocate output ports at 5961.

If the user wanted to display the output from source 1 in the preceding example list, the command is of the form:

```
play 1-10 “ndi://Graphics_Edit.01/macOS AV Output"
```

The ndi list command can be entered on the server console window, via a telnet session to the server, or by using a custom program that connects to the server, parses the response and displays the list to the user.

A second method to access the list of source names uses the free ***NewTek NDI Tools***, available on both Windows and macOS operating systems. One of the installed NDI tools is called **Studio Monitor.** This tool allows all NDI sources on the network to be listed, and a selected source viewed and optionally recorded locally. The current version NDI tools are downloadable from https://ndi.tv/tools/.

CasparCG server version 2.3 was built against the NDI version 4 libraries. The tools are, at the time of writing, using version 5 of the NDI specification. Loading the tools onto the server loads the latest NDI dll support library, enabling CasparCG to list version 4 and version 5 sources.

Skype can both use an NDI source in place of a video camera for outgoing content, and output an NDI signal created from the remote end contribution. This NDI output of the contributor has to be enabled each time a call is started. The activation feature is accessed via the **Calling** set-up page. There is an **Advanced** line on the calling home page. Use the right pointing arrow on the advanced line to open the advanced settings. Locate the **Allow NDI Usage** switch and enable the NDI output. Select the position of the NDI watermark added to the output.

NDI connections are also available for Zoom calls, using features in a hardware-supported Zoom Room. Multiple NDI streams are available via the zoom room hardware installation. Detailed requirements for using NDI in a Skype call are available at:

https://support.zoom.us/hc/en-us/articles/360061903532-Using-Network-Device-Interface-NDI-

Is there NDI Input on Server version 2.1?

The simple answer is "No". There is an available NDI receive mechanism if the user has a spare Windows PC plus a Blackmagic Design video card with SDI output.

NewTek offer an application called **NDI Connect**, available in 'free' and 'Pro' versions. The details of the capability differences, the required operating system and hardware requirements are available at https://www.newtek.com/ndi/applications/connect/.

Downloading the free version requires a simple registration process. Part of the registration process provides a contact email address. After registration is complete, the link to the free software download is sent to the registered email address.

The free version of NDI Connect allows up to two channels of conversion, including converting an NDI source into an SDI signal. That SDI signal can be routed to a CasparCG SDI input enabling reception of the NDI source.

Webcam Interfaces

Windows has a standard interface that enables content from webcam and HDMI to USB convert interfaces to be recognised and used by the host PC without requiring any driver software. The video element of the interface uses JPEG compression of the media from the camera or HDMI source. A similar webcam interface is available on Linux computers.

Webcam devices can be very useful close-up cameras for use by a presenter examining small objects. Using an HDMI to USB capture adapter enables various digital cameras, including DSLR types, to be used for production functions. Low cost extender systems using Cat6 cabling or dedicated fibre are available for both HDMI and USB connections, allowing the source and host computer to be more than 80 metres apart.

CasparCG displays signals from webcam interfacing devices by using the *Direct Show API*. This interface type is indicated by a content target name that starts **dshow://**. A USB webcam that has a direct show name **USB Live camera** is shown in a CasparCG channel using a custom command of the form:

```
play 1-10 "dshow://video=USB Live camera"
```

or entering a target address of **"dshow://video=USB Live camera"** into the target field of a video item in the SVT client rundown. Note that the device identifier string is case sensitive, thus **USB Live camera** and **USB Live Camera** are different devices.

Using a custom command to play webcam content allows filters to be added to the command. An example video filter is one that produces a left-right swap of the image. The video filter element is **vf hflip** giving a custom command of:

```
play 1-10 "dshow://video=USB Live camera" vf hflip
```

Most webcam devices include a microphone. The video and audio can both be activated as part of the same command. Assume a video element name of **USB Live camera** and a microphone name of **Microphone (USB Live camera audio)**. The camera output is displayed using a command of the form:

```
play 1-10 "dshow://video=USB Live camera:audio=Microphone (USB Live camera audio)"
```

How can a user find the names of the available devices? There are two methods to discover the names of devices on a Windows host computer, both methods need user access to the server keyboard and screen.

1) Use PowerShell to list devices

Right click above the Windows icon at the edge of the taskbar. Select Windows PowerShell from the context menu. Enter the command:

```
get-PnpDevice -PresentOnly | Where-Object {$_.InstanceID -match '^USB' }
```

The terminal window shows a list of connected devices that contain USB in their name. If the name is not obvious, unplug the USB connection then run the command. Insert the USB device and run the command a second time, looking for the extra entries in the list.

2) Use ffmpeg to list dshow devices

The ffmpeg executable task that is part of the CasparCG server distribution can be used to list the devices. Type `CMD` in the Windows Taskbar search box, selecting **Command Prompt** in the displayed applications list.

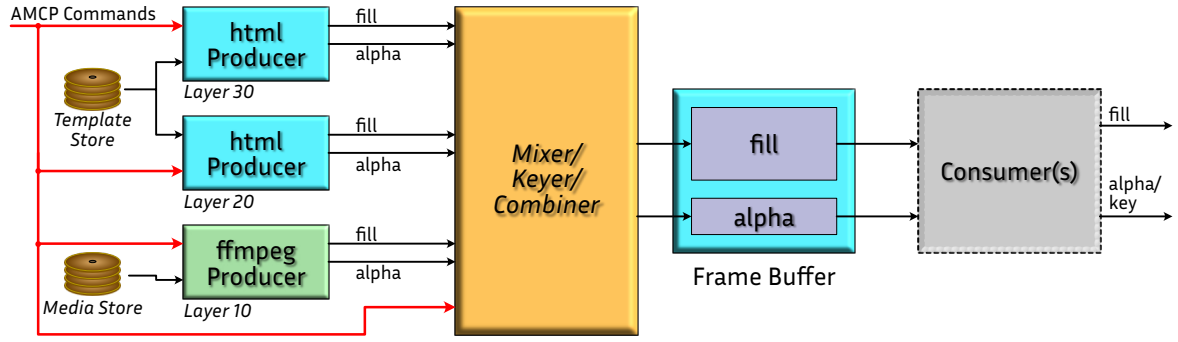
Use the cd (Change Directory) command to set the active folder to the one with the CasparCG server. Use the `dir` command to prove there is a file called **ffmpeg.exe**. Enter the command

```
ffmpeg -list_devices true -f dshow -i dummy
```

After a short pause the output on the command terminal shows properties of the ffmpeg build, followed by a listing of the direct show video and audio source names.

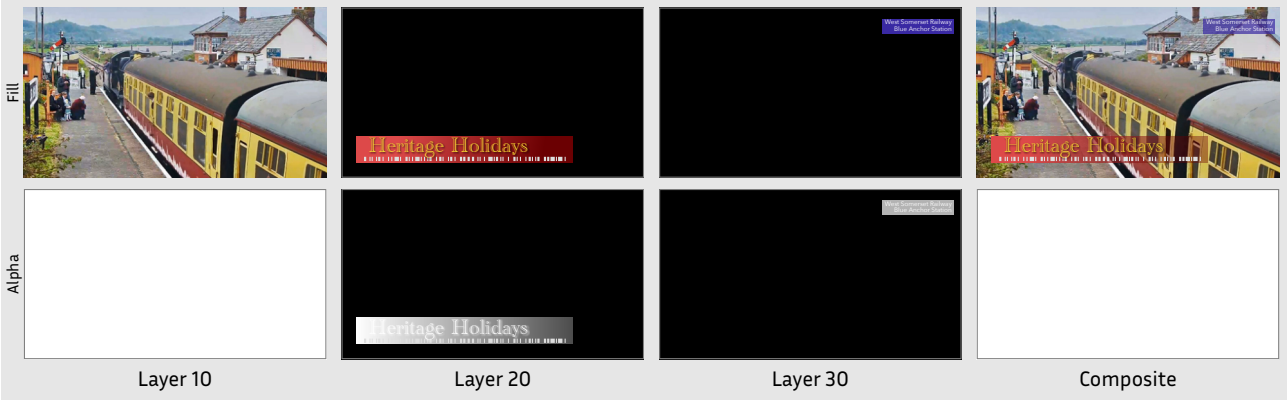
Channel Mixer

Every CasparCG channel uses a mixing and compositing process to merge content from the active layers of the channel. The following diagram illustrates the signal flow for a channel with three active layers.



The compositing process requires fill and alpha content from each active layer. Producers whose content does not include an associated alpha channel, for example a simple video clip, are auto-associated with a full-frame alpha image.

An example set of content for the above signal flow illustration is shown below. The upper row shows the fill content of each source channel plus the composited result. The associated alpha channels are shown on the lower row.



The video used in layer 10 has no associated alpha-channel video, so it is allocated a full frame white alpha channel by the ffmpeg producer.

Layer 20 has a lower-third with a semi-transparent red box over which there is yellow text and some white boxes. The alpha channel shows the white to grey gradation for the background box with full amplitude white where the text and bar blocks are present.

Layer 30 has a semi-transparent blue box with white text, the even grey box in the alpha channel shows the same transparency at all points where the colour is present.

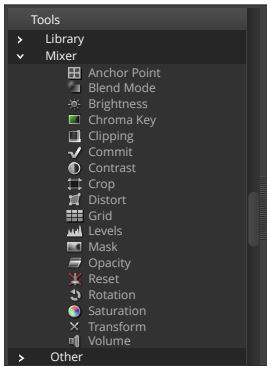
The compositing process uses the alpha channel, the layer order, and the blend mode of each layer to define the output fill and alpha.

The mixer-combiner supports layer-by-layer manipulation of a wide range of properties. The majority of these adjustments have tweened transitions with control curve and duration settings. The underlying AMCP command processing tolerates a very wide range of control values, but the values that effect modification are often limited internally to a much smaller range. The SVT client typically offers a small control range where maximum visible effects happen.

Adjustments can be grouped to achieve an overall production requirement. For example a coloured background in layer 5, a video in layer 10 set to opacity 1.0. When the video completes the final frame

saturation is set to zero and the opacity to 0.4 providing a combined backing for the display of text or other graphic overlays.

Mixer Transforms

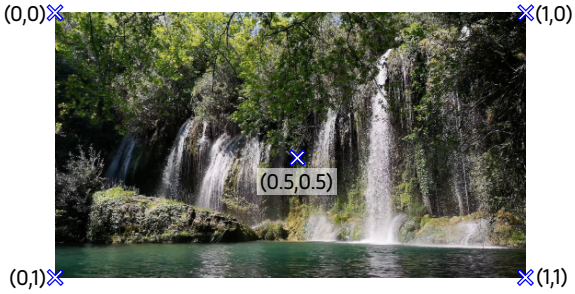


The SVT client includes a set of tools that enable a user to control most available signal adjustments using the CasparCG server mixer process.

Many of these adjustments will be familiar to users of pixel-based image manipulation tools that adjust properties such as brightness, contrast and saturation of the picture content in a layer.

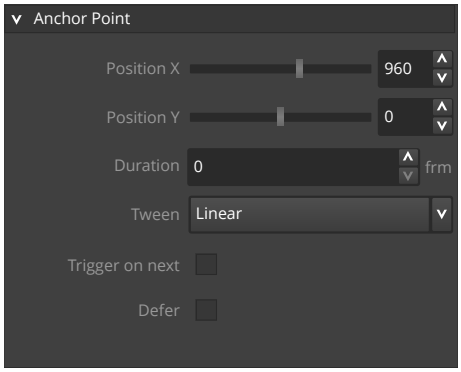
Because CasparCG replays content with audio, audio volume controls are available.

Several mixer tools allow the size and position of the image content to be moved relative to the display viewport. CasparCG server supports many signal standards for both size and frame rate, making pixel based control syntax inadvisable.



AMCP mixer commands use a resolution independent method to specify positions and scaling. The top left corner of the active screen area is at (0,0) with the bottom right corner at (1,1).

The resolution independent position and size values are sent as floating point numbers, hence the centre of the image is at (0.5,0.5).



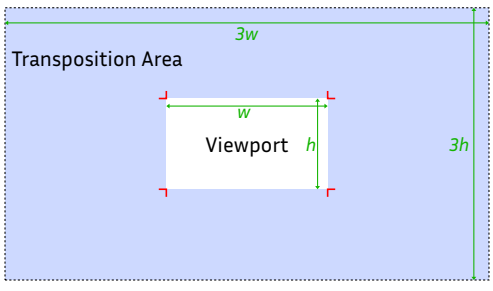
The SVT client knows the selected channel pixel resolution from status data emitted by the server. The resolution data enables the SVT client to operate using pixel-based values in the control interface.

The mixer anchor point inspector is illustrated by the adjacent screen grab. The anchor point position values can be adjusted using either a slider or numeric entry boxes at the side of the slider. When the command is run, the client converts the pixel location value to a floating point resolution-independent value sent to the server channel.

The following paragraphs illustrate the actions of the mixer tools.

Repositioning and Resizing

These manipulations use the **transform** tool. The tool adjusts the picture x position, y position, width and height.



In the illustration at the left, the viewport is the standard display window for the channel, for example 1920 by 1080 pels for a full-HD mode. The transposition area, the light-blue box, is 3 times the width and 3 times the height of the viewport.

Controls in the SVT client transform tool enable the unscaled source media to be placed anywhere in the transposition area, enabling the entire source image to be placed outside the viewport. Using a second transform command with a finite duration and tween type enables the source picture to slide into or out of the viewport.

The SVT client transform tool enables entry of sizes between 0 and twice the width and height of the source. Transitioning between two set sizes, one at zero and one at the normal width or height displays as a picture squeeze transition.

The AMCP command that implements the repositioning and transform is the **mixer fill** command. The syntax of the command is of the form:

```
mixer 1-10 fill <x> <y> <x-scale> <y-scale> <duration> <tween>
```

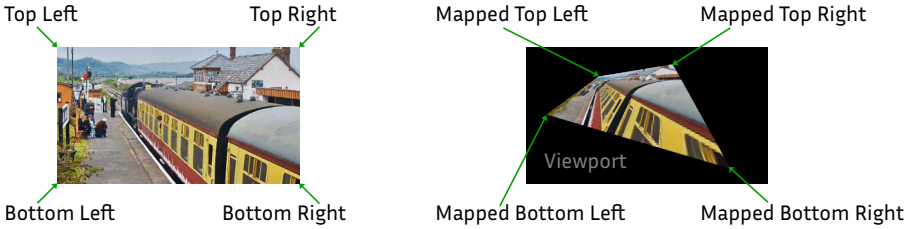
The x and y values set the position of the top left corner, x-scale and y-scale set the width and height. The transition duration is defined as a frame count. The tween control defaults to “linear” if a tween string is not provided in the AMCP command. Setting the picture to centred, half-width and half-height transitioning in 100 frames uses a command of the form:

```
mixer 1-10 fill 0.25 0.25 0.5 0.5 100 linear
```

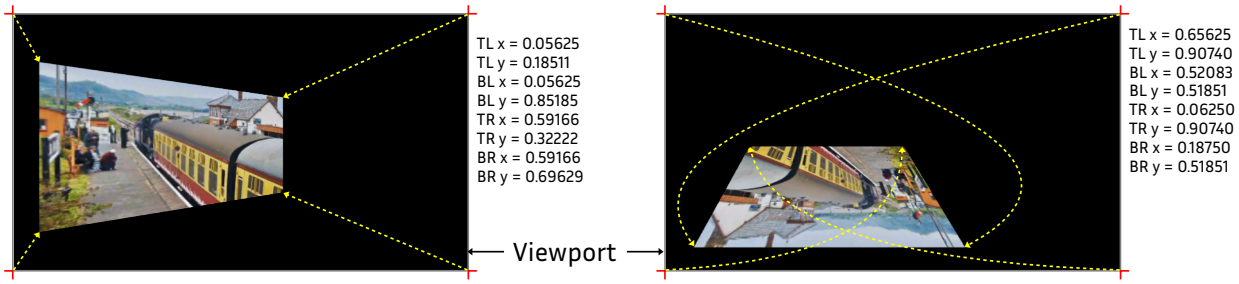
Using a custom command to set the fill command parameters enables larger property values to be sent than those supported in the SVT client widget. For example the width or height can be set to give a zoom in of more than twice size.

Shape Distort

This effect uses the **distort** tool, available in the SVT client mixer tools. The tool uses the AMCP perspective transform that maps the four corners of the source picture to user-defined positions as illustrated below. The source picture is shown at the left, the manipulated picture at the right.



Careful control of width and height produces outputs that look as if a corner has been moved in depth space (z-space) as shown in the following illustration. The numbers shown at the right-hand side of the viewport areas are the resolution-independent x and y values sent to the server to realise the associated picture display. Label TL is top left, BL is bottom left, TR is top right and BR is the bottom right.



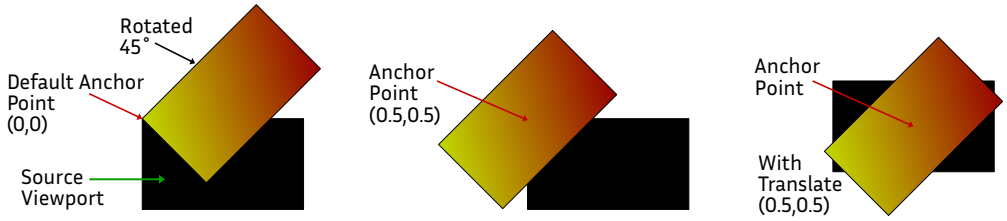
The dotted yellow lines show how the content from each corner is mapped to implement the effect. The leftmost example supports a timed transform from a full frame picture to the position shown whilst the picture corners track the dotted lines. When the right side example uses a timed transition from full frame it shows many intermediate shapes. The syntax of the perspective command used for the client shape distort has the form:

```
mixer 1-10 perspective <TLx> <TLy> <TRx> <TRy> <BRx> <BRy> <BLx> <BLy> <duration> <tween>
```

Where the first eight parameters are floating point values, the duration is a frame count, and the tween is a string, defaulting to linear if no tween is specified. The order of parameters in the command is clockwise from the top left corner.

Rotation

Rotation operates around a defined control point. In CasparCG the centre of rotation position is set by the **anchor point** transform. When mixer effects are reset, the anchor point is set at the top left corner of the picture.



The effect of changing the anchor point is illustrated above. The numbers are position values in resolution independent notation. The examples all use rotation by 45°. The leftmost example shows rotation around the default anchor point. The central example shows the result of changing the anchor point to (0.5, 0.5). The image moves left by 0.5 and up by 0.5, then the rotation is applied. The rightmost example shows the effect of adding a translate of (0.5, 0.5) to the anchor point move. Subsequent rotate commands use the centre of the source picture as the rotation centre. The AMCP rotation command has the form:

```
mixer 1-10 rotation <angle> <duration> <tween>
```

The rotation angle is specified in degrees, and is a floating point value. The mixer anchor point command has the form:

```
mixer 1-10 anchor <x> <y> <duration> <tween>
```

Opacity

The opacity tool adjusts the density of the content in a layer. The SVT client opacity tool supports entry of values as a percentage between 0 and 100. The percentage value, expressed as a floating point number, is sent to the CasparCG server. The AMCP **opacity** command has the form:

```
mixer 1-10 opacity <opac> <duration> <tween>
```

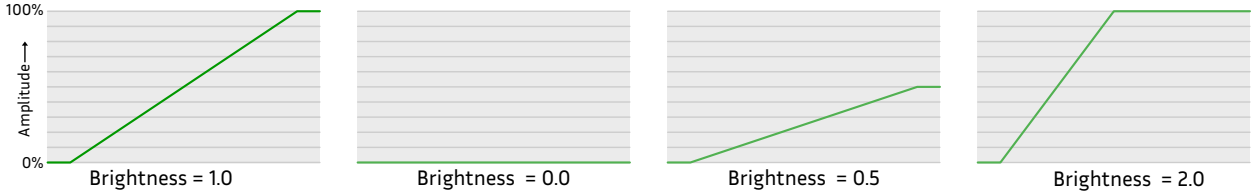
If a custom command sends a **<opac>** control parameter value less than 0, the server forces the used value to 0. A control parameter greater than 1.0 is not clipped, and creates somewhat unpredictable psychedelic mixtures of the target layer and the layer below.

Brightness

The brightness control adjusts the ***gain*** or ***exposure*** of the source picture. Using 0.0 to 1.0 as the source signal range for each component and the brightness gain control *g*, the output amplitude before clipping is applied is computed by the expression:

$$output\_sample\_amplitude = input\_sample\_amplitude * g$$

The SVT client brightness control range is 0 to 200 percent equivalent to a gain, *g*, between 0.0 and 2.0. A *g* value of 0 makes the picture black, but not transparent. A control value of 100 (gain 1.0) does not modify the picture. Where gain values greater than 1.0 are applied, any red, green and blue component of the picture that exceeds the normal digital video range are clipped. This may produce hue distortion or , if all channels clip, coloured zones to become white. The effect of the brightness operation as seen on a waveform monitor is illustrated below. For simplicity of display, a linear left to right ramp source picture is used.



The AMCP command has a single parameter. The command format is of the form:

```
mixer 1-10 brightness <b> duration tween
```

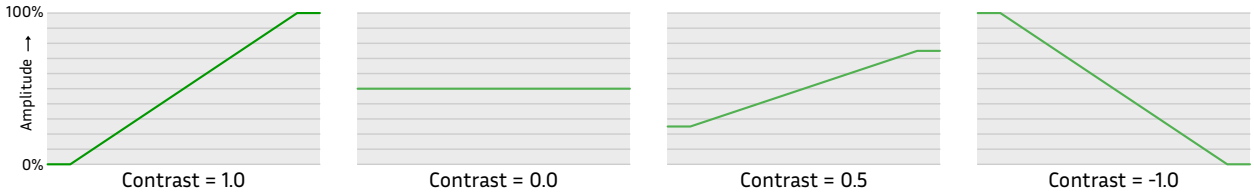
Negative values for the brightness property, <b>, are clipped to 0, but positive value are not clipped. Video levels that exceed the supported maximum value are clipped, as shown by the Brightness = 2.0 waveform in the above illustration.

Contrast

The contrast control also adjusts the gain of the source signal. The significant difference between the brightness and contrast transforms is the zero gain level used. Using 0.0 to 1.0 as the source signal range for each component and the brightness control *b*, the output amplitude before clipping is applied is computed by the expression:

$$output\_sample = ((input\_sample - 0.5) * g) + 0.5$$

Hence a control gain of 0.0 gives a mid grey output. The SVT client contrast control range is 0 to 200 percent equivalent to a gain, g, between 0.0 and 2.0. Negative contrast gain values cause a signal level invert - black inputs becomes white outputs, red inputs become magenta outputs etc. The waveform display of a linear luminance left to tight ramp for various contrast gain settings is illustrated below.



The AMCP command has a single parameter. The command is of the form:

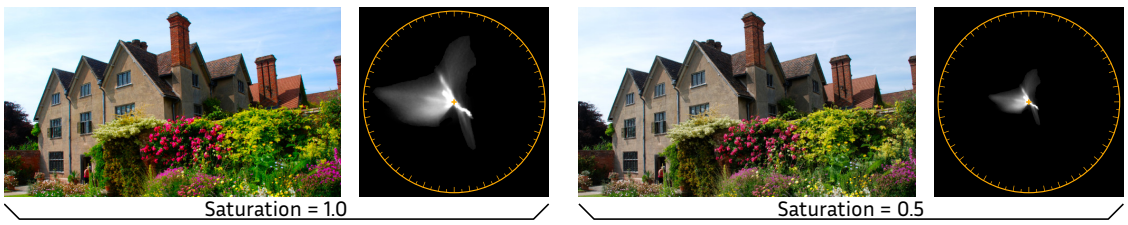
```
mixer 1-10 contrast <c> duration tween
```

The control value, **<c>**, is not clipped.

Saturation

The saturation control adjusts the colour intensity in the picture. The colour can be totally removed when saturation is 0.0 and strongly enhanced when saturation is greater than 1.0. The SVT client controls support saturation between 0.0 and 2.0.

Saturation change is visible on both the picture and on vectorscope style displays. The example pictures below show the picture at standard saturation (1.0) and at half saturation (0.5). The vector display for each example is at the right of the adjusted picture.



Desaturation, sometimes coupled with reduced brightness, is a useful way to leave a background image that shows the connection between overlayed graphic data and a location or event, whilst leaving the primary viewer attention on the overlaid graphic data.

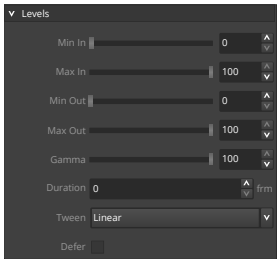
The AMCP saturation command uses one primary control property, with secondary elements that control transition durations and tweening curves. The command takes the form:

```
mixer 1-10 saturation <s> <duration> <tween>
```

No clipping is applied to the **<s>** parameter. Negative values produce colour inversion, and large positive values often cause peak video clipping with loss of detail.

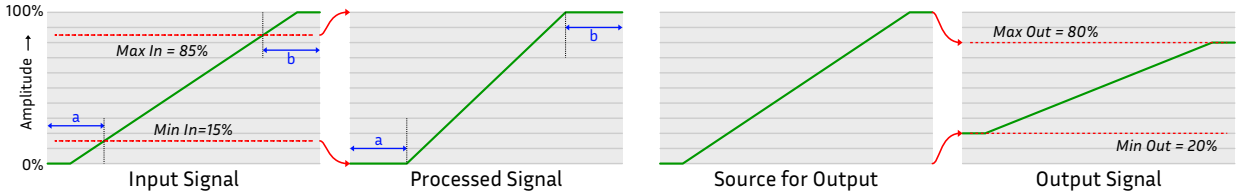
Levels

The levels control adjusts brightness, contrast and tonal ranges of a picture. Pictures that are exposed with low contrast giving shadow areas that are too light, or highlight areas too dark, can be normalised. Similarly a contrasty pictures can have shadows lightened and highlights darkened. The gamma adjustment within levels allows some stretching or crushing of mid-tones.

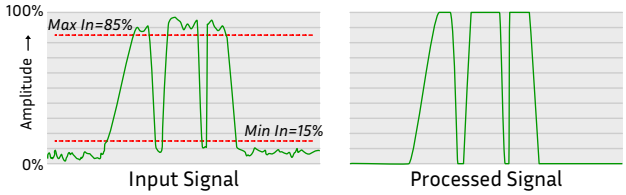


The SVT client controls for levels adjustments are illustrated at the left.

The top pair of controls, Min In and Max In, control the tonal range of the source that is passed to the rest of the levels process. When Min In is set as 15% and Max In is set at 85%, input samples below 15% of peak-value are set to zero; input samples greater than 85% are set to peak-value. Source samples between Min In and Max In are linearly scaled to fit the zero to peak-value range. The effect on a linear ramp signal is illustrated by the left-side pair of graphs below.

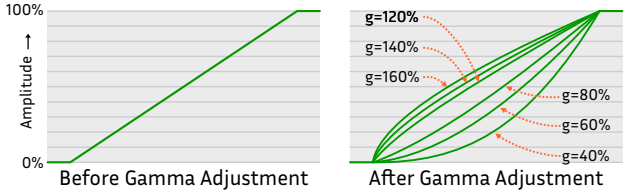


The second pair of controls, Min Out and Max Out, select the range of values output by the levels processing. The samples from the input processing and gamma modification stages are gain adjusted and offset to make the full range input fit the amplitude limits set by the Min Out and Max Out controls. The Min Out - Max Out processing is illustrated by the right-hand pair of graphs above. In those graphs Min Out is set to 20%, and Max Out is set to 80%. The net result makes dark areas of the picture lighter, with reduced contrast.



One application of the max and min in control is to remove small amounts of shading or noise near minimum and maximum signal. The input signal shown at the left is typical of the output from a TV camera looking at some text in a magazine or newspaper. It would not provide a clean key into graphics compositing operations. Applying the min and max level limits tidies the signal waveform.

The transfer characteristic of Cathode Ray Tubes (CRT) used in analogue TV displays is non-linear. The transfer characteristic is a power law of the form light_out = k*V^n where V is the input signal amplitude. The value for n is typically around 2.2, and is called the display gamma. Source content, such as a camera output, has the inverse transfer law applied so that light input to the camera and light output from the display follow a (very nearly) linear relationship.



The Levels Gamma control adjusts the amount of gamma pre-correction applied to the source picture. The graphs to the left show the waveform for a source linear ramp after processing at various gamma settings. In these graphs **g** is the shorthand for gamma. Gamma values less than 100% tend to darken the darker areas of the source picture, whereas gamma values greater than 100% lighten the darker and shadow areas of the picture. Adjusting the gamma parameter using the SVT client only supports gamma values from 100% down to 0. Setting gamma to higher values requires use of a custom command.

The AMCP mixer levels` command is of the form:

```
mixer 1-10 levels <minIn> <maxIn> <gamma> <minOut> <maxOut> <duration> <tween>
```

Where the first 5 parameters (minIn to maxOut) are floating point values. In general, clipping is not applied to the parameter values provided in a custom command.

Chroma Key

The chroma key process replaces user-selected colour areas in a source picture with picture content from a second source. The process can use any target colour, but saturated blue and green are often used as the key source colour because both have good separation from human skin tones.

In CasparCG mixer chroma keying two ***adjacent numbered*** layers are used, for example layers 9 and 10. The picture containing the keying colour zones is played in the upper of the two layers. The replacement content is in the lower layer.

The CasparCG chroma key has two operating modes - legacy and full control. Legacy mode is deprecated, but is the mode supported by the chroma key control widget in the SVT client. Full control mode is available using a custom client or a custom command.

The AMCP command processor tests the first parameter for the presence of one three strings:

A) “none”

B) “green”

C) “blue”.

When one of these strings is present, the keyer operates in legacy mode. Up to 5 further parameters, summarised the the table below, can be present in the legacy command.

Parameter Number	Parameter Name	Optional?	Value Type	Default Value	Description
1	hue_name	No	string		The keying colour - blue, green, none (no keying)
2	threshold	No	Floating point		Range 0.0 to 1.0 controlling several keyer properties via equations in the server software.
3	softness	No	Floating point		Controls the separation of the keyed and non-keyed zones at the edge of the key colour.
4	spill	No	Floating point		Light reflected off a high-saturation background may reflect in areas of the picture causing some keying to occur. The spill control adjusts the keyer sensitivity in such areas.
5	duration	Yes	Integer	0	Transition duration time in frames
6	tween	Yes	string	linear	Transition profile

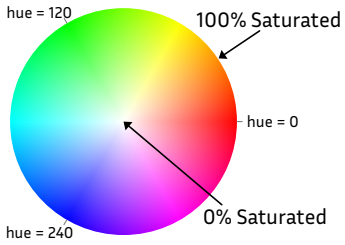
An example legacy command line, including optional parameters is:

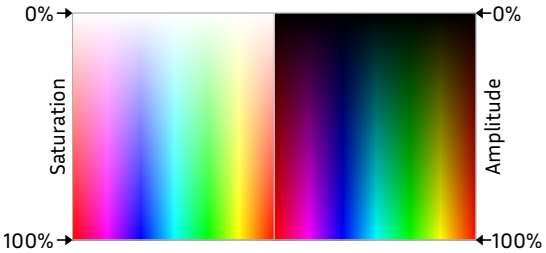
```
mixer 1-10 chroma green 0.5 0.35 0.2 100 linear
```

Full control properties are shown in the table below. Setting the values for a specific source picture may require iterative entry because one property setting may have implications for another property value. For example, changes to the target_hue angle may require adjustments to the hue_width.

Parameter Number	Parameter Name	Optional?	Value Type	Default Value	Description
1	enable	No	string		0 disables chroma keyer on selected layer. Do ***not*** send the other parameters when disabling the keyer
2	target_hue	No	Floating point		Sets the target hue expressed as an angle between 0 an 360. This defines the centre of the hue selection window for the key. Chromakey green has a hue angle around 120.
3	hue_width	No	Floating point		Sets the range of adjacent colours accepted by the hue selection window. Enter a value in the range 0.0 - 1.0, where 1.0 represents 360 degrees.
4	min_saturation	No	Floating point		Sets the minimum saturation required for a colour sample to be inside the chroma selection window. Range 0.0-1.0
5	min_brightness	No	Floating point		Sets the minimum brightness required for a colour to be inside the chroma selection window. Range 0.0 - 1.0
6	softness	No	Floating point		Sets the softness of the chroma selection window edges.
7	spill_suppress	No	Floating point		Controls the spill suppression within 0-180 degree range. It works by taking all hue values within ± this value from target_hue and clamps it to either (target_hue - this value) or (target_hue + this value) depending on which side it is closest to.
8	spill_suppress_sat	No	Floating point		Controls how much saturation should be retained on colours affected by spill_suppress. Range 0.0-1.0. Full saturation may not always be desirable on suppressed colours.
9	show_mask	No	Boolean (0 or 1)		When enabled (value = 1) the monochrome mask is shown in place of the keyer output. This display mode can assist an operator when adjusting the chroma key property settings.
10	duration	Yes		0	Transition duration time in frames.
11	tween_type	Yes		linear	Transition profile.

The effect of adjustments to each of the parameters is easiest to visualise by using a source picture with elements that contains a wide colour gamut which also has variations in saturation. Two such colour element test blocks are illustrated below.





The centre of the circle is 0% saturated (white). Saturation increases with radius, reaching 100% at the outer edge. The commonly green and blue chroma key colours have hue values 120 and 240. The rectangular colour brick has the same colour gamut as the wheel, but the colour property variations are between picture top and bottom.

target_hue

The target hue is the primary selector of key colour. The dark grey areas in the adjacent pictures are the zones where keying has occurred. Edge softness is minimised in these examples.

The three circles to the right show the effect of changing just the hue value in the chroma key control command. The AMCP command for the leftmost colour circle, with the hue element identified in red, is:

```
mixer 1-10 chroma 1 120 0.2 0.4 0.8 0.0 6 0.2 0 0 linear
```

hue_width

The hue width control sets the range of colours included in the keying. The width is symmetrically placed about the target hue angle. The edge is normally softened, but is minimised in these examples.


Hue Width 0.1


Hue Width 0.3


Hue Width 0.5

The hue_width value is in the range 0 to 1, and the value multiplied by 360 gives the range of acceptance angle. The hue width control is the third parameter of the AMCP command:

```
mixer 1-10 chroma 1 120 0.3 0.4 0.8 0.0 6 0.2 0 0 linear
```

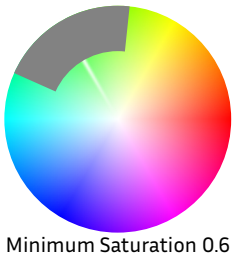
min_saturation

The minimum saturation control sets the strength of colour required to become part of the key. Edge softening is minimised in these examples.

The min_saturation value is in the range 0 to 1. It is the fourth parameter in the AMCP command.


Minimum Saturation 0.2

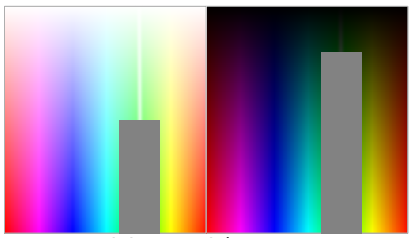

Minimum Saturation 0.4

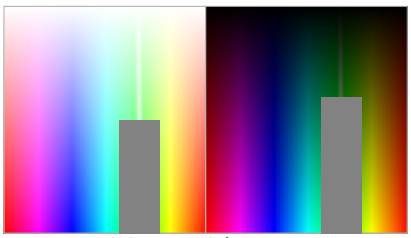

Minimum Saturation 0.6

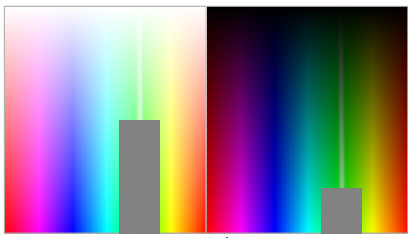
```
mixer 1-10 chroma 1 120 0.2 0.4 0.8 0.000 6 0.2 0 0 linear
```

min_brightness

The minimum brightness control sets the key to include only colours that match the saturation requirement ***and also*** have a brightness (luminance) that exceeds the set brightness value. This adjustment helps avoid keying on shadow areas of the source picture. The examples below show how the key signal changes with a fixed minimum saturation at 50% and minimum brightness values of 20%, 40% and 80%.


Minimum Brightness 0.2


Minimum Brightness 0.4


Minimum Brightness 0.8

The AMCP command for the third example (80%) is:

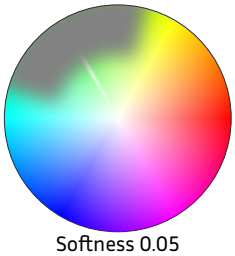
```
mixer 1-10 chroma 1 120 0.2 0.5 0.8 0.0 6 0.2 0 0 linear
```

softness

The softness, as it's name implies, controls the harshness of the transition between the active keyed area and other nearby colours in the picture. A hard edge often shows tearing caused by noise in electronic cameras.

The adjacent examples illustrate how a very small softness value has a significant impact on the transition between un-keyed and fully keyed.


Softness 0.0


Softness 0.05

spill_suppress

When the chroma keying source picture is from a physical studio set, rather than a drawing package, there may be low amplitude, low saturation reflections of the main key screen on objects within the scene. These areas are commonly called *spill*. Ideally such reflections should not be part of the keyed window. The spill suppress parameter attempts to avoid the miskeying.


Spill suppress 10


Spill suppress 20

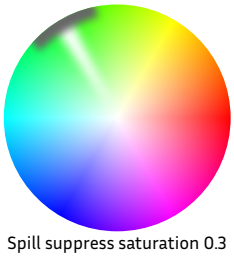

Spill suppress 30

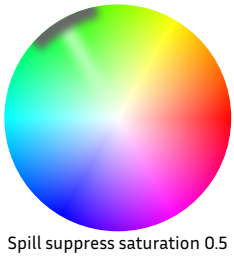
The three examples illustrate how the angle of the white triangle that starts at the centre of the circle and moves out to the circle radius both widens and softens as the suppress parameter value increases.

spill_suppress_sat

The effect of changes to the spill suppress sat control are relatively subtle, As the saturation value increases the chroma suppression on the keying axis reduces as illustrated on the set of colour wheels at the right.


Spill suppress saturation 0.1


Spill suppress saturation 0.3


Spill suppress saturation 0.5

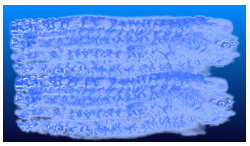
The AMCP command used for the centre example was:

```
mixer 1-10 chroma 1 120 0.2 0.7 0.8 0.04 15 0.3 0 0 linear
```

Mask - Alpha Channel Replacement

The mixer mask tool uses the red channel of the layer defined in the mask command as the alpha channel of the immediately above layer. Because white is an equal amplitude of red, green and blue the mask can use any monochrome source. The effect is illustrated below.

Layer 9


Layer 10

Layer 10 + Layer 9 Mask


Layer 5

Channel Output

The above example has 3 active layers - layer 5 holds the background picture, layer 9 has the mask, layer 10 has the fill for the mask active areas. When layer 9 Mask mode is activated it creates the content shown in the third image with an associated alpha for further compositing. The content of a lower layer, in this example layer 5, is combined with the masked output from layer 10 creating the final output.

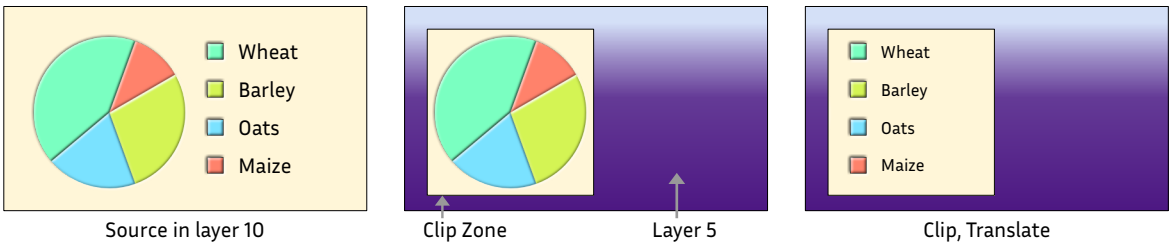
Masking can be used for many visual effects, including adding a film noise and scratch to low-noise electronic content. The mask process uses linear keying, enabling partially transparent zones to be created.

The AMCP command uses a single parameter to enable or disable the keyer. For example the command to enable the keyer for the above example is:

```
mixer 1-9 keyer 1
```

Clipping

The clipping command sets a viewport window in the channel output through which the clipped layer is visible. When position translation is applied to the clipped layer the contents seen in the viewport change as illustrated below.

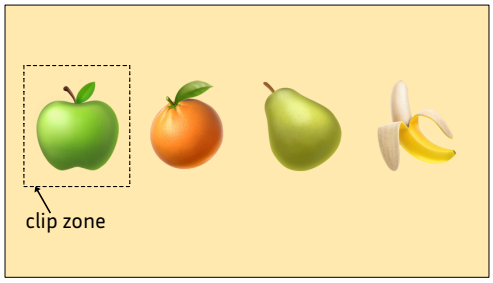


The leftmost picture shows a source in layer 10 of the channel. The centre picture shows a clip operation applied to layer 10, revealing lower layers such as channel 5. The right hand picture shows the result of a leftward x-axis translate of layer 10. The mixer clip AMCP command has the form:

```
mixer 1-10 clip <x> <y> <width> <height> <duration> <tween>
```

Where `<x>`, `<y>`, `<width>` and `<height>` are resolution-independent floating point values.

When the SVT client is used to set the clipping, the inspector property entry units are samples (pixels). These values are mapped to resolution-independent form when the command is played.

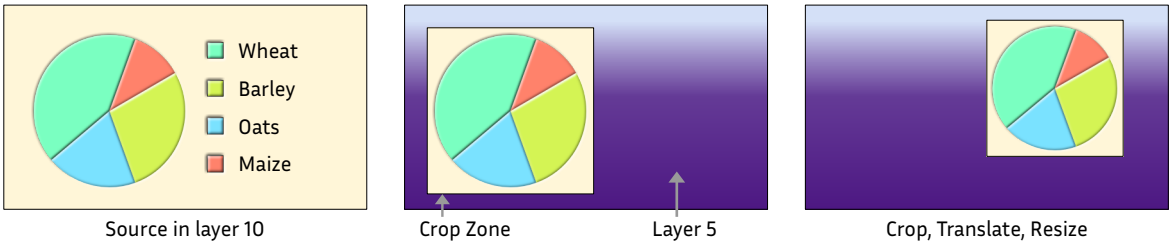


The picture to the left illustrates a simple application for a clip zone. The full frame picture contains four small pictures in a linear ribbon. The clip zone shown around the apple is set in the mixer layer that displays the ribbon (dashed edge).

Multiple translate commands are added to the client rundown. The first translate command moves the apple left out of frame bring the orange into the middle of the clip window. Other commands targeted at other layers can update other graphic or text content

Crop

The crop command creates a view window that can restrict the visibility of some of the layer source. The crop is applied to the layer content, then any translate and rotate operations are actioned. The results of a crop followed by a translate and re-size action are illustrated below.



CROP sets a view window. The cropped content can then be moved around the channel output window. The AMCP control parameters are the positions of the 4 edges, as illustrated by the format below:

```
mixer 1-10 crop <left> <top> <right> <bottom> <duration> <tween>
```

The control parameter values are resolution independent floating point values. The SVT client crop tool expresses the positions in samples (pixels).

Layer Volume

The layer volume control adjusts the gain for the specified layer. The channel gain is set by a single floating point parameter. Numbers greater than 1 increase the channel volume. The SVT client volume control widget supports gain values between 0% and 200%. The AMCP command syntax for a volume change to 75% over 25 frames is:

```
mixer 1-10 volume 0.75 25 linear
```

Layer volume and layer opacity can be grouped to create a show/hide instruction for live sources. Consider a remote traffic webcam whose stream is available. It can take several seconds to lock to and decode that source when the stream is played. Starting the stream and hiding it with opacity = 0 and volume = 0 means the audio-visual content can be instantly shown in a channel by setting opacity and volume to 1.

Master Volume

The mixer has an overall channel audio gain control. There is no SVT client widget for master audio volume control, just a simple on/off master volume button in the bottom right corner. The AMCP master volume control syntax is of the form:

```
mixer 1 mastervolume 0.5
```

There is no tweening control on the master volume, so changes can sound unpleasant. It is simpler to use the master volume just as a channel audio output level control.

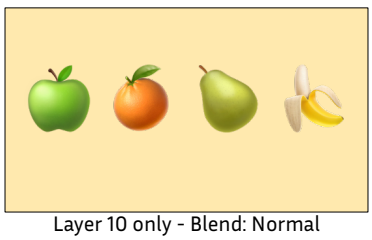
Layer-Blending Mode

Layer blending determines how two or more layers combine to create the composite result. Using blend modes opens a wide range of creative potential. It can be difficult to predict the result from using two source images and adjusting the blend mode, so allow time for experimenting with settings when blends are used.

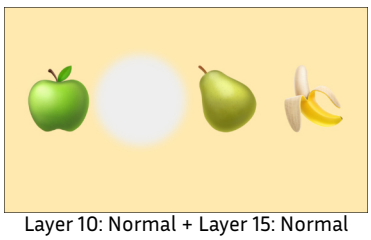
CasparCG server offers about 30 blend modes. When a channel is cleared the blend mode for each activated layer defaults to **normal**, a mode that implements standard alpha compositing. This blend mode enables templated graphic content from a higher number layer to be added to source content from a lower layer.

The default layer allocations in the SVT client create a simple implementation of layering because clips default to layer 10 and templates default to layer 20. The alpha content of the upper layer determines how the two layers combine, hence alpha compositing. The processing is applied to each R, G and B colour component. The alpha channels of the two layers are also combined enabling cascaded blending of multiple layers.

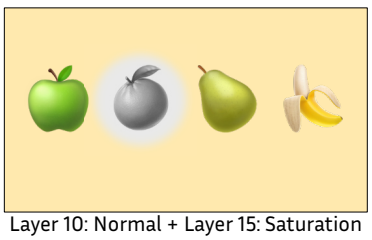
There are layer blend modes that operate in similar ways to other mixer effects, for example reducing the saturation. The significant difference is saturation blend operates at pixel level. This enables desaturation of a defined zone in the combined result. In the example below the centre picture has a defocussed white circle over a transparent background in layer 15, seen here with layer 15 blend mode set to **Normal**. In the right hand picture the layer 15 blend mode is set to **Saturation**. and the object under the layer 15 circle becomes desaturated.



Layer 10 only - Blend: Normal

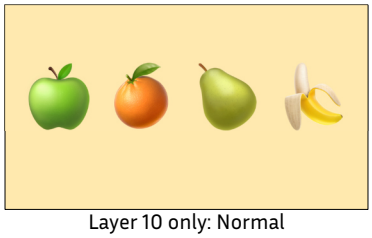


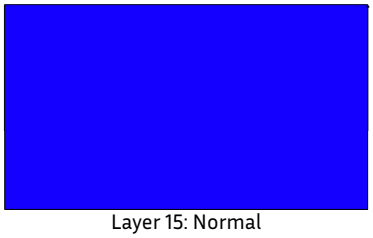
Layer 10: Normal + Layer 15: Normal

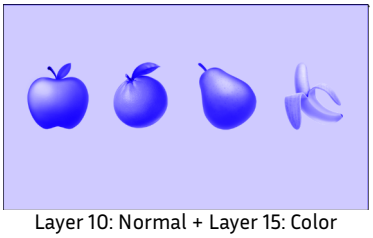


Layer 10: Normal + Layer 15: Saturation

Other blend effects include using a solid colour that replaces the background picture colours in the blended result. This **Color** blend mode is illustrated below.


Layer 10 only: Normal


Layer 15: Normal

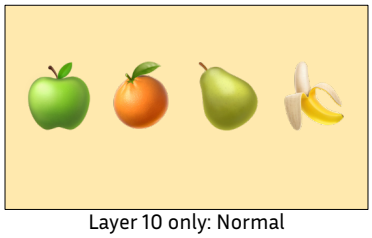

Layer 10: Normal + Layer 15: Color

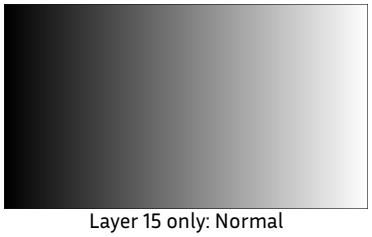
Blending operations enable backgrounds to create a context for other information shown on higher levels in the composite output.

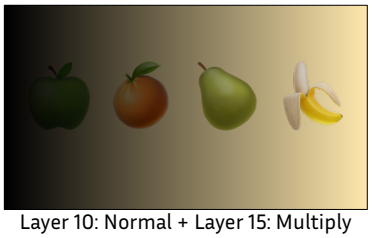
The **Multiply** blend mode is used to darken areas of a picture. As the name implies, the RGB component values from the same x and y positions in the upper and lower layers are multiplied together and re-normalised to the original range. A sample value of 255 is effectively a multiply by 1:

$$R_{blended} = (R_{lower} \times R_{upper}) / 255$$
$$G_{blended} = (G_{lower} \times G_{upper}) / 255$$
$$B_{blended} = (B_{lower} \times B_{upper}) / 255$$

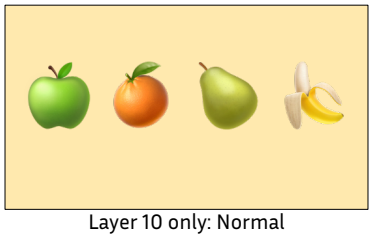
Any samples in the upper that are at maximum do not modify the source samples. The effect of a ramp in the upper layer with multiply blend mode active is illustrated below.

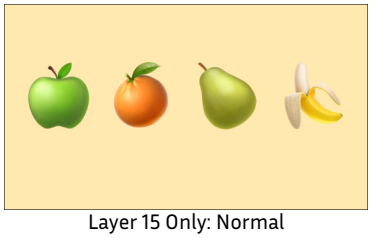

Layer 10 only: Normal

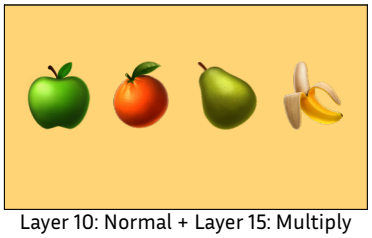

Layer 15 only: Normal


Layer 10: Normal + Layer 15: Multiply

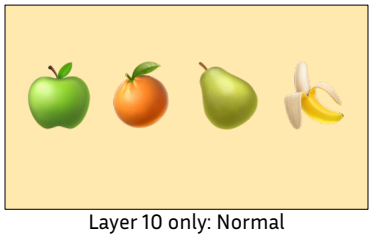
Where a source picture is blended with itself using multiply the result is a gamma modification as illustrated below.

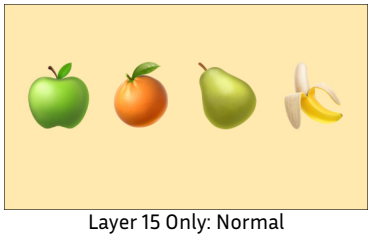

Layer 10 only: Normal

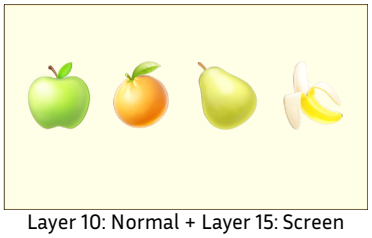

Layer 15 Only: Normal


Layer 10: Normal + Layer 15: Multiply

The **Screen** blend mode lightens areas of a picture because any area lower than white has increased values.


Layer 10 only: Normal


Layer 15 Only: Normal


Layer 10: Normal + Layer 15: Screen

The screen process involves inverting the information from both layers, multiplying the inverted value and inverting again. For a red sample the equation is:

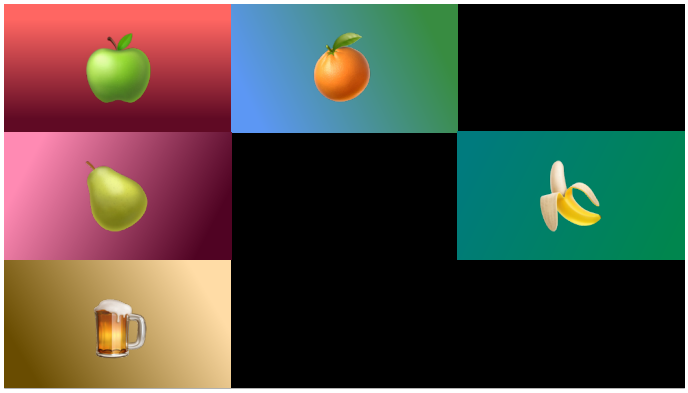
$$R_{blended} = 255 - (((255 - R_{lower}) \times (255 - R_{upper})) / 255)$$

One application for the multiply and screen modes is to add texture to pictures, for example adding film grain and noise to clean electronic pictures.

Grid

The grid command creates a multi-view style of display using the layers of the channels as the feeds for each monitor display. The grid command function is illustrated by the following two diagrams. The examples has 5 active layers, numbered as shown below each display in the top row

When the grid command is activated the display is split into tiles, the number of tiles defined by the parameter value passed to the grid command. Using a 3 by 3 grid with layer content as shown above gives the nine tile display shown below. Channel 1 is always displayed in the top left corner of the grid.



The AMCP command to create the mixer grid in channel 1 is of the form:

```
mixer 1 grid 3 <duration> <tween>
```

Note when the grid size is reduced after initial display it leaves orphaned tiles from the higher numbered channels at their original position.

If a duration is provided on the command line the grid forms dynamically using the tween. The duration is optional, default value 0, and the default tween is linear.

Reset

The operational parameters for each channel mixer command are stored in the working memory of the server. Restoring each changed parameter by a second mixer command requires many commands, and a parameter may be missed through operator error. The mixer clear command provides a fast restore to default value for either a layer specified in the AMCP command, or all mixer parameters for the channel. The two formats of the command are:

Level clear: `mixer 1-10 clear`

Channel clear: `mixer 1 clear`

The SVT client Reset widget implements the level clear command. Use a custom command to clear the entire channel.

Combining Mixer Functions

Many productions require contributions from several mixer functions to create the final graphic display. There may be several methods available to create the final effect, and the chosen solution is often determined by any subsequent adjustments that may be required.

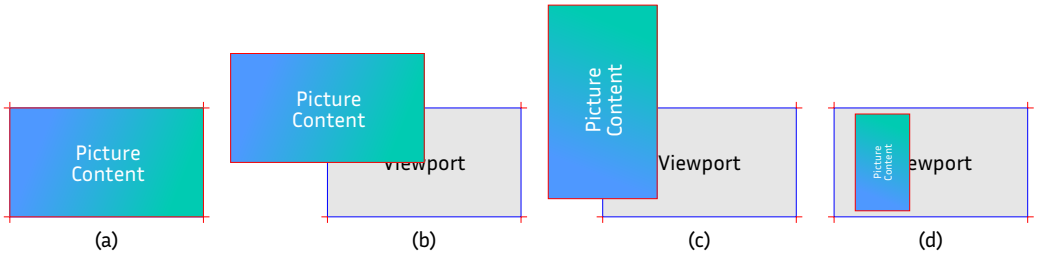
Consider the processing required to show portrait mode recorded mobile phone footage as part of a composite display with that footage plus a text quote. The picture source from the phone and the required composited display are illustrated below.

Portrait source as seen on standard 16:9 monitor

Composited Graphic

The source movie has to be scaled, rotated about it's centre and offset to the desired position in the composite result.

A method to achieve the output is using logical stages used to place and size the mobile phone content is illustrated by the following diagram.



Stage 1 loads the phone footage into the user selected layer of the channel. The phone picture fully fills the output viewport as shown in (a) above. The default rotate anchor position is the upper left corner at (0,0) in resolution independent coordinates.

Rotating the phone footage about the centre of the source needs the anchor position set at (0.5,0.5). This position is implemented by the **anchor** command. When these coordinates are set, the phone content moves such that it's centre is above the top left corner of the viewport as shown in (b).

A user sees the lower right quadrant in the clipping space set by the viewport. Applying the **rotate** command sets the layout shown in (c). The final manipulation uses the **transform** command to set the width, height and position of the top-left corner producing the display seen in (d).

The output can be achieved in less steps by using the Shape Distort command. The complexity of this solution is computing the target address points for the source content.

The rotate anchor is moved using the **mixer anchor** command. For example, setting the rotation centre to the middle of the picture uses an AMCP command of the form:

```
mixer 1-20 anchor 0.5 0.5 0 linear
```

The first and second parameters are the X and Y rotation centre expressed in resolution independent mode. The third parameter is the transition duration in frames (instant in the above example), with the fourth parameter defining the tween transition curve name.

The rotation operation uses the **mixer rotate** command. For example rotating the picture one quarter of a revolution uses an AMCP command of the form:

```
mixer 1-20 rotation 90 0 linear
```

The first parameter is the rotation angle, expressed in decimal degrees. The second parameter is the transition duration, with the third parameter defining the tween transition curve name.

The position and size of the picture are set using the **mixer fill** command. For example:

```
mixer 1-20 fill 0.2 0.5 0.45 0.45 0 linear
```

The first and second parameters are the x and y positions of the image. The third and fourth parameters are the width and height of the resized image.

The configuration requirements of the composited graphic used in the preceding example use instant transforms to set the size ,rotation and positions. Other graphic manipulations may need multiple mixer commands with visible transition durations where it is important that the mixer commands all start their transitions at the ***same*** instant. The asynchronous nature of the communications between the client and server could result in some commands starting later than others.

AMCP mixer commands have an optional **defer** parameter that instructs the command be placed in a queue. The queue is executed when a **mixer commit** command is received.

An example set of mixer commands using a defer operation is illustrated below:

```
mixer 1-10 saturation 0.5 25 easeinsine defer
mixer 1-10 clip 0.25 0.25 0.5 0.5 25 easeinsine defer
mixer 1-10 rotation 45 25 easeinsine defer
mixer 1-10 contrast 0.5 25 easeinsine defer
```

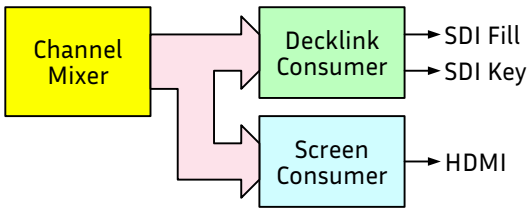
Finally the queue is activated by the command:

```
mixer 1 commit
```

Note this commit command triggers ***all*** layers in the channel that have queued commands.

The SVT client supports the defer mechanism via a simple tick box in the properties inspector of a mixer command. The mixer tools include the mixer commit command.

CasparCG Consumers



The server mixer stage composites the various source element layers into a single layer containing fill and alpha (key). The composited result is connected to a **consumer** for output.

A channel may have single or multiple consumers attached to the mixer output. The diagram to the left shows a channel with two outputs - one via SDI for use in a vision mixer and one for display on a computer HDMI screen. The supported consumers are summarised in the following table.

Screen

The video is sent to a window or to the full screen of the server host computer. The output screen area and position are initially set in the server channel configuration. If the screen output is windowed, the user can drag the size and position of the output to a different location. The channels audio output can be sent to the system audio output, where it is mixed with computer alert sounds etc.

The screen output can be set to fill a full screen area, a useful technique when a small CasparCG server is installed within a scenic element.

Scaling the windowed screen output can be a computationally intensive operation.

Decklink SDI
Bluefish444 SDI

Both Decklink and Bluefish444 consumers provide SDI outputs that can be used by a vision mixer or routed direct to other broadcast studio equipment such as video wall controllers. The server configuration defines the element or elements of the composited source routed to an SDI output. The output route may contain the full frame fill video element only, the fill and alpha/key outputs for use in a vision mixer, or just the alpha/key content.

The initial channel scan standard is set in the server configuration file, but can be changed during operation using a SET <channel_id> MODE command. Examples include:

```
SET 1 MODE PAL

SET 1 MODE 1080i5000
```

Disc

The Disc consumer enables a channels output to be recorded. The recorded file wrapper and codec must be defined. The Disc consumer can be dynamically added and removed to start and stop a recording.

Image

The image consumer is a special case of the Disc consumer. The Image consumer saves a single frame of composited content to the server media store. The file name is provided in the command that activates the image consumer.

Stream

The stream consumer applies data rate reduction to the composite output and sends the result over the network interface. Two types of stream producer are available - ffmpeg and NDI. The NDI consumer sends RGB fill and alpha to the receiver.

Route

The Route consumer has an output that is a route producer. A layer can be routed to another layer in the same channel, or to a layer ion another channel. The composite channel can be routed to a layer in another channel.

Consumers are dynamically added to and removed from a channel. When CasparCG server starts up it uses consumer allocations defined in the channel settings within the casparcg.config file. The consumer allocations can then be modified through two AMCP commands - **add** and **remove**.

The syntax of the add command is:

```
ADD [video_channel:int]{-[consumer_index:int]} [consumer:string] [parameters:string]
```

The optional consumer index is a numeric value that can be used in a subsequent remove command. If no consumer index is provided in the command, CasparCG server creates one for internal use. The following two lines show how to attach and remove Decklink device 3 for use in CasparCG server channel 4.

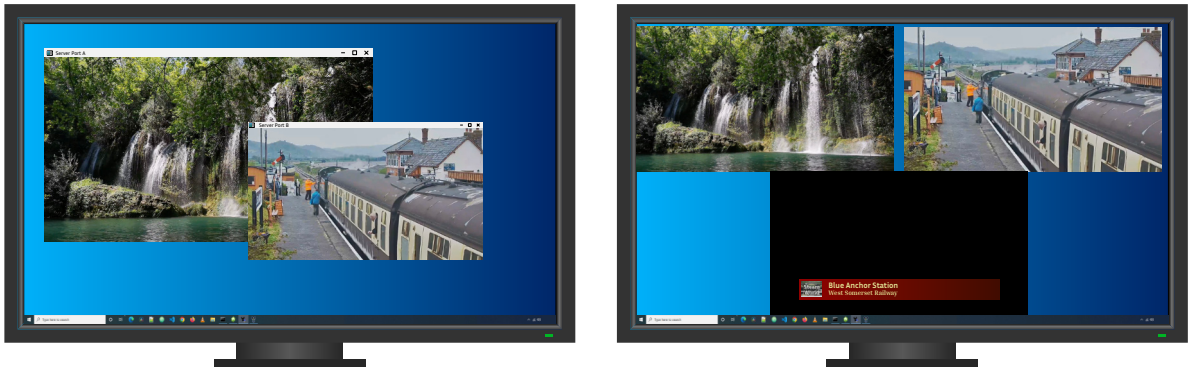
```
add 4 decklink 3

remove 4 decklink 3
```

Including a consumer index simplifies the text needed by the remove commands:

```
add 4-642 decklink 3

remove 4-642
```

Screen Consumer

The screen consumer shows the channel output on a computer monitor connected to the server. The channel may be shown windowed or filling the full screen area. The illustration below shows some of the screen consumer windowed options.



The screen consumer windows can have a standard control bar including close, minimise and maximise icons as shown in the left display above, or be borderless as shown in the right display above. If the window has a control bar that window can be positioned and sized using the mouse. Sizing is free-form and does not enforce an aspect ratio. The borderless fullscreen consumer is ideal where a dynamic graphics display is required as part of a scenic element. A small computer can be included in the scenery structure, with the screen consumer displays controlled via a wired or wireless network connection.

A screen consumer can be added when the server is started, or via a custom command. Configuration as part of the server startup has more control parameters than available with a custom command consumer add. The configuration properties are summarised below.

device	The windows screen number.
aspect ratio	There are three options: default, 4:3, 16:9
stretch	Options are none, fill, uniform, uniform_to_fill.
windowed	When set false the output fills the full screen area.
key-only	When true, the key is shown in the assigned screen area.
vsync	Synchronisation to the screen vertical refresh. Default is false.
borderless	When true, the menu bar at the top of window is removed.
interactive	This property is currently disabled in server 2.3. Earlier servers, such as version 2.1, sent mouse position and click data from the window to the CEF client.
always-on-top	Forces the window to display at the frontmost position of the open windows.
x	Set the left side of the window at this position.
y	Set the top edge of the window at this position.
width	Set the window width to this value.
height	Set the window height to this value.
sbs-key	Sets the window to output a key at the right side of the fill. Used by Datavision adapters.
Colour-space	Default is RGB. Can have other values when used with DataVideo TC-100 and-TC 200.

It is possible to set the width or height as zero, causing the value to be computed from the aspect ratio property and the provided, non-zero, value.

The right-side of the consumer examples is created by defining the x, y, width, height, and borderless properties.

A screen consumer may be added to a channel using the **add** command. This command supports a limited sub-set of the full screen consumer properties. For example, a full screen output can be attached by a command:

```
add 1-1001 screen 1 borderless fullscreen
```

The supported parameters are the screen number, fullscreen, key_only, sbs_key, non_interactive, borderless, name (shown at top of screen with a border present). Only the name property accepts any property value.

Streaming Consumer

CasparCG server supports two streaming mechanisms - NDI and ffmpeg toolkit. The stream output is created by adding the stream to an existing consumer via an add AMCP command, or it can be created as part of the server channel startup process using entries in the server configuration file.

An NDI stream output of a channel can be created by including 4 lines in the consumers segment of a channel in the server config file. For example:

```
<consumers>
  <ndi>
    <name>Casparcg_NDI_01</name>
    <allow-fields>true</allow-fields>
  </ndi>
</consumers>
```

The above definition creates an NDI output stream called “Casparcg_NDI_01” set to support interlaced transmission. Set the <allow-fields> property to false force progressive scan output.

Creating an NDI output from a channel without restarting the server uses an AMCP command to add the consumer. An NDI name, for example "**ccg_01_ch03**", can be provided as part of that command. If no name is present the consumer uses a number as the channel name. Surround the name with quotation marks when the name includes spaces. Include the text **ALLOW_FIELDS** to instruct the consumer to send field based output. For example:

```
add 3-1003 NDI NAME ccg_01_ch03 ALLOW_FIELDS
```

Because the above example includes a consumer index, the NDI output can later be removed using the short form AMCP command.

```
remove 3-1003
```

The second streaming process uses ffmpeg as the encode and multiplex tool. The command line for such streaming can be very long because of the many configuration parameters required. Some command line switches are sensitive to their placement within the command line string. The following description is an overview of the process, not an in-depth tutorial.

The SVT standard client uses the ffmpeg streaming mechanism to provide user selected monitoring in the client “Live Preview” window. If the live preview panel is enabled in the client (client configuration switch), the client selects the fill or key output of a channel through an AMCP command to the server that starts the streaming.

The command sent from the client to monitor channel 1 fill is shown below.

```
ADD 1 STREAM udp://<client_ip_address>:9250 -format mpegts -codec:v libx264
-crf:v 25 -tune:v zerolatency -preset:v ultrafast -filter:v scale=288:162
```

Note: the command is sent on a single line, the two lines shown above are split so the text is legible.

The stream is directed to a client through the special encoded parameter `<client_ip_address>`. The token is replaced by the actual tcp/ip address of the invoking client.

When the user changes the source signal being monitored, or ceases monitoring, a remove command is sent to the server channel:

```
REMOVE 1 STREAM udp://<client_ip_address>:9250
```

A network output stream uses a similar command to the client preview stream. The following example command streams channel 1 video using the H264 video codec to a selected receiver, such as VLC, running on the same PC as the CasparCG server that creates the stream.

```
ADD 1-1001 STREAM udp://127.0.0.1:9999?pkt_size=1316 -codec:v libx264
-tune:v zerolatency -preset:v ultrafast -crf:v 25 -format mpegts
```

The packet size property is not needed on all systems, but may be required to ensure content passes through switches when directed to a remote receiver. The final switch of the above command line sets the multiplexer used for the stream. After a -format switch the rest of the command line is often assumed to be parameters for the format property, hence placing it at the end of the command line minimises potential interpretation issues. The full list of switches available to control ffmpeg is available at https://ffmpeg.org/documentation.html

VLC can be used to display the network stream. Open VLC, then select network stream in the VLC menu. Enter the string below in the source box of the displayed form.

```
udp://@127.0.0.1:9999?pkt_size=1316
```

A short-form VLC stream receive command can also be used:

```
udp://@:9999?Pkt_size=1316
```

A channel can be configured to start streaming when the server starts. This is implemented by including the streaming commands in the channel consumers segment of the casparcg.config file.

```
<consumers>
  <ffmpeg>
    <path>url</path>
    <args>[most ffmpeg arguments related to filtering and output codecs]</args>
  </ffmpeg>
</consumers>
```

Note server version 2.3 uses a build of ffmpeg that requires full argument and filter names, not abbreviated names. Use **`-codec:v`** *not* **`-c:v`**

Streaming and Recording Computation load

Creating a stream, either for network transmission or recording, can impose a significant computation load on the host computer. Simpler, often older, codecs use less computational resource than more advanced codecs. The following table is only an approximate set of values, but it provides a useful guideline. It uses MPEG-2 video codec as the base reference. In the table DCL is the Decoder Computational Load and ECL is the Encoder Computational Load. Note 1 ECL is many times larger than 1 DCL.

Codec	Decoder	Encoder
H.262 (MPEG-2) Video	1 DCL	1 ECL
H.264 Video (AVC)	3 - 4 DCL	10 ECL
H.265 Video (HEVC)	9 - 16 DCL	100 ECL

CasparCG server uses a graphics processor to support various internal processes, such as channel mixing. CasparCG requires OpenGL version 4.5 or better, but does not specify a specific graphics card or processor device, thereby supporting the maximum potential selection of hardware platforms. One consequence is that creating an output stream uses only CPU resources.

Many GPU manufacturers, such as *n*Vidia, have hardware support for some streaming encode operations especially those required in H.264 and H.265. There are versions of ffmpeg that can leverage that processing power. System installers of CasparCG systems that require streaming outputs should carefully measure the effect of the streaming on CasparCG operation, considering if it is better to use a piping mechanism to send the channel output to a separate streaming encode process. This encode process may be an instance of ffmpeg that exploits the GPU chip, or use an external hardware processor such as the Blackmagic Design Web Presenter, AJA Video HELO, or a PC running OBS.

The connecting pipe uses SDI or NDI, NDI being a computationally relatively light compression and multiplexing process. An SDI pipe offers the option to run a hardware signal quality assessment that reports out-of-specification content to a central monitoring system.

A test conducted by the author used Windows Task Manager to report the CPU activity and load statistics with and without streaming. Two 1080i25 channels were playing content to SDI outputs. An H.264 stream added to one output channel used 25% of the CPU capacity. An NDI stream added to the same channel in place of the H.264 stream used 10% of the CPU capacity.

File Recording

CasparCG server includes the ability to record the output of a channel. This allows graphics elements and title elements to be passed to the edit operation ensuring consistency of look and feel to the entire programme content. Because CasparCG records the output of the channel mixer process, the recording can include and live inputs, clip replays and other graphic layers.

Like the streaming consumer, the disc consumer uses ffmpeg to encode video and audio. In general the disc consumer has a high-bandwidth output allowing a user to select high-bit rate intra codecs where significant post-production processing is required, or use a lower bit rate inter-frame codec where record of transmission of simple archiving is needed.

The file recorder uses the AMCP `add` and `remove` commands to start and stop recordings. An example add command for file recording is:

```
ADD 1 FILE newfile.mov
```

The disc consumer has defaults for all parameters, but it is normal to provide detailed audio and video codec properties on the AMCP command line. The recording is stopped by the remove command:

```
REMOVE 1 FILE newfile.mov
```

The AMCP documentation includes a SEPARATE_KEY switch that instructs CasparCG to record the channel into two files - newfile.mov containing the fill video and newfile_a.mov containing the alpha channel. For example:

```
ADD 1 FILE newfile.mov SEPARATE_KEY
```

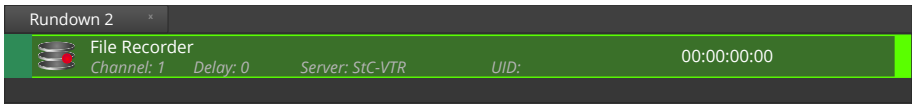
This worked on earlier versions of the server, but has been removed from server version 2.3. If the switch is present in the command it is ignored.

Including the consumer index in the add command simplifies the stop recording command. For example:

```
ADD 1-709 FILE newfile.mov

REMOVE 1-790
```

The server does not check the media store to see if the filename already exists. If a media file of the same name provided in the add command exists it is ***overwritten*** by the new **add** command. The server cannot guess the users intent to replace or create unique filenames. A user must either manually check the name used, or create a custom client that does the unique name test and edit to force unique operations. The SVT client does *not* include name checking.

The SVT standard client includes a tool that can be used to simplify creating the command line for a recording. The File Recorder tool is available in the Tools... Other library group, or via the right click context menus in the rundown. The summary properties in the rundown listing are illustrated below.



The tool provides some default properties including selecting Channel 1 and setting delay and duration times to zero. It also disables the output inspector layer properties and target properties. The user selects the target CasparCG server from the standard client drop-down server name list.

The normal function keys (**F2** Play, **F1** Stop) control the start and end of a manually controlled record. The user may also set a record time duration in milliseconds using the Output inspector duration property, and set the recording to be controlled via a remote trigger such as an OSC message. If other elements need to be running before the recording starts, the delay field value for this item is set to start the recording after other elements stabilise. The latency between issuing the record command and the first recorded frame is short (typically 3 to 6 frames).

The File Recorder properties inspector is illustrated below.

Enter the name of the file to record into the Output entry field. This entry contains two elements, the stem name of the recording and the file wrapper to use. The file recording is stored in the media folder defined in the server configuration file. If the user wishes to make recordings in a sub-folder of the media store, the name of the sub-folder can be included in the file name, for example **ValeView/Recipe_of_week.mxf**. When the record operation starts the folder is created if it does not already exist.

The Codec property does not directly set the record codec properties. The user selects a codec name from a drop down list defined when the CasparCG client was built. Selecting a different codec name in the box causes two property changes - the file wrapper extension in the Output entry box, and the string initially shown in the Preset entry box. *If a sub-folder name was present in the output file name the codec change deletes the name of the folder from the file name.*

The content of the Preset entry box is the properties and filter string passed to the ffmpeg encode process. There is a default string linked to the codec name. The user may freely edit the string in the Preset box to user selected values. The preset string in the illustration is valid for an interlaced scan high-definition channel. The example string contains 5 property values, described in the following table.

Property value	Control function
-codec:v dnxhd	Defines the data-reduction intra frame compression process applied to the video signal.
-b:v 120M	Sets the target bitrate for the video to 120 Mbits/s. The dnxhd codec has a set of standardised bitrate options that depend on mode and resolution.
-flags:v +ilme+ildct	Sets the codec to use interlaced motion estimation (ilme) and interlaced mode Discrete Cosine Transform (+ildct).
-threads:v 4	Limits to four the number of threads the video encoder may use.
-filter:v format=yuv422p,interlace	Set the video as 4:2:2 sample structure using interlace

If the channel is reconfigured from 1080i5000 mode to progressive 1080p2500 the default preset string is invalid. A progressive scan channel preset string should be simplified to:

-Codec:v dnxhd -b:v 120M -threads:v 4 -filter:v format=yuv422p

Once a useful preset string is established there are two methods that speed the re-use of the settings.

1. Set the file name to one that is unlikely to be used, for example EditThisName.mxf then save the File record item as a named library preset in the standard SVT client. Future recordings recall the preset, editing the file name to a descriptive one ready for use.

2. Build an end-user variant of the standard client where all commonly required preset strings are built into the client code. This increases the number of elements in the dropdown codec list, enabling very quick setting of the encode properties. This needs some experience in building code, but all required tools are available as free-to-use. The detail of the software edit process is given in Appendix G.

Preset strings can be very long. An improved default interlaced preset string that sets lots of important properties in the output file is:

```
-codec:v dnxhd -b:v 120M -flags:v +ilme+ildct -threads:v 4 -filter:v interlace,
scale=in_range=full:out_range=tv:out_color_matrix=bt709,format=yuv422p
-colorspace:v bt709 -color_primaries:v bt709 -color_trc:v bt709
```

Several kind individuals have posted some example recording commands for specific wrappers and codecs in the CasparCG support forum. One such example is the settings for making an XDCAM 35 Mbits/s stereo file using 4:2:0 sub-sampling, 1080 line interlaced 25 frames per second. The command line is shown below.

```
-b:v 35000000 -codec:a pcm_s24le -codec:v mpeg2video -filter:v interlace,
zscale=rangein=full:range=limited:primaries=709:transfer=709:matrix=709,
format=yuv420p -alternate_scan:v 1 -g:v 12 -bf:v 2 -minrate:v 35000k
-maxrate:v 35000k -color_primaries:v bt709 -color_trc:v 1 -colorspace:v 1
-filter:a pan=stereo|c0=c0|c1=c1
```

CasparCG server version 2.3 uses 8-channel passthrough as the default audio configuration. The above configuration includes an audio filter instruction, -filter:a, that defines the audio as stereo and maps the first two audio input channels to the stereo output. The equivalent preset string for an XDCAM 50 Mbits/s 4 audio channel 1080 line interlaced 25 frames per second is:

```
-b:v 50000000 -codec:a pcm_s24le -codec:v mpeg2video -filter:v interlace,
zscale=rangein=full:range=limited:primaries=709:transfer=709:matrix=709,
format=yuv422p -alternate_scan:v 1 -g:v 12 -bf:v 2 -minrate:v 50000k
-maxrate:v 50000k -color_primaries:v bt709 -color_trc:v 1 -colorspace:v 1
-filter:a pan=4c|c0=c0|c1=c1|c2=c3|c3=c3
```

The audio in the above preset string has 4 output channels, mapped from the first 4 input channels. More information about using ffmpeg audio filters to create various channel mappings and mix-downs is available at https://ffmpeg.org/ffmpeg-filters.html#pan-1.

Image Consumer - Still Image Capture

Still images may be imported to CasparCG server after processing by a graphics package, or the image may be captured (grabbed) from a video clip or live video input. The capture process is a special version of the **add** command that auto-terminates on execution. The following description illustrates the capture process from an SDI input.It is desirable that the live input can be seen on the CasparCG channel used for the capture. Channel 3 and Decklink input 1 are used in the following example.

There are four steps to the image capture process:

1. Set the Decklink or Bluefish card input to the desired source using the SDI video router or mixer aux bus.

2. Route the SDI input channel to an SDI output channel to enable viewing of the input source.

3. Instruct CasparCG to take an image capture from the source. Repeat this capture operation as many times as required.

4. Remove the CasparCG route that is sending the SDI input signal to the viewing channel (STOP the route).

There are two AMCP methods available for the grab event in step 3 - **print** and **add image**, and the SVT standard client includes a snapshot tool that uses the add image process.

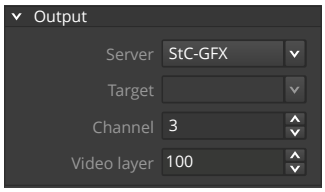
Preparing to grab images

Route the desired grab source to the CasparCG server SDI input.

Do not mix the programme replays and the grab operation in a single rundown. Create a new rundown using the File Menu, New Rundown, or use the shortcut key combination Ctrl N. Make the new rundown the active rundown. There are two methods to set the required input:

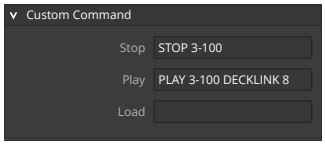
Method A

Add an instance of the **Custom Command** to the empty rundown. Edit the metadata label to indicate the purpose of the command - for example **Grab Input Control**. Set the properties of the output and custom commands using the Inspector panel:



i.) Select the CasparCG server from the drop-down list (StC-GFX in this example).

ii.) Set the channel to 3 and set the video layer to 100. Using layer 100 leaves lower layers unmodified by the grab operation (hidden by the input full-frame signal).



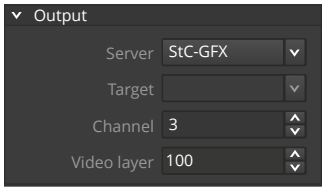
i.) Enter text into both the Stop and Play command boxes. The text case does not matter in the commands.

Stop box: **STOP 3-100**

Play box: **PLAY 3-100 DECKLINK 1**

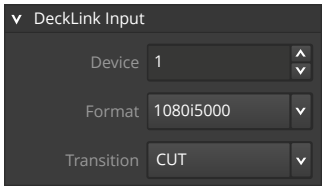
Method B

Add an instance of the Decklink Input command to the rundown. This command is available in the Tools Other section of the tool palette, or it can be inserted using a right click over the rundown display, then selecting Tools → Other → Decklink Input from the context menus. Use the Inspector panel to:



i.) Select the server (StC-GFX in this example)

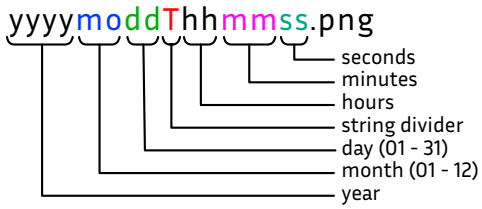
ii.) Set the Output properties to Channel 3, Layer 100 (examples)



iii.) Set the Decklink Input Properties. Device 1 and input format 1080i5000

The input format value is important. If the SDI input signal format does not match the value in the Format field there is a black video input.

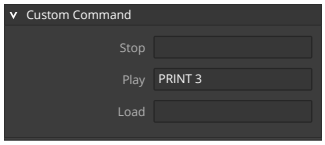
Capture a still using AMCP Print Command



The print command captures a single frame, storing the captured data in a PNG file. The file name is auto-created from the capture date and time, expressed in a modified ISO 8601 date and time format as illustrated by the diagram at the left.

The captured file is placed in the root of the CasparCG server ***media*** folder. Files captured using print **should** be renamed to more human readable versions. Good practice also moves captured stills into a programme-related media sub-folder.

The CasparCG SVT client does not have a pre-defined tool to issue the print command. Use a custom command instance to run the print command.

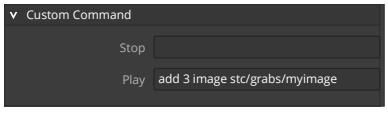


Add a custom command to the rundown in use for grab operations. Set the metadata label to Capture Still or similar ident. Set the server name in the Output inspector properties.

To capture the preview output of channel 3 use the string PRINT 3 in the Play box of the custom command properties edit form. Each time the F2 key is pressed a still is captured from the signal feed on output 3. To capture a different channel edit the number after the PRINT command. This capture method is very effective when a number of slides need capturing in a short time.

Capture a still using Add Image Command

An advantage of the add image mechanism is captured picture is named as part of the capture instruction. The following descriptions assume an SDI video input has been routed to channel 3 layer 100.



The filename is provided as part of the capture command. Multiple custom command can be created, one per image capture, or a single capture command can be used, editing the file name element prior to the grab.

Add an instance of Custom Command to the rundown. Select the CasparCG server using the properties Inspector. Enter text into the Play box of the custom command using a format similar to:

```
add 3 image ValeNews/fire_chief
```

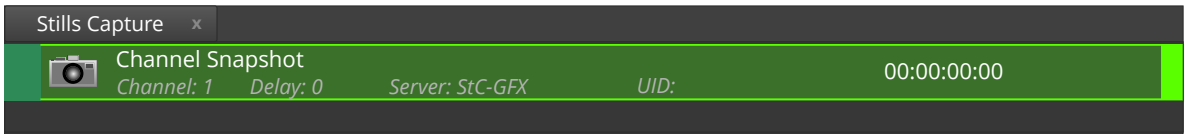
This command, when run using the F2 key, adds the image producer to the channel and layer selected in the command, captures the still to a file named in the command, then removes the image producer.

When the file name includes a path, as shown in the above example, the path is *relative* to the CasparCG Media folder on the server, and that path ***must already exist***.

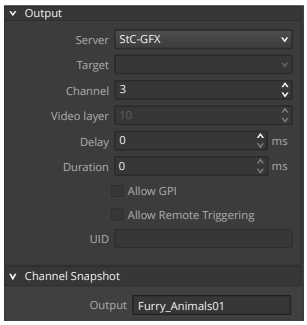
Capture using the SVT Client Channel Snapshot tool

The SVT client includes a Channel Snapshot tool that works the same way as the generic custom command add image process described above. The tool is available from the Tools... Other library inspector, or via the right-click Tools... Other context menu.

The snapshot tool in a rundown list has the camera icon at the left of the rundown item, as illustrated below.



If a programme episode requires several stills, grabbed for that specific episode then no longer required the channel snapshot can be allocated a remote control URL and the named capture overwrites any older item. The replay entry in the main rundown then needs no editing. In such uses the Item Metadata field should be edited to describe the function of the captured still.



The inspector properties for a channel snapshot instance are shown at the left. The target (source file name) and video layer properties are disabled. The CasparCG server name and channel number to capture are set in the output panel section of the inspector.

The capture filename is set in the Channel Snapshot Output field. The field may include a folder name. That folder must already exist in the media store or the capture will fail.

Restore Capture Channel settings

When all captures are complete, select the rundown item that routed the source input to the channel layer, then press the F1 (stop) key to restore the normal replay display operations for that channel. If there is active content on one of the lower layers in the channel the lower layer or layers will again show on the channel output.

If the grabbed images are required in the future, the images should be copied from the media folder or sub-folder onto an archive store.

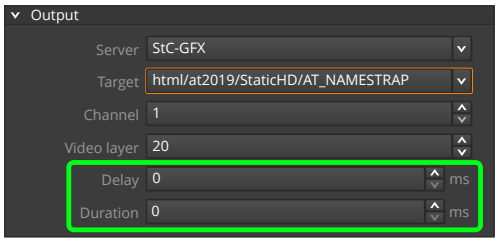
Finally use the client library refresh command, Ctrl R, to ensure the client library lists are up to date.

Advanced Operations

This section describes some advanced modes of control.

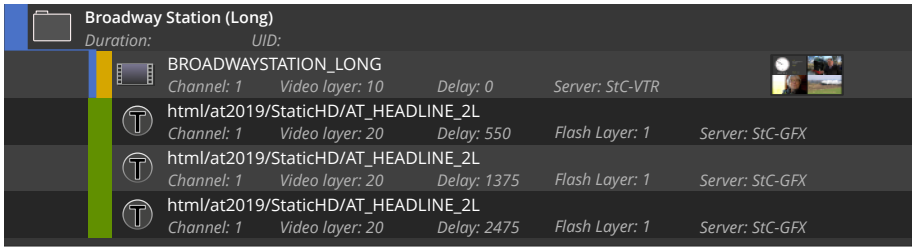
Auto Timed Caption Replay

Edited news stories may need multiple name captions overlayed during clip transmission. The insertion and removal of the caption can be triggered manually under the guidance of the production member timing the programme.



The CasparCG standard client can be configured to insert and remove captions on a channel at times relative to the start of the clip. The mechanism uses a group to combine the clip and associated templates. The delay and duration fields of each template are set in the template Output panel inspector. These fields are identified by the green rectangular ring on the example panel at the left.

The template is instanced multiple times as needed for the clip. The video replay command and all template instances are grouped together. The **Delay** value gives the time, relative to the start of the video clip, when the caption starts its input animation. The **Duration** sets the time, relative to the start of the input animation, when the caption starts its output animation.



In the above example, the video clip called **BROADWAYSTATION_LONG** uses three captions. Hence there are three template entries in the group, each entry has a delay relative to the start of the video clip. The duration value of each template is not shown on the template summary on the rundown.

The delay and duration times can be entered as frames or milliseconds. The entry mode is defined in the CasparCG client configuration.

The user selects the group item and starts playout using the F2 play key. The client issues template play and stop commands at their defined times.

Virtual Channels

CasparCG supports the use of ***Virtual Channels***. Such channels have defined properties such as the video standard, but are not routed to an output consumer.

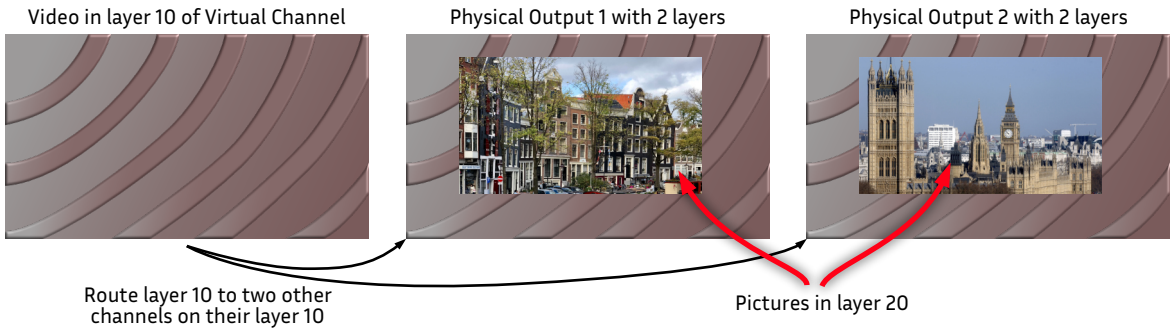
Virtual channels can receive and execute AMCP control commands. The commands are executed just as if there was a connected physical output. Virtual channels have access to the channel mixer/keyer, so they can add and remove templated graphics above a video clip or create a picture-in-picture overlay.

CasparCG server supports the use of both single layer routing and composited channel routing. The virtual channel or virtual channel layer can be routed to multiple physical channels. This mechanism supports applications such as sending a moving background to appear behind picture overlays, and splitting a high-resolution video clip to feed multiple in-vision monitors such as a monitor wall.

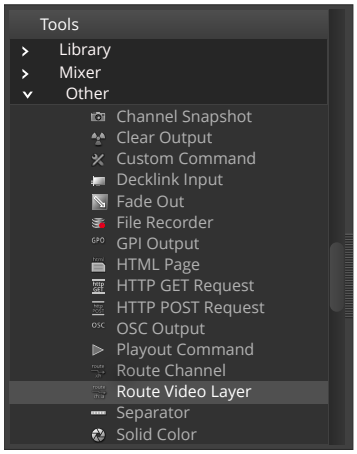
Virtual channels are defined in the CasparCG Server configuration file and are created when the server software starts.

A Common Background for Multiple Output Channels

The images below illustrate how a virtual channel can be sent to two physical channels. Each physical channel performs layer mixing to overlay a reduced size still over the common shared background.



Using the route process ensures the two backgrounds of output 1 and output 2 remain in sync with each other, enabling a vision mixer to transition between the two outputs with no temporal offsets on the background areas.

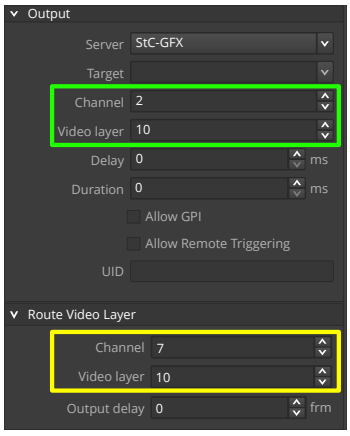


The CasparCG SVT client has a command available that enables a channel or layer route without the operator needing detailed knowledge of the underlying AMCP command.

The illustration to the left shows the Tools tab of the client, with the Other tools section expanded. The route video layer command is highlighted.

The selected command, route layer or route channel, is inserted into the rundown by double-clicking the tool or by click-dragging the tool to the rundown panel.

Select the route command in the rundown panel to inspect the source and destination routes.



The route properties panel for a channel layer route is illustrated at the left. A green rectangle outline shows the route ***destination*** settings, and the yellow rectangle outline shows the ***source*** channel and layer.

When the route command is played (key F2) it makes the route sending the AMCP command:

```
play 7-10 route://2-10
```

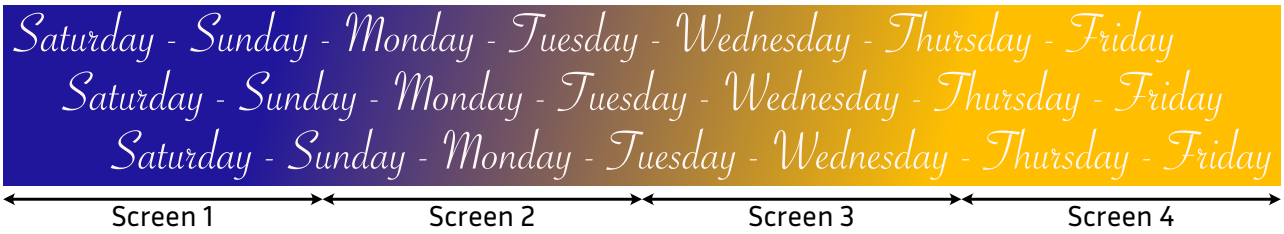
Before or after the route is activated the virtual channel can be given a command to play an image or a video that then appears on the routed layer of the destinations of the route or routes. There is a small latency (1 frame) added by using the route layer mechanism.

The example route illustration near the top of this page shows extra layers added to form the production requirement composite result, using the CasparCG server's mixers ability to change the size and position of a layer element.

Creating a Monitor Wall

A high-resolution virtual channel can be used to drive multiple display monitors placed such that they form an in-vision monitor wall. Assume four in-vision monitors are needed as part of the set, and that the video content flows from left to right on the video wall. The video wall content is played in a high-resolution virtual channel using channel routing plus the channel mixer to create the physical set of outputs that drive the monitors. The example that follows is based on using an Ultra-1 HD source creating four HD screens.

The video for the wall is created using a software tool such as Adobe After Effects or Apple Motion to create a file that is 7680 x 1080 pels - the size of four full-HD screens adjacent to each other. This file can be designed as a seamless loop, or it can be rendered with sufficient duration that a single play lasts longer than the programme that uses the clip. A simple example of the quad-size source is shown below.



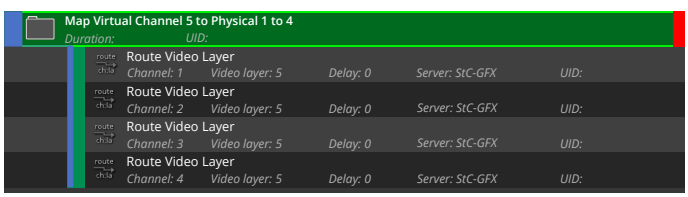
The second stage process uses a video edit package to create an Ultra-1 HD video clip. Two copies of the super-wide source are placed one above the other using a position offset on the lower copy to keep the right-hand side of the source. The resulting Ultra-HD clip is as shown below, where a light-green dotted box has been added to show the display targets for the four quadrants.

CasparCG server plays the Ultra-HD clip in a virtual channel, using layer routes plus mixer size and move commands to each of the four physical channel outputs.

A production could require one or more layers added in front of the wall content. Video clips and stills default to layer 10, hence a good operational choice is placing the video wall content on layer 5 to avoid later “finger trouble” when adding any overlay elements. It may be operationally easiest to use the same layer number for both the virtual and physical channels.

Source			Destination	
Channel	Layer		Channel	Layer
5	5	→	1	5
5	5	→	2	5
5	5	→	3	5
5	5	→	4	5

Assume that the physical outputs are channels 1 to 4 and the virtual channel is number 5. The 4 routes needed are shown in the adjacent table.

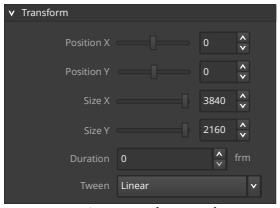


Placing the four route commands in a group provides a simple mechanism to activate the four routes by a play command directed at the group. After creating the group edit the group label to identify the function of the group.

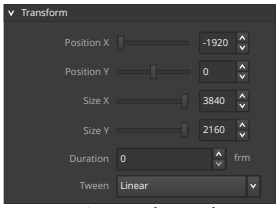
The inherent auto-fit operations in CasparCG mean that the routed Ultra-HD signal is shrunk to fit the HD resolution output channels. The size change is auto-implemented by the graphics mixer in CasparCG.

That default re-size operation is modified by further commands from the CasparCG client, instructing the mixer not to reduce the size but to apply position offsets such that the quadrants of the source are mapped onto the HD outputs of the physical channels.

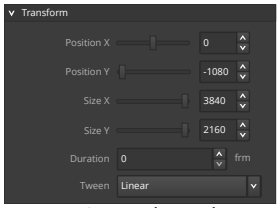
Four transform commands are required, one for each physical output. Placing these transforms in a group allows speedy application of the offsets and sizes, and the commands can be included in the routing set-up group. The transform properties for each channel are shown below.



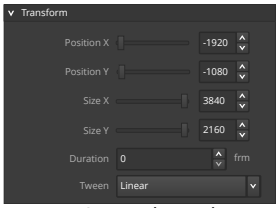
Settings - Channel 1



Settings - Channel 2

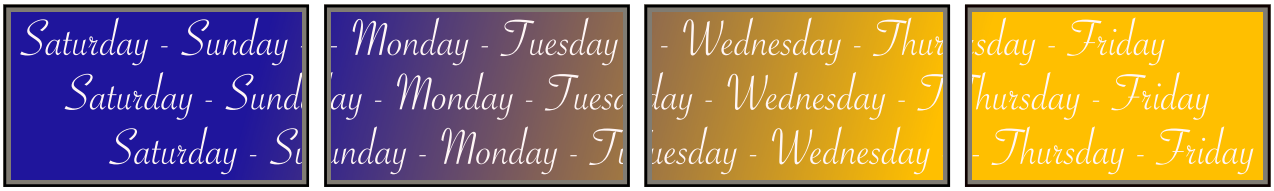


Settings - Channel 3



Settings - Channel 4

When the video is played in the virtual channel, it creates a video wall display similar to the one shown below.



It may be desirable to separate the configuration of the video wall from the playback of content on that wall. This is simple to implement by using multiple rundowns - one containing the configuration commands and the second containing the content playout commands.

Mixer Commands - Defer

The CasparCG server mixer supports a large range of image manipulations including adjustments of size, contrast, rotation, position cropping, masking and changing layer blend modes. A mixer transitions can be instantaneous (cut), or set to complete over a user specified time. The available mixer commands are listed in the tools tab of the SVT client.

Multiple manipulations on a single channel layer send a sequence of commands to the server, each command acknowledged by the server. This sequential transmission can cause a problem where some transformations intended for simultaneous start actually start at offset times. The biggest issue is the time offsets are not consistent when a command sequence is re-run. The random offset can be avoided by using the ***Defer*** facility in the mixer commands.

In the mixer command transform properties inspector panel there is a **Defer** tick box. Set the Defer flag on all the transformations that are in a sequence group. When the server receives a deferred mixer command it places that command in a stack awaiting a "go" command.

The sequence of commands is activated by a ***Commit*** command. When this command is processed in the server it will action all of the items in the delayed process stack. This mechanism ensures a reproducible effect. The commit item is available in the Tools panel, Mixer command set.

Video Clip Random Access

Video clips use the ffmpeg producer to read and decode frames from a source file. When the file is loaded or played the AMCP command can set start and end points that are not the first and last frames of the clip. The start point is set by the **seek** parameter, and the end point is set by the **length** parameter. The clip can be played in a loop by including the **loop** keyword in the command.

The ffmpeg producer includes a mechanism that allows updates to the seek frame number, clip length, loop mode and playhead position. The update process uses the AMCP **call** command. The general syntax for this command is:

```
call [video_channel:int]{-[layer:int]|-0} [param:string]
```

Update loop mode

The clip loop mode is set by using the call command with the parameter string starting containing the reserved property "loop" plus a value that can be interpreted as a boolean value. The following examples show how to activate (example 1) and deactivate (example 2) loop mode:

```
call 1-10 loop 1
call 1-10 loop 0
```

Update In point and Duration

The clip in point value is set using either of two keywords - **in** or **start**. The following two lines both set the first played frame value to frame 125.

```
call 1-10 in 125
call 1-10 start 125
```

The last frame number is set using the **out** keyword as a parameter.

```
call 1-10 out 750
```

The user can also set the length of playback, and the call command computes the last frame number using the in/start number added to the new length.

```
call 1-10 length 280
```

Update playhead position

The call command can change the playback position to a new frame number. The playhead position can be set to a user provided frame number, the first or last frame defined via the in and out properties, or relative to a position, for example 200 frames ahead of clip end position.

If clip playback is active when the seek position is updated, playback continues from the assigned frame. If playback is paused when the seek position is updated, the playback remains paused but showing the newly selected frame.

Codec Types and Random Access
Accurate seeking is ***only*** possible on clips that use intra-frame codecs (e.g. DNxHD, ProRes). It is possible to create software that performs frame accurate random access on Long-GOP codecs (e.g. MPEG-2, H264), but the process is highly complex, codec specific, and computationally expensive. CasparCG attempts random access with long-GOP codecs, selecting the first intra coded frame at or beyond the target frame number.

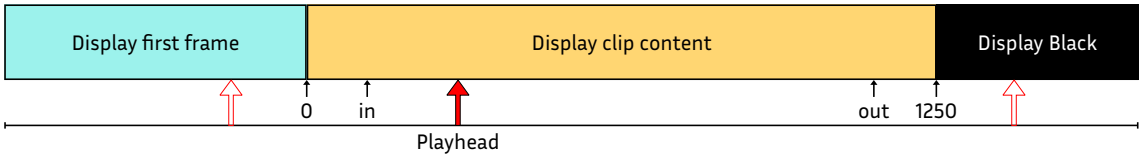
Assume a clip is playing in channel 1 layer 10. The playhead can be set to a new position, for example frame 153, using the command:

```
call 1-10 seek 153
```

The seek command can have either one or two elements. A single element provides an absolute seek, two elements allow offsets relative to a position described by element 1. The following table shows examples of seek commands and the response of the CasparCG server.

Command	Server Action
`call 1-10 seek in`	Move playhead to the clip in point (seek value in LOAD or PLAY)
`call 1-10 seek out`	Move playhead to the clip out point. Out point is the in point plus the set duration
`call 1-10 seek rel 20`	Move playhead 20 frames towards the start of the clip. The offset can also include a plus sign: `call 1-10 seek rel +20`
`call 1-10 seek rel -20`	Move playhead 20 frames towards the start of the clip.
`call 1-10 seek in +25`	Move 25 frames after the in point.
`call 1-10 seek end -1`	Move to the last frame of the clip. The value for end is the clip duration, but the frame numbers are zero based hence the offset of -1 in the command.

Note the playhead position is not limited by the duration of the clip. The illustration below shows the timeline and playhead for a clip that is 1250 frames long. Logical in and out points are set within the available content.



If the server is in pause mode and is instructed to place the playhead at frame -50, the video output is the first frame of the clip. When play mode resumes the output shows the first frame for 50 frame periods then starts normal playback. If the playhead is set after the last clip frame, in the above diagram any frame greater than 1250, the output content is black.

End Credits - Stacks:Rollers:Crawlers

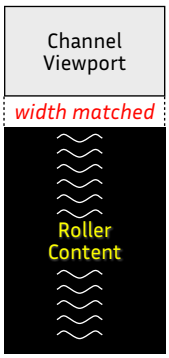
There are at least three methods to create and output end credits:

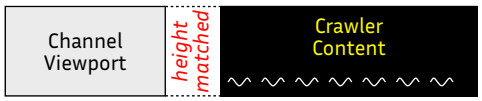
1. Use still pictures containing the text and associated alpha channel.

2. Use templated graphics with scripted control of colours, typeface and fount properties. Creation and instancing of this mechanism is beyond this introductory document.

3. Use a video edit or motion graphics package to create a video file with an associated alpha/key video.

The final credit is often a static full frame picture that contains the copyright owner and date plus production company information.





The image scroller creates roller captions when the image width matches the channel viewport width. For example 1280 pels wide for a 1280 x 720 HD format channel.

The image scroller creates crawler captions when the image height matches the channel viewport height.

CasparCG has two still image producers - Image and Image Scroller. The Image producer replays still pictures or static caption boards, and the Image Scroller producer creates roller and crawler caption effects using a still image source.

The core requirement for roller or crawler operation is that one still edge size matches the viewport width or height for the channel, as shown in the illustration at the left.

Static caption cards or roller and crawler slides can be edited using widely available still-picture editing software such as *Gimp* (free, open source), *Adobe Photoshop* (commercial licensed) or *Serif Affinity Photo* (commercial licensed). The credit still or stills should be saved in two formats - editor native and bitmap.

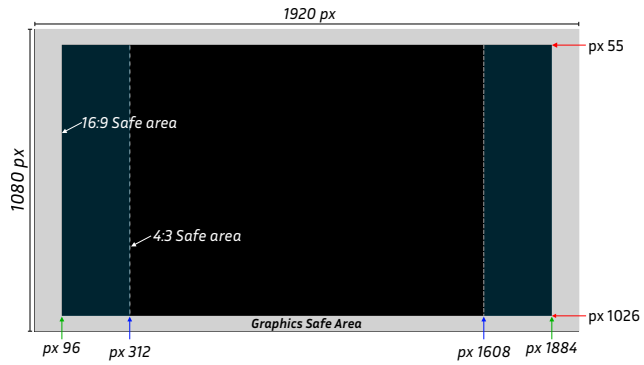
The editor native file enables subsequent revision of the text content. The bitmap file (e.g. TGA, TIFF or PNG) is replayed by CasparCG.

It is also possible to use presentation software such as Powerpoint (PC and macOS) or Keynote (macOS) as the editor, exporting the caption slides as bitmaps.

Any alpha channel is either embedded in the still, for example in a PNG file, or is saved as a second file. CasparCG auto recognises external alpha channel files using a simple naming convention which adds **_a** to the end of the filename stem - e.g. mypic01 (fill) and mypic01_a (alpha). It is usually better to use uncompressed or lossless compressed files for credits, with PNG providing small files using lossless data compression and an embedded alpha channel.

Credit Stacks

A typical programme needs several stills slides to show all the programme credits, hence the **credit stacks** label for this section. Credits should use safe area guidelines appropriate for the geographic area of transmission.



4 x 3 graphics safe zone from end of current programme

Upcoming Events

The following description uses the UK technical delivery standard specifications used by the major broadcasters. The positions of the 16:9 and 4:3 safe areas for high-definition productions are illustrated at the left.

Broadcast channel credit guidelines commonly state programme credits must be 4:3 safe to support programme sales to areas using 4:3 transmission chains.

These constraints also allow network playout presentation to use an end-credit squeeze that creates space for summary data about other programming shown at one side of the screen. Keeping the credit roller or credit stills inside the 4:3 safe zone ensures their visibility during the squeeze process. See the illustration at the left.

Planning overall caption layout is an essential starting point ahead of any data entry. Some text entry edit packages need a set of horizontal guide lines that are used as the baseline for text boxes. Other packages can use a single multi-line text box whose properties, such as fount size and line spacing, are well defined. Where different fount sizes are required for a job title and the persons name grid lines provide a better solution.

It is sensible to create one or more templates for the credit layout required, including the design for any end board that has copyright data and production company logos. Save the template for later use, either as a named template for the text editing program or as a file that is copied and renamed before new data entry occurs. Write protecting the master file that is copied to make new instances provides a degree of protection against “bad finger” operation.

Creating the SVT client rundown requires locating the programme credit slides in the CasparCG server media folder. Storing a set of credit stills in a named folder simplifies both search and archiving processes.

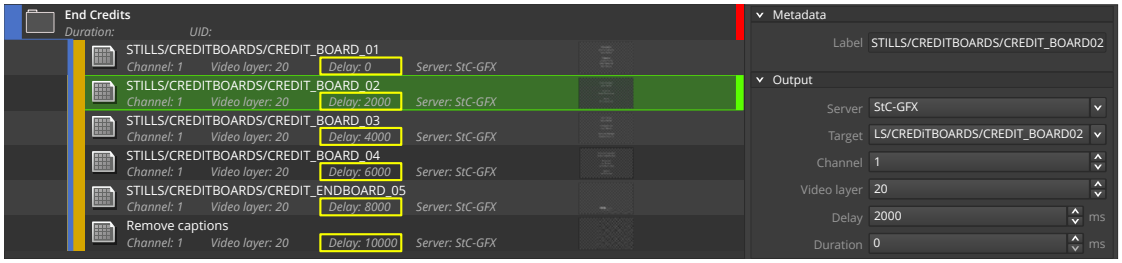
Do ***NOT*** use spaces in the folder and file names as this can sometimes cause problems for the file location process in the CasparCG. Use underscores or hyphens as word separators.

The CasparCG client library lists need updating once the credit slides are stored in the server media folder. Use either a mouse click in the client programme menu **Library** element or use the keyboard shortcut **Ctrl R** to update the lists of available media.

Use the library filter tools to locate the captions in the stills tab, loading the stills set into the rundown.

Full frame credits over a black background can be played out as stills into the vision mixer. If any of the credits will be shown over the studio output using the mixer DSK follow good operational practice by including a ***Clear Channel*** command that is played ***before*** the mixer DSK is enabled.

Replaying a sequence of end-credit stills can use manual playout of each still. Alternately the credits can be grouped together with user entered delays, relative to the first caption, for each caption in the group. A single play command addressed to the group sequences the credit stills playback. The group mode is illustrated below. The delay values in milliseconds, highlighted here in yellow edged rectangles, differ for each member of the group.



The above rundown example contains four credit stills plus the end board that has the production logo and copyright date. The final entry in the group uses the special file name **EMPTY** to cause a final blank output. Setting a mix transition mode for this final still fades out the end board.

Roller Captions

The roller content is created as a still image played by the image scroller tool. As stated previously, the width of the still image is important, and must match the full screen width of the replay channel, for example 1920 samples for full HD channel. The height of the image is set as large as needed to hold the full credit list.

When the image scroller starts playout the top of the roller starts just below the output viewport. The upward movement speed for simple rollers is set using the rundown item property inspector. The speed is set in lines per scan output as this minimises any potential motion judder. When a roller is played using a custom command the roll speed can use a floating point value, but minimum judder usually occurs when the speed is an integer value.

A blur filter can be applied to the still image content to further reduce judder visibility. The amount of blur is set using the property inspector.

Movement continues until the base of the image has moved above the top of the channel viewport

This is illustrated by the adjacent example roller caption. Preparing a roller still is, generally, relatively simple using a text layer in a bitmap editor.

A simple workflow starts by creating the list of credits as lines in a simple text editor, saving the text file in case further edits are required.

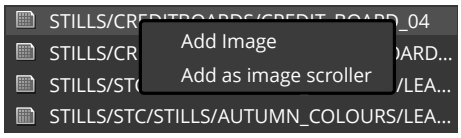
Using a full HD channel as an example, open the bitmap editor and create a new still 1920px wide by, say, 6000 px high. The final height is adjusted once all text is visible in the bitmap. Select the text tool, placing a text box starting near the top of the bitmap and extending to the bottom of the bitmap.

Set the typeface/fount(font), the text height, and select centred text entry. Copy and paste the text from the text editor into the text field of

the bitmap editor. Edit the position and format to create the desired results. A full size black background layer may help to see the captions as they are checked and positioned. The background can be retained for a full screen credit system, or it can be hidden before the final bitmap export.

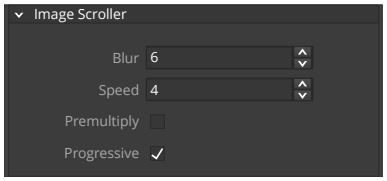
Finally edit the canvas size to leave just a small amount of canvas below the text list. Save the bitmap file in native file format.

Apply any filters, such as a drop shadow, required for the desired display then export the bitmap as a PNG image storing this file in the CasparCG server media folder for the programme.



Refresh the CasparCG client library.

Locate the roller still in the Stills tab. Select the still, then right click, selecting **Add as image scroller** from the pop-up menu.



Set the client output properties to the allocated channel and playout layer. The image scroller defaults to layer 10. Use a **channel clear** command before the credit replay commands in case residual text from a templated graphic is present on layer 20.

Use the Inspector panel to set the Image Scroller properties. Experiment to find the optimum **Speed** and **Blur** values. Adding some blur tends to make the roller less juddery, but it also softens the text. The blur action is applied to the movement direction axis.

Higher scroll speeds tend to need higher amounts of blur to avoid caption judder. Also test the effects of the Progressive flag on output quality. The inspector speed parameter can only use positive integer values.

Crawlers

Creating a crawler uses the same workflow as the roller, but the fixed size for a crawler is the ***height*** of the bitmap file - 1080 lines or pels for full HD. Adding the crawler to the rundown and adjusting the blur and speed parameters is identical to the process described above for a roller caption.

If the crawler text is to display above a graphic strap there are two simple mechanisms to set up the strap. Method one includes the strap in the bitmap passed into the scroller. Method 2 uses the graphics layer capabilities of the CasparCG, with the strap placed in a lower number layer than that used by the scroller bitmap with it's alpha channel. A group is created for the replay, with the opening and closing animations of the strap managed by standard CasparCG transition and delay controls.

Starting a Roller or Crawler

The SVT client enables the user preferences for the roller or crawler to be set using the item inspector. These values are passed to the server as a standard AMCP command. The syntax of the AMCP command for the scroller properties shown a few paragraphs above is:

```
play 1-10 “endcredit01” blur 6 speed 4 progressive
```

The SVT client issues the command when the user presses the F2 (play) key. The roll/crawl can be paused and restarted using the F4 key. Alternately the command line can be created using the custom command. The custom command allows a negative speed value to be included in the command:

```
play 1-10 “endcredit01” blur 6 speed -4
```

Positive values roll upwards or crawl from right to left. A negative speed value reverses the direction of the roll or crawl, hence a roll speed of **-4** starts with the caption above the channel viewport, rolling downwards to finish as the topmost part of the caption passes the bottom of the viewport.

There are two other command parameter formats that can only be sent using either a custom command or a custom client. Sometimes the end roller or crawler duration should match the run time for the closing music. The image scroll producer can compute the speed if the target duration is specified. For example:

```
play 1-10 “endcredit01” duration 20.0 blur 2
```

This resulting speed, in pels per field/frame, may be non-integer and potentially show some judder, but it will achieve the target duration specified in the command. The duration may be a negative value, which reverses the default motion direction.

A live programme may be required to end transmission at an exact time. Minor over or under runs in the timing of a segment can be challenging for graphics setting the speed or duration required to end the credit sequence at an exact clock time. There is an AMCP command format that supports the exact time end for the roller:

```
play 1-10 “endcredit01” end_time “2022-06-15 17:59:45” blur 2
```

When CasparCG server receives this command it starts the playout, computing the speed that finishes the roll or crawl at the specified date and time. The CasparCG host clock is used as the time reference for the end of the roll/crawl. If the end_time and the duration are both specified on the command line, the end_time is ignored.

There are two optional parameters that may be added to the image scroll command. These are boolean properties that can be set using a tick in the SVT client inspector, or added as a name to the AMCP command. The two properties are called **premultiply** and **progressive**.

Add premultiply if the image is sourced as straight alpha, setting this flag makes the caption display correctly in CasparCG.

When an interlaced video format is used the default processing moves the image each field. This behaviour can be overridden by specifying progressive in the command, causing movement only on full frames.

Dynamic Adjustment of Image Scroll Producer

The image scroller producer properties can be adjusted by using an AMCP **call** command. The implementation is similar to that used to adjust video clip properties.

The crawl or roll speed can be changed, either as a instantaneous value update or tweened over a user defined time period. Assume there is a roller caption running in channel 1 layer 10. The roll speed can be instantly changed by a custom command:

```
call 1-10 speed 3
```

Setting the speed to 0 pauses the roll or crawl. The movement is restarted by using a non zero speed value.

Instead of an instant speed change the command may include a tween duration and a tween control curve.

Assume the scroll playout was started and immediately stopped (using speed 0) before the text appeared on the channel output. The playout can be subsequently started where it accelerates to final run speed using a custom command of the form:

```
call 1-10 speed 6 150 easeinoutsine
```

The transition duration in the above command is 150 frames from current speed to speed 6 using a tween profile that accelerates gently and smoothly achieves final scroll speed.

Finish an image scroll with information still visible in viewport

The default image scroller action continues the scroll operation until the entire bitmap has been moved through the output viewport. There are instances when it is desirable to end the movement with some information still visible - for example the directors name credit.

Using the SVT client group mechanism a roller or crawler can be set to smoothly end whilst content is still displayed in the channel viewport. The first command in the group starts the image scroller movement. The second command in the group uses the producer call mechanism to ease the movement to a speed of 0 using either a linear or smoothed tween profile. This second command uses a custom command, placing the text of the call instruction in the F2 play entry field.

The first item of the group is manually started at the same instant as a stopwatch is started. The stopwatch is stopped when the final element of the roller or crawler is at the correct screen position. The time on the stopwatch is entered into the item Inspector, Output segment, Delay field.

Then select the group, and start the group operation with the F2 play key. Check the final position of the roll or crawl. Update the delay value as required to end movement at the correct position.

A third command can be added to the group so that the end-board with production company names and copyright date are shown after the scrolling data has stopped. This can either be cut onto a higher numbered layer, or transitioned onto the scroll layer using a mix or cut.

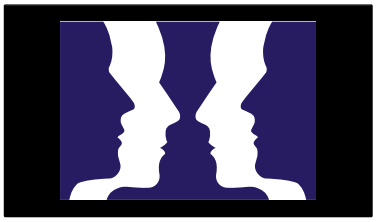
Static Opening Title

Not all programmes use video clips for their opening title sequence. Some use shots of the studio overlaid with text or other graphics as their programme title. Such titles can use either templated graphics or stills with alpha channels.

Using templated graphics provides easy editing of programme name, episode name or presenter name using the template item inspector panel. The slight complexity when using templated graphics method is the time and specialist knowledge required to create the HTML/Javascript code, although this may be mitigated by use of a visual design tool such as Google Web Designer or Loopic. Subsequent style changes also need specialist knowledge and preparation time.

Where the programme name is known a short time in advance of the studio session it can be simpler to use a still bitmap with alpha channel. The stills producer output passes via the channel mixer where size reduction and placement of the title can be adjusted using the CasparCG mixer transform tools.

If the title uses a graphic logo as well as name text, two stills and two layers provide highly flexible options for the final layout, including in-vision dynamic changes. Some examples using two layers are shown below.



Layer 30 - Full frame

Layer 40 - Full Frame + Alpha

Combined Output

Resize Example 1

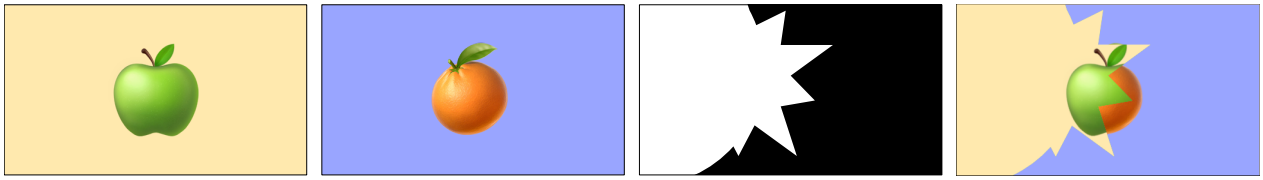
Resize Example 2

Output of Mixer Keyer

The logo is placed in layer 30 with the text name in layer 40. The layer combination places the text in front of the logo when the layers are combined as shown in the top row right hand side. Using mixer transform commands allows the two layers to be independently sized and placed creating the options shown by the first two examples on the lower row of pictures. The reduced size title elements can be layered above studio pictures as shown in the lower right picture.

STING transitions

A sting transition provides a dynamic change from the current source to a new source. The boundary between the two videos is determined by a monochrome mask video. Black areas of the mask show the current source, white areas show the new source, and mid-tones generate a linear mix of the current and new sources. The illustration below, using still images, shows the basic combine process of a sting transition.



Current Source | New Source | Mask Video | Result

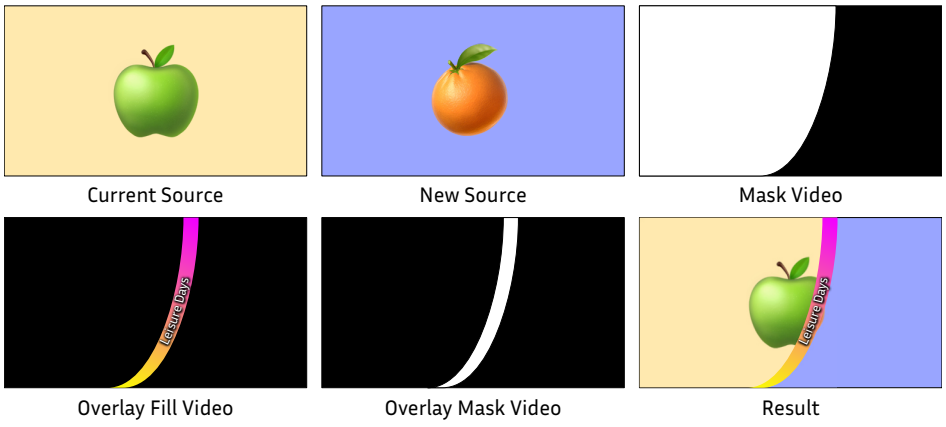
CasparCG uses the foreground/background capability of the layer producer. It requires a clip already playing in the foreground producer for the layer. The sting is loaded into the background using an AMCP command of the form:

```
loadbg 1-10 newSource sting maskVideo
```

Where newSource is the name of the video file that will start playing with the sting, and maskVideo is the monochrome mask video.

The sting transition starts when the server receives a `play 1-10` command.

The full version of the sting transition supports an overlay video playing at the same time as the masked transition. This is illustrated below.



Current Source | New Source | Mask Video

Overlay Fill Video | Overlay Mask Video | Result

Although the above illustration shows the overlay video strongly related to the masked transition, this is not a required feature. The overlay could be a firework that flies across the screen. The overlay and mask videos will normally be similar durations.

The full version of the sting transition AMCP command is of the form:

```
loadbg 1-10 <newSource> sting <maskVideo> <tp> <overlayVideo>
```

<newSource> is the name of the new video that starts playing in the sting.

<maskVideo> is the name of the monochrome mask video

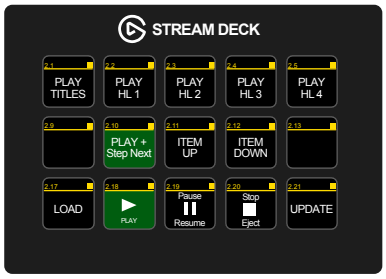
<tp> is the integer number of frames between when the sting starts and when the <newSource> video starts to play.

<overlayVideo> is the stem name of the video show at the uppermost layer of the sting. Two files are expected of the form myOverlay.mov (fill video) and myOverlay_a.mov (alpha video). Audio can be present as part of the overlay video.

The transition playout is initiated by a `play 1-10` command.

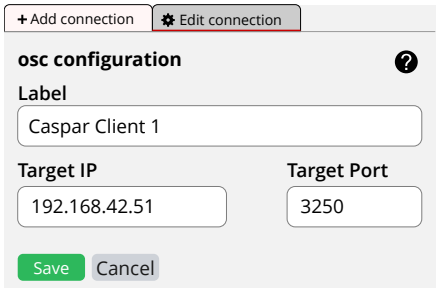
Elgato Stream Deck

Elgato manufacture items of hardware that offer control of streaming content for the web. Their hardware range includes microphones, lights, green screens, and a set of hardware button keypads known as Stream Deck.

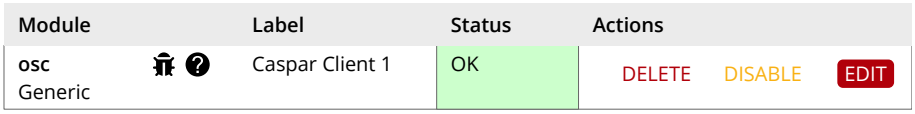


The Stream Deck hardware button panels are available with 32 buttons, 15 buttons and 6 buttons. Each button includes an LCD display enabling pictures, text or both to be shown in the button. The units connect to a host computer using a USB connection. The 15 button panel is illustrated at the left. The button units are cost effective and have been adapted for use in many control processes.

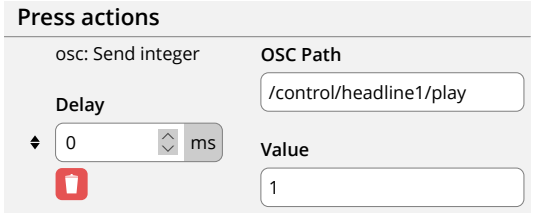
Combining the hardware panel with open-source BitFocus Companion software provides control of many items of audio-visual and broadcast equipment, including CasparCG. Companion can emit OSC messages triggering items in an SVT client rundown.



At launch, Companion runs in the background of its host computer. Stream Deck button actions are configured using the browser-based GUI. Each controlled device needs one or more connection definitions. The device type is selected from the list of supported units and the interface properties edited. The property edit form for a generic OSC connection is shown at the left. The label property is the name of the device displayed in the button editor. The network properties must also be specified, in the example the target port is the CasparCG version 2.2 Client default. When the property edit is complete and the settings saved the configuration is displayed in the connections list as illustrated below. The status shows if communications with the selected device are active.

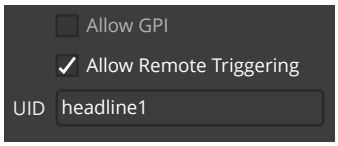


Companion software includes a Stream Deck emulator that supports 99 pages each with 32 buttons. Each of those 3168 buttons can be programmed to perform one or more actions when the button is either pressed or released. Multiple actions for press or release can be triggered simultaneously or sequentially using user specified delays. Actions include emitting a protocol specific message to an equipment instance such as a Blackmagic Design Hyperdeck disk recorder, or issuing a crosspoint change request to a crosspoint router that supports Probel SW-P-08 control protocol.



Configuring a button to send an OSC control command to trigger a specific item in an SVT client rundown is illustrated by the button press properties form on the left. The message must include an integer value of 1, used as part of the message validation in the client. The OSC path starts with the client reserved path element **/control/**. This is followed the URL text for the triggered item, **headline1**,

followed by the command action **/play**. Specific item triggers, such as the example here, work directly on the open rundowns in the client. The item with matching URL does **not** need to be part of the currently selected rundown.



The SVT client item trigger configuration entry is shown at the left. It is important that the UID text is unique in the open rundowns.

It is also possible to create a Stream Deck configuration that uses a set of simple UID names, for example 01, 02, 03 etc., and allocate the names to the items in the rundown to match operational requirements. This avoids any need to reconfigure the Companion buttons for each programme.

A rundown can be played out using the Stream Deck to move the selection cursor through the list, issue play or pause commands etc. This can reduce the desk space needed for the operator. In essence the Stream Deck plus companion become a remote keyboard for the client.

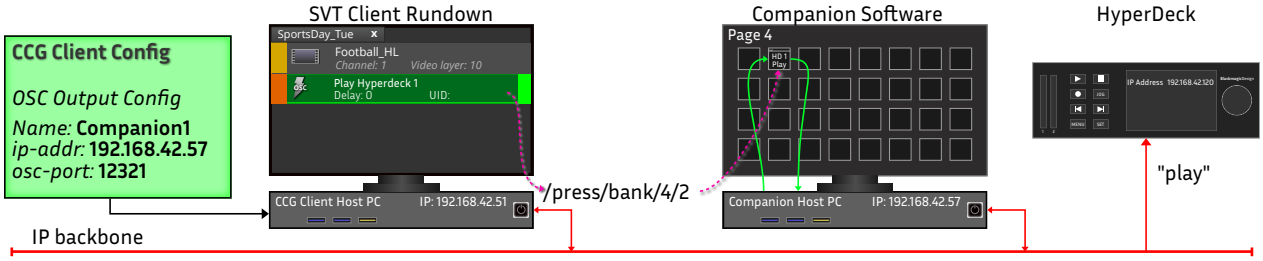
Configuring a Stream Deck button to act as a 'remote keyboard' uses the same configuration process, but the OSC path has just two elements such as **/control/play** or **/control/playandautostep**. These command only operate when the current rundown tab is remote control enabled (Rundown menu, Remote Control Enabled). A red lightening flash shows in the rundown name tab when remote control is enabled. The SVT client control verbs are listed in appendix B (here).

Companion includes a Stream Deck emulator that runs as a webpage and it can be used without Stream Deck hardware. Clicks on the buttons in the browser operate as if the hardware were present. This allows a simple tablet or mobile phone to operate as the button controller. The tablet or phone needs network access to the computer running the emulator.

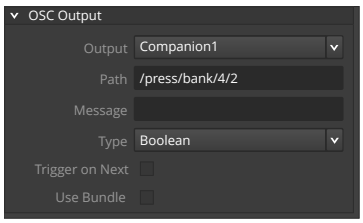
Companion has a further useful control mechanism. The buttons on a page can be actioned through one of several IP delivered message protocols. One of supported protocols is OSC on UDP/IP, enabling the SVT client to send messages that trigger a button on a page. The OSC address for this process is of the form:

```
/press/bank/<page>/<button>
```

Where the <page> and <button> are numbers. The SVT client has an OSC message output item, with a configuration element that holds the target IP address and port as a named destination. The OSC output item is inserted in a rundown using the Tools... Other... menu. The item inspector selects the named destination and enables entry of the OSC message detail.

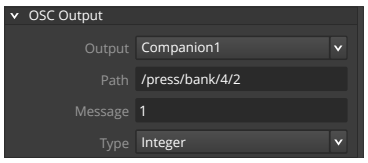


The above illustration shows the Client Host configured with an OSC output destination called Companion1 using address 192.168.42.57 and port 12321 (the Companion default OSC control port).

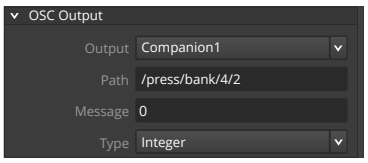


An OSC output item control is present in the current rundown. The properties inspector for the OSC output is shown at the left. The message output destination is selected from a drop down list. The list contains the OSC destinations defined in the client configuration. The Path field is the target OSC address. In this example the command is set to press button 2 on page 4. No parameters are required with the OSC command so the message box is left blank, and the parameter type is set as Boolean.

The Companion software has configured page 4 button 2 to send a ‘play’ command to a Blackmagic Design Hyperdeck at address 192.168.42.120 when the button is clicked. When the OSC item is played in the SVT client it causes the Companion button to be pressed, and Hyperdeck starts to play the current media.



Latch On



Latch Off

Some control actions need a stable state output, which may require one output control value when the button is pressed, and a different action when the button is released. The companion buttons can be latched or released by changing the message type to integer, sending a message value of 1 to latch the button and 0 to unlatch the button.

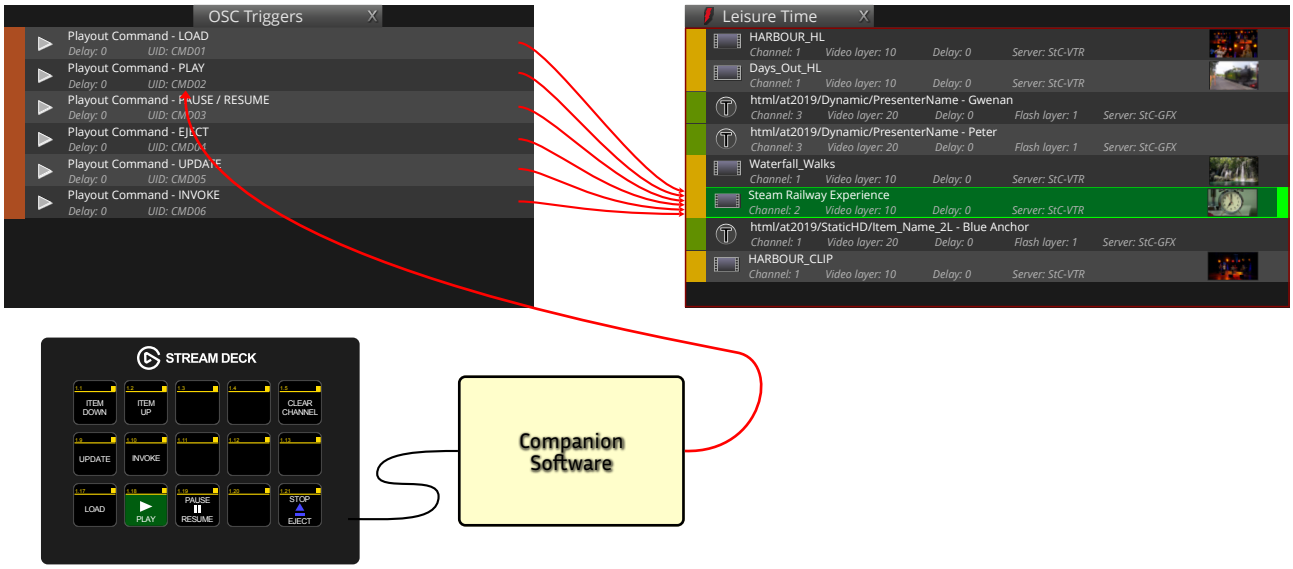
Client Playout Command

The Playout Command is one element in the Tools... Other... menu. The property inspector for this tool has just one parameter selected by a drop-down selector. The selections in the drop down match, with one exception, the commands available using the F1 to F12 function keys. The selections are listed in the table below.

Stop	Play	Play Now	Pause
Load	Next	Update	Invoke
Preview	Clear	Clear Video Layer	Clear Channel

The extra selection is Play Now which instructs the client to ignore any delay set in a item and immediately issue a play command. The above selections are also available using OSC control of a rundown.

The way the Client Playback operates is illustrated in the diagram below.



Two active rundown tabs are used. The above example has a simple media rundown shown at the upper right. This rundown is remote control enabled, shown by the red lighting symbol in the name tab. The rundown shown at the upper left contains only Playout Commands. The item properties for each playout command have been set by the item inspector. The name of the selected command has been added to the label text for ‘operator convenience’. Each playout command is remote control enabled, with the UID for each item shown in the summary properties of the item.

The UID triggers operate even when the OSC Triggers rundown is not the front tab in the rundown collection. The commands in the OSC Triggers rundown all link to the selected (active) item in the

current media rundown. This is indicated in the above diagram by the red lines joining the two rundown lists.

The Stream Deck button array links to the BitFocus Companion software. Companion is programmed to issue a **/control/CMD02/play** OSC message to the SVT client host when the PLAY button is pressed. Similar actions are set for the other media item buttons.

The top left pair of Stream Deck buttons issue standard remote control functions that move the item selection up or down the media list.

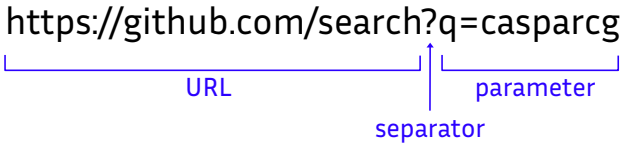
Overall, the client playout command delivers similar functionality to that achieved by using the OSC remote controls described in appendix B. Both can enable an operational control position that is remote from the client host PC. A scan converter can be used to provide the rundown list as a conventional video displayed on a monitor stack.

As at July 2022 there is a minor bug with the SVT client distribution which affects one playout command selection- the pause command does not work. The text in the drop down box should be **Pause / Restore** not **Pause**. The line of code to edit is documented in the client Github repository issue #301 (link), after which the code base must be rebuilt.

HTTP GET and HTTP POST - Control External Hardware

The SVT client includes tools to control various hardware devices including PTZ cameras, ATEM mixers, Tricaster mixer etc. When there is a requirement to control other hardware the user must provide some middleware that accepts an instruction from the client and sends the specific control protocol to the target hardware.

The SVT client has three tools that can send requests to the middleware - OSC output, HTTP GET and HTTP POST. The OSC output implementation is designed for simple message structures, not messages that require multiple parameter values. **GET** and **POST** support complex messages with multiple parameters. Search requests to a web site are often sent as a GET request. The GET request is part of the message header, but the text in the browser URL bar shows the parameters sent to support the query. Searching Github for references to CasparCG uses the URL shown below:



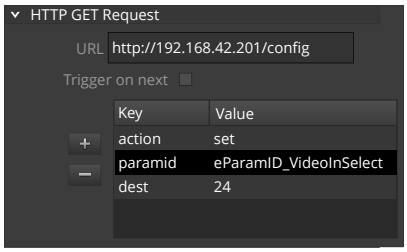
Equipment manufacturers increasingly offer a control interface that uses GET and/or POST. Although the GET command is a request to return data, it passes parameters to the destination and hence can be used to send control requests. POST messages are commonly used to send data from web forms to the server for storage, so this message can also be used to send control requests.

AJA manufacture a record/replay device called Ki Pro which has an http control api. The unit has several signal input connectors and the http control interface can select the required input source. Assume a Ki Pro has a network address of 192.168.42.201. The control access to that Ki Pro uses a URL of http://192.168.42.201/config. Setting the source input requires three parameters:

action=set
paramid=eParamID_VideoInSelect
value=0

Parameter 1 defines the control action as set rather than query current value. Parameter 2 is the name of the element to control. Parameter 3 is the input connector to select, in this example the SDI input. The full URL request sent to the Ki Pro is:

http://192.168.42.201/config?action=set&paramid=eParamID_VideoInSelect&value=0



The SVT client can send that request using the HTTP GET tool available in the Tools... Other... menu.

The address for the AJA is placed in the URL field, and the parameters entered as Key/Value pairs. When the GET request is played each key/value pair are combined into a string with an equals character joining the elements. The key/value strings are combined using the ampersand character.

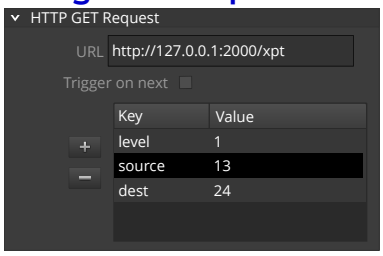
The Ki Pro replies to the selection request, but the SVT client is unable to display the outcome of the request as this would need extensive knowledge of the specific response protocol. The response from the server includes a standard status code, for example 200 for a successful transaction. This code number is recorded in the SVT client log.

The HTTP GET or HTTP POST mechanisms can be used to send control requests to equipment that does not have native support for these message types. The process requires middleware that receives the GET or POST message, translating the control message into the native protocol used by the device to be controlled. The remainder of this section describes how use user written middleware code to send a route request to a crosspoint router.

Three parameters are required to specify the crosspoint - the router level (video, audio etc.), the source number and the destination (dest) number. The sources and destinations are specified with 1 as the lowest number. If the specific router protocol uses 0 as the first source or destination the middleware converts the incoming numbers to the values required by the router.

The example uses node.js as the middleware system as this has extensive libraries for creating a web server and receiving and processing a GET or POST message. This example uses a number for the level, source 13 and destination 24. Assume the middleware is running on the same computer as the SVT client, using localhost as the network connection. The middleware listens for requests on port 2000.

Using a GET request



The HTTP GET request inspector with the example item values is shown at the left.

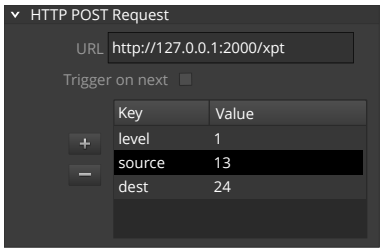
The basic code used by the node.js handler is shown below. This code illustrates how the GET request is linked to the active web service. The handler requests notification of get requests that are sent to port 2000 using a url route of **/xpt**. A more comprehensive handler could support multiple remote control protocols by using more url routes. The demonstration handler in the following code prints the request parameters on the node.js console.

```
// Node.js core code to demonstrate how to create a simple handler for
// GET requests from the SVT CasparCG client.
// The request handler uses the 'express' library to implement the server.

const express = require('express');   // Import the express library for use
const app = express();                // Make an instance of express called app
const myPort = 2000;                  // Listen for traffic on port 2000

// Register a callback for path "/xpt"
app.get('/xpt', (req, res) => {
   // XXXXX
   // Display the list of keys and values in the GET request
   for (var key in req.query) {
      console.log('Key_name=\"%s\"  key_value=\"%s\"', key, req.query[key]);
   }
   console.log('- - -');     // Simple visual separator on console display
   // XXXXX
```

```
   res.sendStatus(200); // Confirm the request arrived by returning "200: OK"
});

// Start listening for network traffic
app.listen(myPort, () => {
   console.log ('Listening on port %d', myPort);
});
```

Sending the crosspoint request to the physical router device requires the relevant protocol support code inserting between the two sets of `// XXXXX` in the above example. A real world system also needs parameter validation, checking all required values are present and in-range. If the validation test fails the returned status should indicate the failure so a later log check of the CasparCG client would show the error report.

Using a POST request



The properties inspector form for using the POST process requires the same entries as the GET process.

The code in the handler needs more elements as the request needs a parser. A suitable parser is available in the node libraries.

The same url route and port number used for the GET example are also used in this example.

```
// node.js core code that illustrates how to receive a remote control request
// sent using an http POST message from the SVT CasparCG client.
// Commonly used library functions are used for the web server (express) and
// message parser (body-parser) functions.
const express = require('express');
const bodyParser = require('body-parser');
const router = express.Router();
const app = express();
const myPort = 2000;

app.use(bodyParser.urlencoded({extended: false}));  // Attach the parser
app.use(bodyParser.json());      // Expect JSON encoded message

router.post('/xpt', (req, res) => {
   // XXXXX
   // Display the list of keys and values in the POST request
   for (var key in req.body) {
      console.log('Key_name=\"%s\"  key_value=\"%s\"', key, req.body[key]);
   }
   console.log('- - -');
   // XXXXX
   res.sendStatus(200); // Confirm the request arrived by returning "200: OK"
});

app.use('/', router);

app.listen(myPort, () => {
   console.log ('Listening on port %d', myPort);
})
```

The same set of validation tests apply for the GET and POST messages when deployed for operational use. Replace the code between the two sets of `// XXXXX` with the code that validates the request and sends out the request to the controlled equipment.

Logs and Log Housekeeping

CasparCG servers and clients each keep logs of their actions. There is no automatic purge of the logs inherent in either server or client. Monitoring and limiting the storage volume of log files requires a local management process. One possible solution is to use a windows scheduled application that deletes log files that are older than a user-selected age.

CasparCG Server Logs

CasparCG servers store log files in a folder whose location is specified in the server configuration file. Server configuration files are stored in the same folder as the casparcg.exe executable. The default configuration file is **casparcg.config**. It is possible to use a different configuration file name by passing the full name of the configuration file to the server using a command line parameter. For example to use configuration file **casparcg_SD.config** use command line:

```
casparcg.exe casparcg_SD.config
```

Using other configuration file names is not good practice for server version 2.2 and later, because the media scanner task finds the folders it must monitor by reading the paths stored in the casparcg.config file. Hence it is simpler to manage multiple configurations by using a batch file that copies the instance configuration from the instance name to *casparcg.config*, then starting the media scanner and server.

The configuration file uses XML to define many operational properties. There are two elements that are significant for the log process → **log-level** and **log-path**. The block below is a section from an example configuration file. The red coloured tag names shown here are the log file related items.

```
<log-level>info</log-level>
<paths>
  <media-path>M:/CasparCG/CCG/media/</media-path>
  <log-path>C:/CasparCG/CCG/log/</log-path>
  <data-path>M:/CasparCG/CCG/data/</data-path>
  <template-path>M:/CasparCG/CCG/template/</template-path>
  <thumbnail-path>M:/CasparCG/CCG/thumbnail/</thumbnail-path>
  <font-path>M:/CasparCG/CCG/font/</font-path>
</paths>
```

The log-level text controls the detail of reports stored in the log. The minimum amount of data is stored when the log-level is set as fatal. The supported values for log-level settings are shown in the table below.

log-level	Log content includes
fatal	Events that cause the server to have unrecoverable errors.
error	Events that are unable to successfully complete the request.
warning	Events that may have operational consequences. This may indicate a missing hardware feature, or an internal timing issue.
info	Lists all events. This level allows support teams to check that the server is receiving requests and includes the source tcp/ip address of the requests.
debug	Intended for use when developing the server software and wanting to check the sequence of sub-functions involved in a request.
trace	Very fine detail of each operation. Very large logs are created. This level is mostly used during server software development.

The log includes the chosen level plus the information from the higher levels in the above table. Thus a log-level setting of info sets the log process to also store messages from the warning, error, and fatal levels.

The log-level tag value defined in the configuration file sets the log-level when the server starts operation. The working level can be modified during server operation using an AMCP control message. The command is takes the form:

```
LOG LEVEL info
```

Log messages are displayed in the server's text window and stored in the daily log file. The log file location is set by the <log-path>...</log-path> token value of the configuration file. It is good practice to store the log on a different drive to the one that holds the media. The log file name uses the format **caspar_yyyy-mm-dd.log** where yyyy is the year, mm the month, and dd the day of the month.

The panel below shows an example extracted from a log file with the log-level set at **info**.

```
[2021-02-19 13:59:04.314] [11724] [info]    Received message from 127.0.0.1: LOAD 1-10 "MAE_BRADBURY/HARBOUR_CLIP" CUT 1 Linear RIGHT\r\n
[2021-02-19 13:59:04.322] [11552] [info]    ffmpeg[Harbour_Clip.mov|720x576p25.00|0/1512] [video-decoder] DV (Digital Video)
[2021-02-19 13:59:04.323] [11552] [info]    ffmpeg[Harbour_Clip.mov|720x576p25.00|0/1512] [audio-decoder] PCM signed 16-bit little-endian
[2021-02-19 13:59:04.324] [11552] [info]    ffmpeg[Harbour_Clip.mov|720x576p25.00|0/1512] [audio-decoder] PCM signed 16-bit little-endian
[2021-02-19 13:59:04.330] [11552] [info]    transition[empty=>ffmpeg[Harbour_Clip.mov|720x576p25.00|0/1512]] Initialized
[2021-02-19 13:59:04.331] [9188]  [info]    Sent message to 127.0.0.1:202 LOAD OK\r\n
[2021-02-19 13:59:04.335] [7716]  [info]    [frame_muxer] deinterlace_bob 720x576i50.00
[2021-02-19 13:59:05.170] [11724] [info]    Received message from 127.0.0.1: LOAD 2-10 "LOADTEST01" CUT 1 Linear RIGHT\r\n
[2021-02-19 13:59:05.193] [10600] [info]    ffmpeg[Loadtest01.mxf|1920x1080p25.00|0/3000] [video-decoder] VC3/DNxHD
[2021-02-19 13:59:05.197] [10600] [info]    transition[empty=>ffmpeg[Loadtest01.mxf|1920x1080p25.00|0/3000]] Initialized
[2021-02-19 13:59:05.197] [9188]  [info]    Sent message to 127.0.0.1:202 LOAD OK\r\n
[2021-02-19 13:59:05.211] [8876]  [info]    [frame_muxer] simple 1920x1080i50.00
[2021-02-19 13:59:17.706] [11724] [info]    Received message from 127.0.0.1: PLAY 1-10\r\n
[2021-02-19 13:59:17.706] [9188]  [info]    Sent message to 127.0.0.1:202 PLAY OK\r\n
[2021-02-19 13:59:21.730] [11724] [info]    Received message from 127.0.0.1: PAUSE 1-10\r\n
[2021-02-19 13:59:21.731] [9188]  [info]    Sent message to 127.0.0.1:202 PAUSE OK\r\n
[2021-02-19 13:59:26.202] [11724] [info]    Received message from 127.0.0.1: PLAY 2-10\r\n
[2021-02-19 13:59:26.202] [9188]  [info]    Sent message to 127.0.0.1:202 PLAY OK\r\n
```

Each line of the log includes four elements - the date and time of the log entry, the thread_id that generated the entry, the category level of the entry, and the log message.

In the example log above, the first line shows a message was received from tcp/ip address 127.0.0.1 (localhost) and that message requested a video clip called "MAE_BRADBURY/HARBOUR_CLIP" be loaded for playout in channel 1 layer 10. The \r\n at the end of the line indicates the command ended with the required terminating sequence of carriage return (\r) and line feed (\n). The next three line of entries were issued by the ffmpeg producer reporting properties of the clip. The fifth line was issued by the mixer sub system stating it is ready to transition from an empty layer to the video content. The sixth line is the message returned to the client, in this instance a success report.

SVT Client Logs

The SVT client, like the CasparCG server, keeps logs of actions and received messages. The client needs configuration data such as the tcp/ip address of CasparCG servers and other controlled broadcast kit such as switchers and cameras. The client configuration data is stored in an SQLite database.

The database and the client log files are stored in a CasparCG specific folder called **.CasparCG**. This folder is a branch of the users home folder - typically ***C:\Users\User_Name***. The configuration is stored in **.CasparCG\Client\Database.s3db** and the logs are in **.CasparCG\Client\Logs**

The log file name uses of the format **Client_yyyy-mm-dd.log** where yyyy is the year, mm the month and dd the day of the month. An extract from a log is shown in the box below.

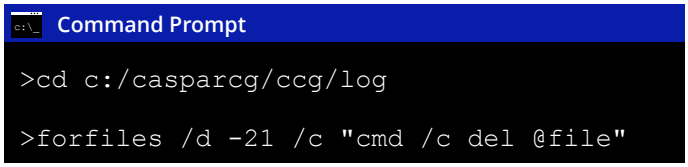
```
[2021-02-25 16:50:08.053] [3288] [D] Starting CasparCG Client 2.0.8.012c50b49a
[2021-02-25 16:50:08.124] [3288] [D] Using SQLite database
[2021-02-25 16:50:08.768] [3288] [D] Listening for incoming OSC messages over UDP on port 6250
[2021-02-25 16:50:08.808] [3288] [D] Sent message to 127.0.0.1:5250: VERSION SERVER\r\n
[2021-02-25 16:50:08.809] [3288] [D] Sent message to 127.0.0.1:5250: INFO\r\n
[2021-02-25 16:50:08.813] [3288] [D] Received message from 127.0.0.1:5250: 201 VERSION OK\r\n
[2021-02-25 16:50:08.830] [3288] [D] Received message from 127.0.0.1:5250: 200 INFO OK\r\n
[2021-02-25 16:51:11.873] [3288] [D] Sent message to 127.0.0.1:5250: PLAY 1-10 "STILLS/STA/MISC/BLETCHLEY_LAKE_01" CUT 1 Linear RIGHT\r\n
[2021-02-25 16:51:11.898] [3288] [D] Received message from 127.0.0.1:5250: 202 PLAY OK\r\n
```

Each line of the log includes four elements - the data and time, the thread id, the category level, and the log message text. The category levels are [D] for debug, [W] for warnings, [C] for critical, and [F] for fatal.

Deleting Log Files

Deleting old log files can be done manually on every server and client computer. View the Windows screen output of the host computer. Use the Windows file manager to navigate to the relevant log folder. Use the list display tool to show the log files in date order. Select the unwanted older files, fully deleting them from the system (do ***not*** leave the deleted files in the recycle bin).

The Windows operating system includes tools that can speed the deletion of older files, both manually and through a time scheduled process. The **forfiles** tool can be run in a DOS command window. This tool uses various command line switches to act on files in the currently selected directory. For example to delete files that are 21 days or more older, open a DOS window and enter commands as shown in the box below, selecting the relevant directory in the first command line.



The first line changes the working directory to the folder holding the CasparCG server log files. In this example the log file location is the one defined in the example configuration file. The second line uses the **forfiles** command to look for files that are more than 21 days old then runs a delete operation on each such file.

It is also possible to combine the path to scan into a single command line as shown below:

This command can be saved in a batch file.

Windows includes a task scheduler that can run an event daily, weekly, or monthly. The description below should work for both Windows 7 and Windows 10 host computers to set up a once per week purge of old logs.

Use the taskbar search tool to find and run **taskschd.msc**. In the right-hand side pane select the Create Basic Task... Wizard and run it. Enter a task name, for example ccglogpurge, and a description for the task, then click on the Next> button. Set the trigger to Weekly, then press Next>. Choose a day of the week and time when the task is to run, and set the Recur to every 1 week. Do **not** use Sunday 0100 to 0300 to avoid issues when daylight savings start and stop. Press Next>. Set the Action to Start a program, then press Next>. Fill in the Program/Script and arguments boxes with the data based on the entries shown in bold below (the path setting is from the example configuration).

Multiple entries can be created for each set of log files that need management. If the host computer is not continuously powered it is also possible to run the purge process described above whenever the computer is powered.

CasparCG Server Configuration

Each CasparCG server requires customising to implement the end-user requirement. Server deployments may use:

- A single-screen consumer output
- One or more network-based outputs (e.g., NDI)
- Multiple SDI outputs
- A hybrid combination of output types.

Each physical and virtual channel of a server needs initial spatial and temporal resolution values. The spatial and temporal properties can be adjusted whilst the server is running using AMCP commands.

When the CasparCG server software starts, it reads a configuration file stored in the same folder as the CasparCG server executable code. The default configuration file is called **casparcg.config** but multiple configuration files can exist, each having the **.config** file extension. On startup, CasparCG server software checks the command line supplied. If the command line has no parameters the server uses the default configuration file, otherwise it assumes the first parameter is the name of the configuration file to use.

Note the media scanner system used in server versions from 2.2.0 also needs to access the configuration file. Use a batch file to copy the desired configuration file, naming the copy as *casparcg.config*.

CAUTION: An incorrect configuration file can stop server operation. Before starting any configuration edit ensure there is an up-to-date safety copy of the current configuration file. If the edited configuration then proves unworkable, the file safety copy provides a mechanism to restore functionality.

Loading Server Software

New versions of the server or client software are released from time-to-time. These may provide bug-fixes or add new features that are operationally desirable. The installation process is relatively simple, requiring manual installation of couple of software dependancies if the host computer has not previously been used for CasparCG.

The software dependancies are:

1. Microsoft C++ run-time support library
2. Microsoft .NET framework V4.0 or later (normally already present on a Windows 7 or Windows 10 Operating System).
3. Driver software for any Blackmagic Design Decklink or Bluefish444 SDI cards
4. NDI iVGA output software for CasparCG servers prior to version 2.3. Obviously this software is only needed if NDI outputs are required. Version 2.3 has built in NDI version 4 producers and consumers, but does require an NDI runtime library. This library can be installed as part of the free NDI tools.

Downloading any needed Dependancies

The Windows versions of the server are built using Microsoft Visual Studio. The NRK branch uses Visual Studio 2015 and the SVT branch uses Visual Studio 2017. There is a common C++ runtime available for VS2015, VS2017 and VS2019. When this document was written the common runtime library (vc_redist.x64.exe) was available from Microsoft [here]. Before downloading and installing the support library check if already exists on the computer, as another task may have loaded it.

To check if the runtime library software is already loaded, run the Windows Settings tool, select Apps... from the displayed set of options, then select Apps & Features from the left-side menu. Scroll the list to the Microsoft section and examine the files. Look for *Microsoft Visual C++ 2015-2019 Redistributable(x64)*.

The Blackmagic Design Decklink drivers and other utilities are part of a package called Desktop Video which is available from the Blackmagic Design website support pages [here]. Generally the latest version of desktop video is the one to use. When this package is installed it adds the Decklink configuration tool, the Media Express capture tool, and a disk speed check tool to the host computer applications. Again check the computer to see if a recent version is already present. Look for Blackmagic Desktop Video. If present, a single click shows the product version.

Downloading of the iVGA driver needed for NDI output in CasparCG version 2.1.xx_NRK involves a lot of searching as many links in search engines are broken. CasparCG version 2.3_LTS branch (or later) has built-in support for NDI inputs and outputs. An NDI run-time component is required. If this is missing when the server starts, you are prompted to fetch and install the component. The component can also be installed as part of the free NewTek NDI tools. These tools are useful when checking NDI operations on CasparCG.

IMPORTANT: If downloads are required, follow local policies that ensure downloaded files are free of unwanted content such as viruses, spyware, trojans and adware.

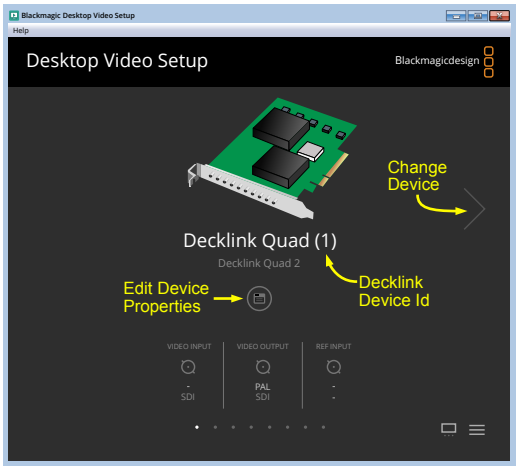
Each dependancy has a simple wizard to guide users through the install process. The Decklink driver system will ask for permission to re-boot the computer. When the re-boot occurs there may be several instances of a Blackmagic Design pop-up asking to search for the latest drivers. Allow the installation using the *search the host machine for drivers* option.

The Decklink install process may also show pop-up menus if it finds the board firmware is not current, and it wants permission to run the firmware updater.

Once the drivers are available the Decklink cards require several property values setting, especially allocation of SDI outputs on a multi-output card.

Example of Blackmagic Design Desktop Video Set Up

The configuration software is installed as part of the Desktop video installation. It can be started using a shortcut in the Windows Start Menu. At task startup it shows a screen of the style shown below, without the yellow arrows and text.

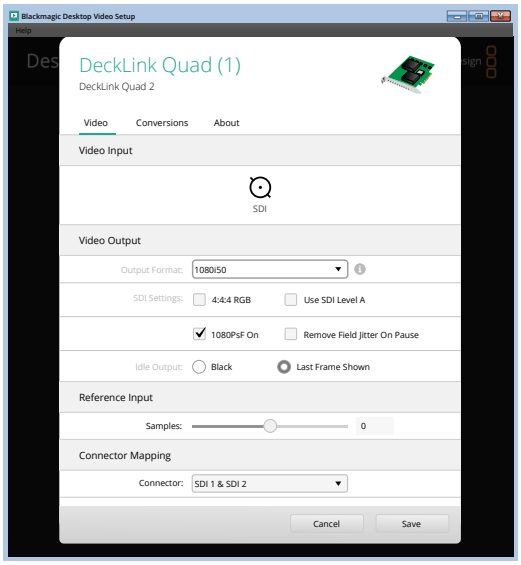


The picture of the card shown at the upper centre of the application window changes according to the type of card being configured.

The Device ID (shown in brackets, see yellow highlighter text) is significant for CasparCG Operations. It is used as part of the mapping of a CasparCG logical channel to a physical output.

The series of eight white dots at the bottom of the software window correspond to the Device ID. The pictograms above the dots show the video settings for the selected channel. In this example the output is an SDI signal, the video output format is set to 576i25 ("PAL"), and the REF Input shows the selected reference. This display element is only available when a reference is connected to the Quad 2 card.

Click the Edit Device Properties button to display the properties form. This will be similar to the picture shown in the following page.



For this illustrative example assume the first output channel has both a fill and key signal, requiring two SDI connectors.

Set the Connector Mapping (Drop down selector near bottom of the edit screen) to "SDI 1 & SDI 2".

Set the Output Format to required value - e.g., 1080i50.

Do ***not*** activate the Remove Field Jitter on Pause

Set Idle Output to "Last Frame Shown".

Set the reference to 0 samples offset.

Save the config. This may require confirming to Windows that the software may change the currently saved settings.

Repeat the config exercise for the other Decklink IDs. The major differences are in the Connector Mapping. Devices 2, 3, 4, 6, 7, 8 should be set to select just a single SDI connector.

Device 5 should be set as Connector NONE because the nominal connector was grabbed for use by channel 1 key output.

After a device edit, review the settings for all 8 devices to check no accidental changes to a connector mapping that has affected a partner device.

Once all Decklink configuration is complete, reboot the PC. Whilst this may not always be required, experience shows it reduces problems with subsequent CasparCG Server configuration.

Installing CasparCG Server Software

If the CasparCG server software is an update to an existing installation take copies of all **.config** files in the existing CasparCG folder. These will be restored after the updated server software is installed.

The software for both the NRK and SVT LTS branches is available from Github. Most search engines return a valid link using a search string of either **github casparcg server nrk** or **github casparcg server lts**. When this document was created the NRK release was available here, and the SVT LTS version was available here. Click on the **Releases** button in the browser page to ensure the latest release is at the top of the page.

Create or locate a suitable folder on the target host computer to hold the unzipped file content, for example C:\casparcg\unzips. Copy the zip archive to a temporary folder on the target computer, then select the zip file in the Windows file browser. There may be packaging structure differences between the various server distributions. If there are folders in the archive, search for the one called **server** and open that folder.

Click on the **Extract all** icon of the file browser. A pop-up window appears with a target folder for the extract. Browse to the chosen extraction folder and click OK to extract the archive.

Create the target folder that will hold the running copy of the files - for example a folder that contains some identification of the server version such as C:\CasparCG\Server_2d3d3_LTS. Select all files in the unzip folder and copy to the target folder.

If this installation is replacing an existing server version copy the .config files saved earlier in the install process into the server folder.

Delete all desktop shortcuts to CasparCG server and create a new shortcut to the batch file called **casparcg_auto_restart.bat**. Whilst the NRK server can be run directly using the exe file, the LTS branch has

a separate media folder scanner task that must be run before the server code starts. The batch file starts the scanner and server as required by the release version.

Editing the Server Configuration File

The configuration file defines the storage paths to the media and template files, the spatial resolution properties of output channels and the control access ports. The default configuration file is called casparcg.config and it is available in the same folder as the CasparCG executable task. The file uses XML tags to define the properties, and it can be edited using any text editor.

The bare minimum configuration would contain:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
     <paths></paths>
     <channels></channels>
     <controllers></controllers>
</configuration>
```

<paths>

The paths section of the configuration defines the paths to the resources the server needs to access. The paths may be set relative to the folder where the server executable is stored, or absolute including the drive letter and the paths. Take care entering paths, as the common Windows path separator, \, is an escape character in XML encoded files. If the media is stored on a drive known to Windows as **M:** the two methods to enter the content are:

```
M:\\CasparCG\\ccg\\media\\

M:/CasparCG/ccg/media/
```

NOTE: All path entries *MUST* include the slash at the end of the path name. Omitting this will cause errors when the server runs. The two main code branches have a different number of tags.

The path configuration for the SVT 2.3.x_LTS branch takes the form:

```
<paths>
     <media-path>m:/CasparCG/ccg/media/</media-path>
     <log-path>c:/casparcg/server/log/</log-path>
     <data-path>m:/CasparCG/ccg/data/</data-path>
     <template-path>m:/CasparCG/ccg/template/</template-path>
</paths>
```

The <media-path> entry is the root folder for the video and stills media. Sub-folders are allowed in the media folder, and these can simplify media management and client library searches.

The <log-path> often points to a different drive to the other entries. It defines the location where the log reports from the server are stored. The log detail is defined by another tag in the configuration file.

The <data-path> folder is where user-defined data sets are stored. These data sets are often written by an external computer process, and a template is directed to read the content from the named data set.

The <template-path> is where the template files and their resources are stored. Templates may share the media drive or use a different drive, leaving maximum data bandwidth for the media transfers.

The path configuration in the NRK code branch takes the form:

```
<paths>
     <media-path>m:/CasparCG/ccg/media/</media-path>
     <log-path>c:/casparcg/server/log/</log-path>
     <data-path>m:/CasparCG/ccg/data/</data-path>
     <template-path>m:/CasparCG/ccg/template/</template-path>
     <thumbnail-path>m:/CasparCG/ccg/thumbnail/</thumbnail-path>
     <font-path>m:/CasparCG/ccg/font/</font-path>
</paths>
```

The <thumbnail-path> folder is where the media scanner process of the server stores the thumbnail pictures of the video media and still images. The thumbnail pictures are sent to the client when requested. In the SVT code branch the media scanner process is a separate process used by the server code but that also provides an http interface for other software accesses. The thumbnail images are stored in a POUCH database folder within the server folder.

The <font-path> is the folder where fonts required by the text producer are stored. Note these fonts are **not** used by templates which use fonts from the host computer font folder or an external URL. This tag is not required or used by the SVT LTS server code branch as it does not include a text producer. If the tag is present in the configuration file the information is ignored by the SVT server code.

<channels>

This section of the configuration defines the properties of all the server physical and virtual output channels. Each channel is identified by a <channel></channel> tag pair. The order of entries in the configuration defines the channel numbers. For example:

```
<channels>
     <channel>
          <!-- Properties of CasparCG channel 1 -->
     </channel>
     <channel>
          <!-- Properties of CasparCG channel 2 -->
     </channel>
     <channel>
          <!-- Properties of CasparCG channel 3 -->
     </channel>
</channels>
```

A channel that uses a Decklink card for output will have a channel properties section of the form:

```
<channel>
     <video-mode>1080i5000</video-mode>
     <consumers>
          <decklink>
               <device>1</device>
               <embedded-audio>true</embedded-audio>
               <latency>normal</latency>
               <keyer>external</keyer>
               <key-only>false</key-only>
               <buffer-depth>3</buffer-depth>
          </decklink>
     </consumers>
</channel>
```

The <device> tag value is the Decklink card number that will output the channel. The Decklink configuration software links the number used in this tag with the SDI connector on the card edge.

The <embedded-audio> tag is set true when the channel is to output audio. Enabling embedded audio also enables other audio outputs such as analogue or AES on Decklink cards that support separate audio outputs. Pure graphics channels can set this tag to false, reducing the latency through the output by 1 frame.

The <latency> tag controls the latency through the Decklink card processing. The settings of normal or default add 1 extra frame of latency relative to low, but this default setting tends to be more stable (lower chance of a dropped frame).

The <keyer> tag tells the server how to manage the keyer that is onboard some Decklink models. If the card, such as an Extreme model, supports internal keying set this tag to external. If the configuration is to use a separate output channel for the key use a value of external_separate. To use the card to key content onto the signal applied to the SDI input connector set the value to internal. The <keyer> tag can be omitted if the channel is used for fill video only.

The <key-only> tag is set true to output only the key signal. This mode is activated when a channel is set to output the key generated by another channel.

The <buffer-depth> sets the buffer size in frames used in the path to the card. Modern 4k cards may be stable with a depth of 2, but most cards require a buffer depth of 3.

<controllers>

The entries in the controllers tag set ports and protocols used to control CasparCG. There are usually entries for AMCP control and OSC status and control.

```
<tcp>
     <port>5250</port>
     <protocol>AMCP</protocol>
</tcp>
```

The above example listens on tcp/ip port 5250 for AMCP commands sent from any source tcp/ip address.

Miscellaneous tags

There are several tags that control aspects of the server operation.

Channel Locking

A CasparCG client can set exclusive control of a selected channel of a server using the AMCP **LOCK** command. When locking the server the client must provide a user selected password for both lock and unlock operations. For example locking channel 1 can be achieved by:

```
lock 1 acquire brutus
```

where the password is brutus. Unlocking the channel uses the command:

```
lock 1 release brutus
```

How can the channel be unlocked if the user password is not known? One solution is restarting the server - best described as a rather drastic mechanism! The second method is to use the password defined in the <lock-clear-phrase> tag. For example:

```
<lock-clear-phrase>verysecret</lock-clear-phrase>
```

The server channel is then unlocked by a message:

```
lock 1 release verysecret
```

Log Control

The amount of information stored in the server log can be adjusted by the AMCP **LOG LEVEL** and **LOG CATEGORY** commands. The initial values for the logging process are set by <log-level> and <log-categories> tags:

```
<log-level>info</log-level>

<log-category>communication</log-category>
```

The log-level values are shown in the following table. Data is logged from the selected level plus those shown lower in the table. Thus a log-level of warning includes warnings, errors and fatal events.

Log-level	Logged Information
trace	Very detailed reports showing route taken through the software. Intended for deep debug operations during code development. Results in very large log files.
debug	Reports various diagnostics and debug properties, including dropped frames and other timing matters such as buffer allocation time usage.
info	Lists all commands and responses with clients, and shows the server startup process including the content of the selected configuration file. This label is the default and is set if there is not a valid <log-level> tag in the config.
warning	Actions or requests that result in a warning being issued.
error	Actions or commands that report errors are logged.
fatal	Only catastrophic errors are logged.

Each log entry includes a thread id and the date and time at which the log event occurred.

The log category has three options:

```
communication
calltrace
calltrace,communication
```

The default is to log the communication messages. If the communication category is set off, the log shows the messages that arrive from a client, but does not show the server response.

When calltrace is enabled it writes to file calltrace.log in the log folder enabling trace log capture even though the main log has disabled the trace level.

OSC control
CasparCG server can emit status data for the active outputs and layers using UDP packets carrying OSC data messages. This information is used in the SVT client to show the video playout progress bar. The `<osc>` tag defines the ports and addresses to be used.

```
<osc>
     <default-port>6250</default-port>
     <disable-send-to-amcp-clients>false</disable-send-to-amcp-clients>
     <predefined-clients>
          <predefined-client>
               <address>127.0.0.1</address>
               <port>5253</port>
          </predefined-client>
     </predefined-clients>
</osc>
```

The `<default-port>` tag defines the udp port number on which the status data is emitted. Each AMCP client that connects to the server receives the osc data unless the message send is disabled by the `<disable-send-to-amcp-clients>` tag.

Some CasparCG installations, for example automation playout systems such as Sofie, require knowledge of the progress of clip outputs. Such installations can receive broadcast OSC messages by defining the addresses and ports in the `<predefined-clients>` entries.

Media-Scanner
The SVT V2.3_LTS code branch uses a media scanner that runs as a stand-alone task. That task is used by the main server but it also allows other access processes to request information about the available media via http get requests. The ip address and port used for access is defined in the `<amcp>` tag set.

```
<amcp>
     <media-server>
          <host>localhost</host>
          <port>8000</port>
     </media-server>
</amcp>
```

Thumbnail Images

The media scanning process creates thumbnail pictures for video clips and still images. The thumbnail generation process differs significantly between server version 2.3.X_LTS and server version 2.1.X_NRK. The 2.3.x server branch splits the server operation from the media scanning and thumbnail generation. The 2.1 NRK branch media scanner and thumbnail generator are part of the main server code. Thumbnail images are stored in a designated folder, and sent to the client on request from the client.

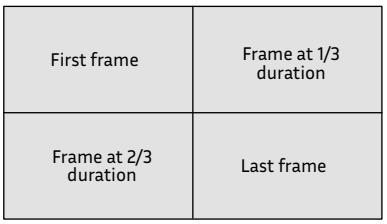
Server 2.3.X_LTS

The entries in the casparcg.config file for the media scanner access are shown in the previous paragraph in the `<amcp>..</amcp>` tags. The thumbnail image video is taken from the first frame of the video.

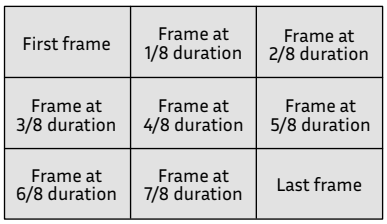
Server 2.1.X_NRK

The thumbnail images are stored in a user defined folder whose address is included in the `<paths>` tag block. The thumbnail image configuration defines several properties for the thumbnail images. A still image thumbnail is a reduced size version of the still, but the thumbnail for a video clip can be created from multiple time points through the video.

```
<thumbnails>
     <generate-thumbnails>true</generate-thumbnails>
     <width>256</width>
     <height>144</height>
     <video-grid>2</video-grid>
     <scan-interval-millis>5000</scan-interval-millis>
     <generate-delay-millis>2000</generate-delay-millis>
     <video-mode>1080i5000</video-mode>
     <mipmap>true</mipmap>
</thumbnails>
```



<video-grid> value = 2



<video-grid> value = 3

The width and height of the thumbnail image are defined in the `<width>` and `<height>` tags. The above example tag set has the width and height set for 16:9 media. The `<video-grid>` tag defines the sub-division of the thumbnail area for video clip operations in the NRK branch. The structure of the resulting thumbnail for values 2 and 3 is illustrated at the left. A value of 2 generally gives a good balance between an ability to resolve the picture information and understanding the video content progress through the clip.

If the video-grid tag value is changed, existing thumbnails are not regenerated. Manually delete the content of the thumbnail folder to force regeneration. The media store is scanned every few seconds, the interval between scans is set by the `<scan-interval-millis>` tag. Each scan checks for new, modified or removed media and updates the thumbnails as appropriate.

When the <mipmap> tag is true the thumbnail generation process may have reduced aliasing.

HTML Templates

HTML templates are rendered by the embedded Chromium engine. This engine supports remote debug access from the tools built into the Chrome browser. The `<html>` group tag has a `<remote-debugging-port>` tag that sets the tcp/ip port number used. If this is set to 0 (the default value) the debug action is disabled.

```
<html>
     <remote-debugging-port>9001</remote-debugging-port>
     <enable-gpu>false</enable-gpu>
</html>
```

To use the debug access start Chrome browser on the server computer setting the URL as **chrome://inspect/#devices**

On the displayed page, tick the ***Discover network targets***, and click the ***Configure*** button. Add the localhost address with the port number defined in the <html> tag - **127.0.0.1:9001** and click the Done button. After a few seconds there should be a list of templates running in CasparCG server.

It is also possible to enable the use of the graphics processing unit, gpu, to speed rendering. When gpu support is enabled there may be some issues caused by the dual use of the gpu which is needed for the mixer effects. As at April 2021 there is also a bug report for a continued growth of memory use (i.e., a memory leak) when gpu acceleration is enabled. This bug is proving elusive to locate!

Flash Support
CasparCG initially used Adobe Flash as the technology for templated graphics rendering. During 2017 Adobe announced that it was stopping support for Flash at the end of 2020. In January 2021 Windows rolled out an update that permanently removes Flash from a standard Windows install. There is a mechanism to continue to use Flash for CasparCG, useful for some organisations that have hundreds of templates that use Flash and ActionScript 3. The default for new CasparCG server deploys is to disable Flash support in the configuration file. If Flash is needed consult the CasparCGForum for instructions of how to install the modules and set registry keys. The edit the server configuration file to set the `<flash> <enabled>` tag to true.

```
<flash>
     <enabled>false</enabled>
     <buffer-depth>auto</buffer-depth>
</flash>
```

Audio Configuration
The SVT 2.3.X_LTS branch defaults to 8-channel audio passthrough, whereas the version 2.1.X_NRK server branch defaults to stereo audio operations. When more audio channels than the default are required the audio configuration must be set to enable the extra channels and to offer conversion between various modes. The mode to be used is set by a tag in the `<decklink>` section of the `<consumers>` segment of the configuration file.

The available channel layout options and the process of converting from one audio config to another are defined in the `<audio>` tag group. A sub-set of options is shown below:

```
<audio>
  <channel-layouts>
    <channel-layout name="mono" type="mono" num-channels="1" channel-order="FC" />
    <channel-layout name="stereo" type="stereo" num-channels="2" channel-order="FL FR" />
    <channel-layout name="matrix" type="matrix" num-channels="2" channel-order="ML MR" />
    <channel-layout name="8ch" type="8ch" num-channels="8" />
    <channel-layout name="16ch" type="16ch" num-channels="16" />
  </channel-layouts>

  <mix-configs>
    <mix-config from-type="mono" to-types="stereo, 5.1" mix="FL = FC | FR = FC" />
    <mix-config from-type="mono" to-types="5.1+downmix" mix="FL = FC | FR = FC  DL = FC |
                                        DR = FC" />
    <mix-config from-type="mono" to-types="matrix" mix="ML = FC | MR = FC" />
    <mix-config from-type="stereo" to-types="mono" mix="FC &lt; FL + FR" />
  </mix-configs>
</audio>
```

Setting a Decklink channel output is implemented by adding two entries to the channel definition, one at the channel global level and one at the decklink consumer level. The configuration segment below shows how a channel and decklink card are set to operate in 8-channel passthrough mode.

```
<channel>
    <video-mode>1080i5000</video-mode>
    <channel-layout>8ch<channel-layout>
    <consumers>
      <decklink>
      ...
         <embedded-audio>true</embedded-audio>
         <channel-layout>8ch<channel-layout>
      ...
    </consumers>
</channel>
```

Supported tag values

When a new CasparCG server is installed the default configuration file has two blocks, the actual configuration data, and an XML comment block that shows the recognised tags and supported values. An XML comment block is present between **<!--** and **-->** markers.

Each tag set shows the tag name, followed by the default value then the list of supported values. An example from the Decklink consumer is the keyer mode:

```
<keyer>external [external|external_separate_device|internal|default]</keyer>
```

This states that the default keyer mode is external, and this value is set if the tag is not found in the configuration data entries. The recognised values are listed between the square brackets, with each option separated from a neighbour by the vertical line (pipe) character.

Do **not** just copy the options line and paste into the configuration area without editing the copy. Remove unwanted values and remove all whitespace. For example `<keyer>internal </keyer>` would ***not*** be recognised as the tag value is “internal “ and the recognised string value is “internal”

Configuration Validator Tool

Sometimes a new or updated configuration does not operate as expected. Diagnosing the error can take time and lots of testing. Standard fault-finding mechanisms can be used - a software version of the “rule of halving” to locate the problem zone can speed debug operations. Make extensive use of the XML comment markers to disable blocks if the configuration. Comments can not be nested so if the configuration includes lots explanatory comments, make a new version of the configuration without the comments in place and use that comment free version for the debug operation.

There is an online configuration checker tool available at https://casparcg.net/validator/ which may also provide useful pointers to issues. The checker tool can be downloaded and run locally.

Config Backup - Important!

It is very important to keep a well-named safety copy of the edited and proven configuration. This can be used either to restore operations after a major failure, or as the copy source when multiple active configurations are required.

CasparCG SVT Client Configuration

The SVT client has many properties controlled by configuration. The client uses an SQLite database to store client configuration data, some operational option lists such as easing modes, and copies of media thumbnail pictures. On a Windows host operating system the default folder for the database is stored in the active users workspace. With a user login name of ***myName*** the client logs and database are saved in folder **`C:\Users\myName\.CasparCG\Client`**

The default database file name is ***Database.s3db***. If required, this database can be opened and the content inspected or modified using the free ***DB Browser for SQLite*** application.

At launch, the client tries to open the database. If the file does not exist a new database is created containing default settings.

The type and content of fields in the database changes with client version. This is known to cause issues when a version 2.0.9 and a version 2.2.0 are used on the same host computer if they use the *same database file*. Client version 2.0.9 is used with server version 2.1.X_NRK, and client 2.2.0 is used with server version 2.3.X_LTS. Different clients are required because:

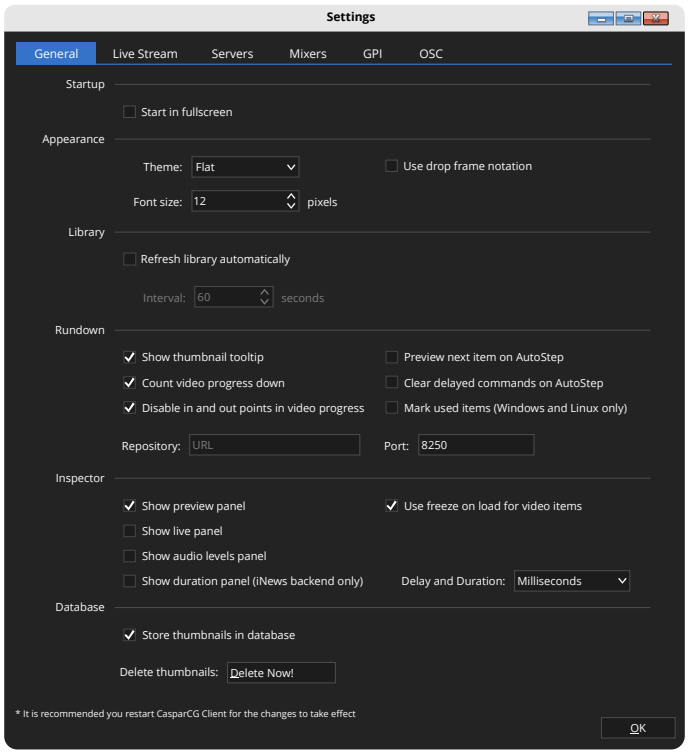
- OSC status data emitted by the servers differs in format,
- ffmpeg parameters used for live streaming of a preview are different because the ffmpeg code base has been updated for server version 2.3.

If the playout progress meter is missing on the rundown displays this may be because a client has modified the database. This error can only be resolved by deleting the database file, launching the client, and doing a full configuration.

The database file name to use can be provided as a command line parameter. The identifying argument is **`-t`** and is followed by the full path and name for the database file. An example command line is:

```
“C:/CasparCG/ClientV2d0d9/casparCG Client.exe” -t c:/casparclient/usedb209.s3db
```

The client configuration screens are accessed from the client application Edit menu, Settings... item. The General configuration window panel is shown below.



The Startup option allows the client to open in full-screen mode, probably the most desirable display mode in operational use.

The Appearance options define the look and feel of the client interface. The available Theme options are Flat and Curve. The latter option gives a rounded appearance to buttons and menu bars. Drop frame notation is relevant to non-integer frame rates such as 29.97 Hz.

The Library option allows the client to regularly request the state of the media libraries from connected servers. The time intervals between the requests are user adjustable when the auto-refresh mode is enabled.

Auto refresh operation is very helpful when a CasparCG server is operated in a news application where media is updating both ahead of and during the programme transmission.

The Rundown section of the General settings panel has several options that affect the rundown display. It is generally easier for users to see how much media is left in a clip, and this display mode is enabled when the Count video progress down option is active. The Mark used items option dims the rundown data for items that have been played. This can assist users to find the current playout position in a long rundown display. The rundown display brightness is reset when needed via a Rundown menu item Mark Item sub-menu.

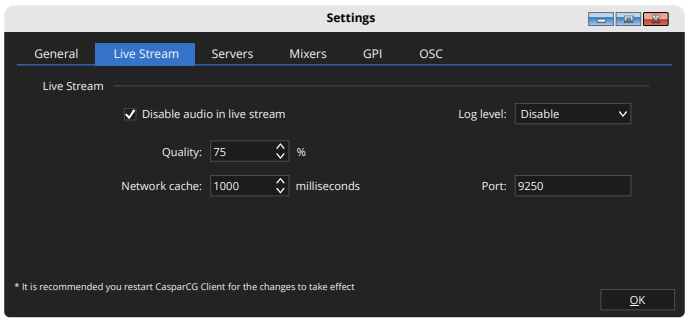
The repository URL and port are used for integration with a news production system. The endpoint is an html server system. When the client opens the repository a communication chain is created that supports adding and removing stories via a unique story id.

The Inspector section of the window controls the client Inspector panel appearance. The Preview panel is shown at the top left of the client library display section. When this panel is active, user library selections display the thumbnail still in the preview box.

The live video and audio level sub-panels are displayed at the top right of the client window area when enabled. The panels allow video fill or alpha from any channel to be seen in the client along with audio level bar graphs. Creating this preview requires spare processor capacity in the server which has to H264 encode the video content. Despite significant effort by the development team there is an as yet unresolved (April 2021) issue in the Windows server version 2.1.X NRK branch. The server crashes when the live panel values are changed.

The Use freeze on load setting for video items is a very useful operational feature, enabling a director to see the first frame of an upcoming video.

The Database section of the general settings enables thumbnail pictures to be stored in the local database, and provides a mechanism to flush the thumbnails from the database.

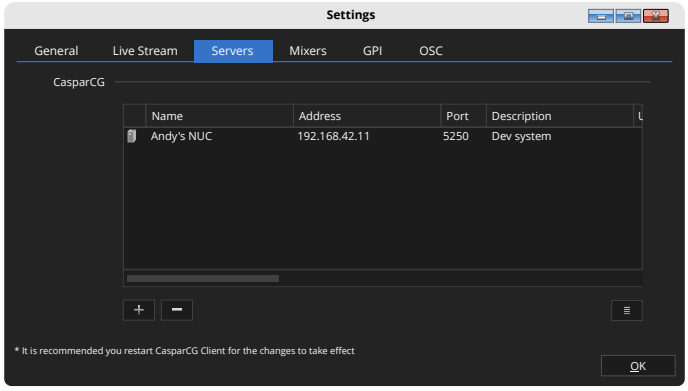


The Live Stream settings panel is illustrated at the left. Unused blank space has been removed to reduce the height of this image.

The settings in the segment are only significant when the live panel and audio panel are enabled in the general settings.

The Log Level selector controls logging for issues in the received live stream.

The Servers configuration section of the panel is illustrated below. The height of the server list section has been minimised to reduce the overall height of this graphic.



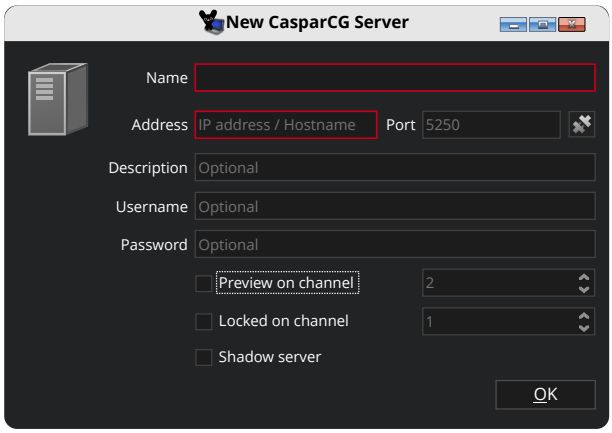
The function of this entry area is informing the client how to access the servers that it will control.

Each entry in the list includes a logical name, the TCP/IP address and port number for the server. There is also a text field that describes the server. This allows a short cryptic entry in the name field and a more reader friendly ident in the description field. For example a server name of ***StC-GFX*** could have a description ***Studio C Graphics Outputs***.

When media items are added to the rundown, or commands inserted into the rundown, it is the logical server name that is shown in the drop-down server list in the item inspector panel. The logical name is also included in saved rundowns. If a saved rundown is opened in a different client that uses a different logical name for the same TCP/IP address all the entries in the rundown have to be edited to enable

control of the intended server. It is better to use a common set of names for every server available at the premises, as this ensures ease of copying rundowns between servers.

To edit an existing server entry double-click the line in the list of servers. To add or remove a sever use the **+** and **-** buttons below box with the current list of servers.



When the add server button is pressed a pop-up form displays requesting server details.

Enter a logical name for the server in the Name box. On networks that use DHCP service for all computers enter the server host name into the Address field, and the Port address in the port field. On other networks enter the dotted server IP address in the Address field. The name, address and port must be supplied, the other fields are optional.

The Username and Password fields enable the client to use Windows verification when accessing the server.

The Icon to the right of the Port number entry field runs a connection test. Pressing the icon starts an attempt to connect to the server, reporting a successful connection if it can establish the route. If no connection establishes, the client continues a background poll for the connection, reporting success if the route subsequently connects.

The F8 function key allows a user to show a preview of an item on a CasparCG channel if an output is available. The Preview on channel tick box enables the preview facility for the selected server. The server channel that provides the preview still is specified in the box at the right of the preview label.

The Locked on channel tick box constrains the client to control only the specified channel number. When an item is added to the rundown it auto-selects the locked channel number.

The shadow server tick box stops media lists from the shadow server showing in client library lists. CasparCG assumes the installation team provided an automated media copy facility to maintain the same content on the primary and shadow servers. A github project available at https://github.com/hreinnbeck/casparcg-multicast-amcp uses multicasting to send each AMCP command to multiple client machines.

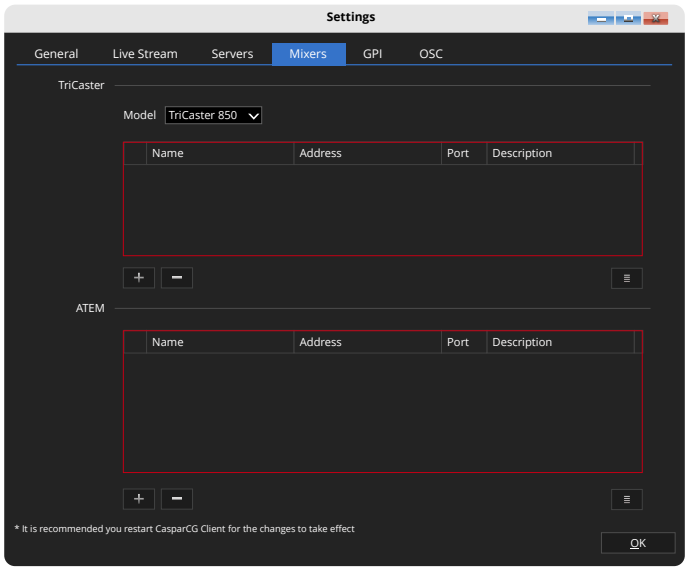
The CasparCG client can import server properties from a file containing details of many servers. A short distance below the right hand end of the bottom of the server list is a small icon with four horizontal bars. When this is clicked it allows the user to navigate to a file that holds the server list, then select servers from that list that are added to the list of controllable units. Using such a server list helps ensure the same names are used throughout a multi-server multi-client installation.

The server list is an XML document using the structure illustrated below.

```
<?xml version="1.0" encoding="UTF-8"?>
<servers>
   <server>
      <name>demoimport</name>
      <address>192.168.43.21</address>
      <port>5250</port>
      <username>myUserName</username>
      <password>myPassword</password>
      <description>Studio A Stills</description>
      <shadow>no</shadow>
      <previewchannel>0</previewchannel>
      <lockedchannel>0</lockedchannel>
   </server>
   <server>
```

```
      ...
   </server>
</servers>
```

Each XML server tag entry has nine sub-tags that correspond to fields in the manual New Server form.



The Mixers settings panel enables entry of the connection properties for Tricaster and ATEM mixers.

Existing entries are opened for edit using a double-click on the entry. New devices are added by clicking on the + button. The client displays an entry form for the properties.

The TriCaster form has fields for the name, the address, the TCP/IP port number and the description.

The ATEM form has fields for the name, address and description fields. The description is shown in the list starting in the Port field.

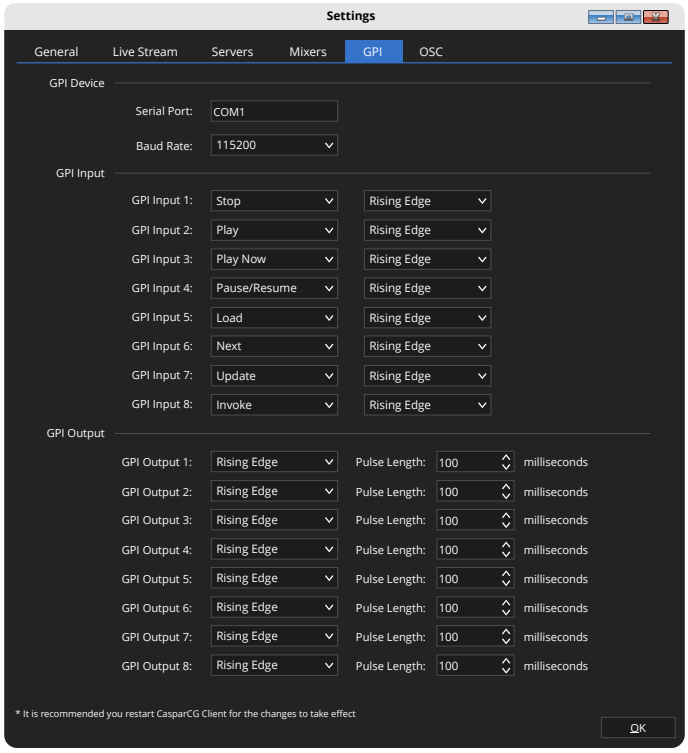
Both of the entry forms have communications test buttons.

Both types of mixer can import configuration settings from an XML file. The structure of the TriCaster XML configurations file is:

```
<?xml version="1.0" encoding="UTF-8"?>
<mixers>
   <mixer>
      <name>Tricaster V16</name>
      <address>192.168.52.11</address>
      <port>5950</port>
      <description>V16 Experimental system</description>
   </mixer>
   <mixer>
      ...
   </mixer>
</mixers>
```

The structure of the ATEM mixer configurations XML file is:

```
<?xml version="1.0" encoding="UTF-8"?>
<mixers>
   <mixer>
      <name>AtemMini 1</name>
      <address>192.168.50.11</address>
      <description>1U Studio unit</description>
   </mixer>
   <mixer>
      ...
   </mixer>
</mixers>
```



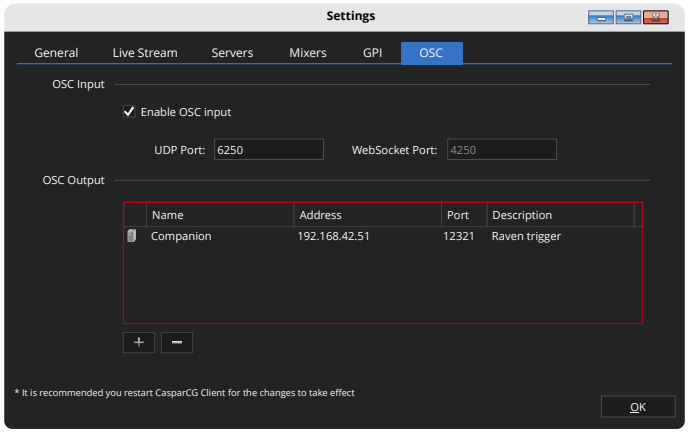
The GPI tab provides the configuration settings for GPI input and output connections.

The GPI Device is the serial communications communications port used for the GPI service. Two Baud rates are supported - 9600 and 115200.

GPI input properties set the active edge for each input, and the allocation of the action when the trigger condition occurs.

GPI output properties set the output state for each bit through the transition. A rising edge will be normally low, pulsing high for the duration defined in the associated pulse duration entry.

The OSC tab enables or disables OSC control of CasparCG, and defines targets for OSC messages that can be emitted by rundown events. The settings details differ slightly between client version 2.0.9 and 2.2.0



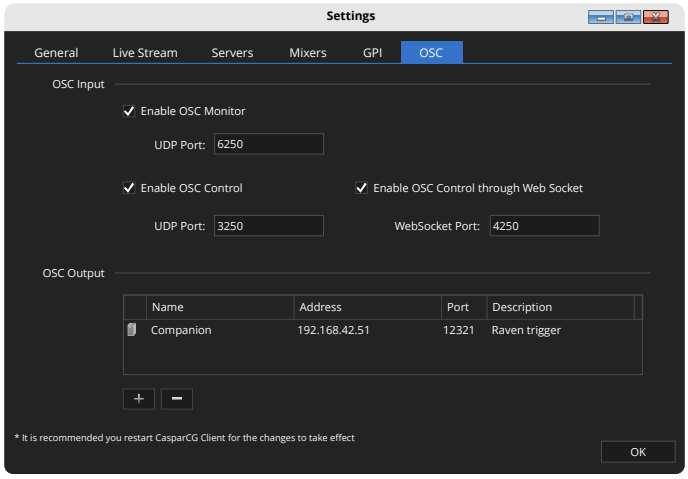
The example at the left is from the client 2.0.9 configuration.

When the Enable OSC input box is ticked the SVT client listens for OSC messages on it's own IP address on both the UDP and WebSocket ports selected by the numeric entry boxes.

Enabling OSC control enables both item-trigger and rundown-item select and trigger control.

The OSC output properties define one or more named OSC targets. The entries in the Name column are available in the drop down destination selector of the OSC Output tool in a rundown. The message sent to the destination is entered into data boxes in the OSC output element of the rundown via the element inspector.

The Osc properties form for client version 2.2.0 is shown to the left. The major change is that there are separate enable and port number boxes for the OSC monitoring and control connections.



When OSC monitor is enabled the client listens for the progress information reported by clip playout.

When OSC control is enabled the client accepts both item and rundown control triggers.

The OSC Output segment defines the names of devices or computers that can accept OSC messages from the client OSC output widget.

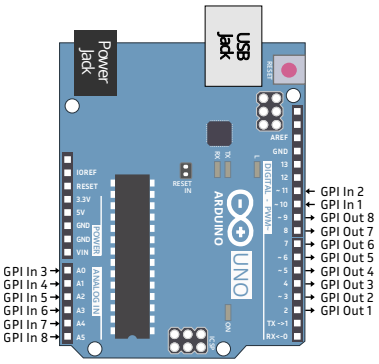
Appendix A - GPI Interfacing

There are some programme events where a user needs to trigger multiple pieces of equipment, for example to change some graphics and to change some lighting settings.

The general purpose interface (GPI) provides a simple mechanism to effect external triggers without the need to use complex communications protocols. The controlled equipment is user-configured to run a macro or trigger an internal event when the trigger state is received.

General-purpose computers require specialist interface cards to directly support GPI interfacing. Adding a card is possible in some desktop or server style computers, but not with a laptop. Serial RS232 interfaces are available as low-cost cards for desktop computers, and as USB adaptors for laptops or desktops. SVT adopted a serial data link using a simple message protocol to implement CasparCG client GPI functionality. They also created an Arduino UNO based microcontroller to implement the serial to GPI physical interface. The Arduino sketch code is available from a read-only Github archive at:

https://github.com/CasparCG/Tools/tree/master/cpp/gpio/trunk/arduino_gpio controller



The allocation of the Arduino interface pins to the GPI functions is illustrated by the adjacent drawing.

Any engineer building the Arduino interface is expected to use relevant protection circuitry between the controller and the "outside world". This includes using opto-isolation on the inputs and miniature relays for the output connections.

The USB connection between the Arduino and the host computer provides the bit-serial connection used by the message protocol. Use the Windows Device Manager to find the com port number allocated to the connection, and enter the port number in the CasparCG client GPI configuration.

Modern equipment often provides an open API allowing external control via a TCP/IP connection. It is relatively simple to create a translation gateway that communicates with the CasparCG client via serial protocol, translating the assigned GPI bits into API messages sent via the TCP/IP connection. The translation gateway programming can use any language that supports both the serial messages and the TCP or UDP connection to the equipment.

GPI Serial Protocol

The message exchange uses 4-byte blocks using ASCII characters to encode the message information:

```
<byte 1> <byte 2> <cr> <lf>
```

Where `<cr>` is the carriage return character and `<lf>` is the line feed character. In software coding the carriage return is commonly written as `'\r'` and the line feed is written as `'\n'`.

Client to GPI Interface

There are four messages sent from the CasparCG client to the GPI interface:

Byte 1	Byte 2	Byte 3	Byte 4	Message content
'i'	'?'	<cr>	<lf>	Return the number of GPI inputs
'o'	'?'	<cr>	<lf>	Return the number of GPI outputs
'a'	'?'	<cr>	<lf>	Are you alive?
'0'...'7'	'0' | '1'	<cr>	<lf>	Set output port number in byte 1 to the state in byte 2

GPI Interface to Client

There are four messages sent from the GPI interface to the client:

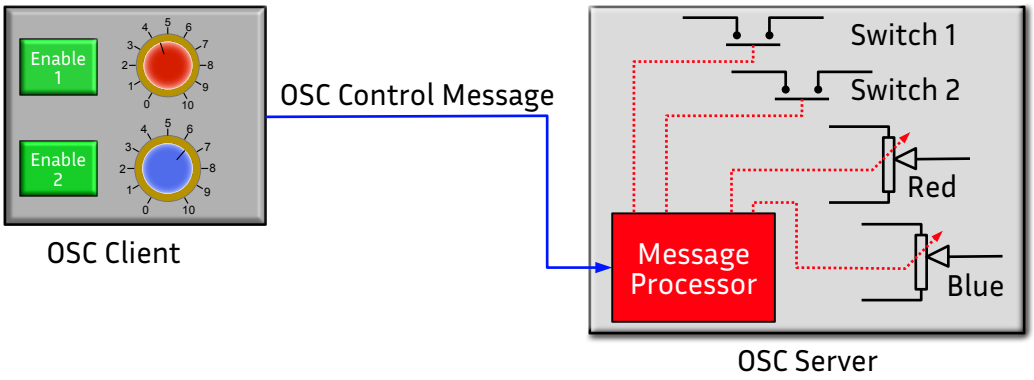
Byte 1	Byte 2	Byte 3	Byte 4	Message Content
'i'	'0' + Num_GPI_In	<cr>	<lf>	Response to Query from client. Returns the number of GPI inputs.
'o'	'0' + Num_GPI_Out	<cr>	<lf>	Response to Query from client. Return the number of GPI outputs
'a'	'!'	<cr>	<lf>	Response to "Are you alive?" query.
'0' + port_no	'0' | '1'	<cr>	<lf>	Report a GPI input value.

Appendix B - Open Sound Control

Open Sound Control (OSC) was created to pass control messages between units of audio equipment. Version 1.0 was released in March 2002, with a subsequent minor update to version 1.1 in 2009. Version 1.1 supports more data tag types, thereby supporting a wider set of information formats. The flexibility of the OSC message package enables the transport of control data for non-audio applications, meaning OSC has developed more into Open System Control.

Full detail of version 1.0 is available via this link, and details of the 2009 version 1.1 changes are available via this link.

Control messages are sent from an **OSC client** to an **OSC server** that actions the control request.



In this example system, the content of the OSC control message identifies the property to adjust and the new value for the property.

The OSC server implementer decides if the switch element control is encoded as a boolean value, or as an integer value. Similarly, the implementor can choose if gain settings controlled by the rotary client controls are encoded as an integer or as a floating point value. The implementor can also choose to support multiple representations of each controlled parameter. The OSC client implementer or system installer require documentation of the address and data structure supported by the server.

OSC messages are transport system agnostic. Commonly used connection methods include bit-serial (EIA RS232, EIA RS422 etc), USB, UDP/IP and TCP/IP. No pre-defined IP ports are allocated for OSC control functions, end users select port numbers that are available in their control network.

OSC use in CasparCG

CasparCG uses OSC over UDP transport. OSC is used in Caspar CG for 5 functions:

1. Sending status information from a CasparCG server to connected clients. The OSC data is used to control the progress bars of active media clips.

2. Sending status information from a CasparCG server to designated OSC servers. This mechanism is used to create status displays showing media names and playout progress etc.

3. Sending event triggers into the CasparCG client from an external OSC client.

4. Sending event triggers from a web page using Web Sockets to trigger events in a CasparCG client.

5. Remote control of the rundown operations in a CasparCG client. This includes selecting the active item, and issuing commands such as load, play and stop.

OSC Message Structure

OSC messages are wrapped into a ***packet***. Each OSC packet contains an **address** that identifies the controlled element, a **type tag** that describes the encoding mechanisms of the control value or values, and an **information** field containing the updated property values.

Sequences of ASCII characters in an OSC message are known as an **OSC string**. The end of the character sequence is identified by a null character ('\0'). The OSC string length must be a multiple of 32-bits, implemented by appending extra null characters when required.

The control address is a variable-length URL-style name, encoded in an OSC string. An example address is **/control/titles/play**. There is a small list of reserved characters that may ***not*** be used as part of an address:

(Space character)' ' **# * , / ? [ ] { }**

The type tag is an OSC string that identifies the data type or types transported in the packet. This OSC string starts with a comma and is followed by a series of letters identifying the encoding of the parameter values that follow. Commonly used parameters are 32-bit integer values, 32-bit floating point values, OSC strings, OSC blobs (binary data), midi data (4-bytes), logical True and logical False.

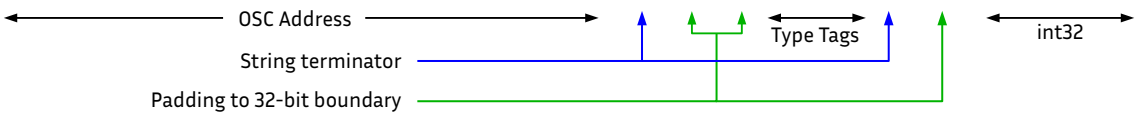
The OSC information field has the same number of values delivered in the same order as the type tag identifier bytes in the Type Tag. The table below shows some of common Tag Ids and their encoding.

Tag ID	Data Type	Encoding
i	int32	32-bit big endian two's complement integer
f	float32	32-bit big-endian IEEE 754 floating point number
s	OSC-string	A sequence of non-null ASCII characters followed by a null, followed by 0-3 additional null characters to make the total bit length a multiple of 32.
b	OSC-blob	An int32 size count followed by that many 8-bit bytes of arbitrary binary data, followed by 0-3 additional zero bytes to make the total number of bits a multiple of 32.
m	midi	4 byte MIDI message. Bytes from MSB to LSB are: port id, status byte, data1, data2
T	boolean	No data encoded for a boolean item
F	boolean	No data encoded for a boolean item.

Example Encoded Message

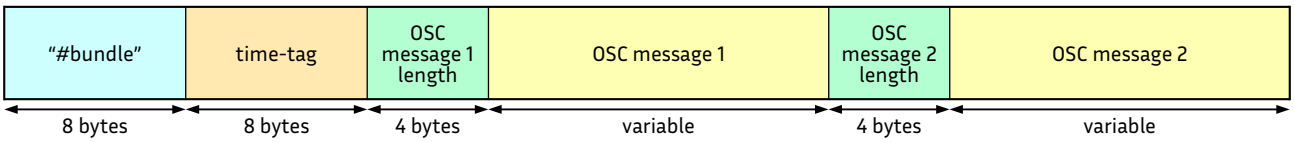
The example encoding has an address of **/myaddr/level** with one **integer** parameter with a value of **8**.

Byte No	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
ASCII	/	m	y	a	d	d	r	/	l	e	v	e	l	'\0'	'\0'	'\0'	,	i	'\0'	'\0'				
Hex	2f	6d	79	61	64	64	72	2f	6c	65	76	45	6c	00	00	00	2c	69	00	00	00	00	00	08



OSC Bundles

Some control events require several commands sent in the minimum possible time. The OSC bundle mechanism is an outer wrapper that combines multiple OSC packets into a single transmission block. The structure of a bundle is shown below:



A bundle is identified by an initial OSC string of **#bundle**. The bundle ident is followed by an eight byte time tag. The upper 32-bits are the number of seconds since midnight on January 1st 1900. The lower 32-bits are the fractional parts of a second to a precision of about 200 picoseconds. This is the same time format used in NTP time messages. The encoded value is the instant the commands in the bundle should be actioned by the server. There is a special time value comprising 63 zero-bits followed by a 1 which has the meaning ***immediately***.

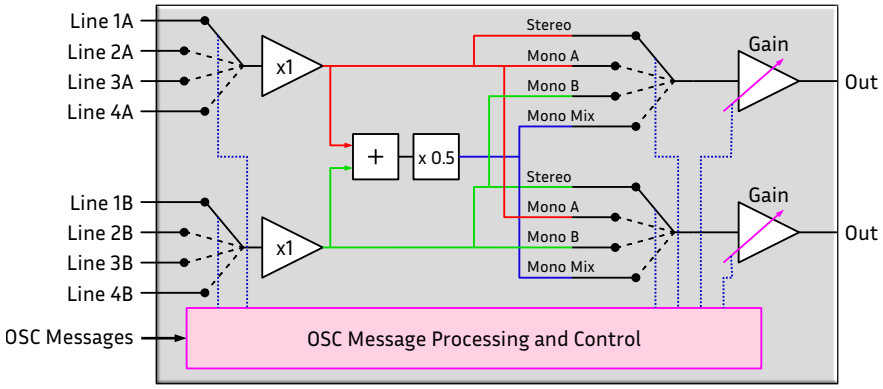
The ident and time tag fields are followed by any number of OSC messages, each message preceded by the length of the message in bytes. The OSC message may be a simple message or another bundle.

The CasparCG server contains an OSC client that uses OSC bundles to output status reports. The SVT CasparCG client can send OSC messages to other equipment. The default wrapping uses a simple message structure, but a tick box enables the encode to be changed to a bundle.

OSC Address Schemes

The OSC address scheme is highly adaptable. The address scheme is very similar to the one used for web browsing, starting with just a single **/** rather than **http://**. The address scheme uses a **/root/branch/twig/leaf** structure. The final element in the address is the controlled target, known as the **OSC Method**.

The diagram below shows a hypothetical audio amplifier with OSC remote control of some facilities.



The amplifier block has two input chains, notionally for left and right channels.

There are two switches in each chain, one selects the input source and one the input mode. The available modes are stereo, mono from left, mono from right or a 50-50 mix of left and right. The mode switches of each chain are independently controlled, enabling swapping of left and right sources if required. The switch positions and output gain of each chain is controllable via OSC messages.

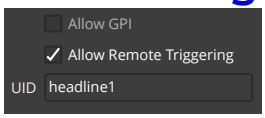
A possible address scheme starts with a root of **/amp** that divides into two sub-addresses of **/amp/left** and **/amp/right**. The controllable hardware, the OSC methods, are named as **input**, **mode** and **gain**, giving an address set for the left channel of:

/amp/left/input	use an integer parameter for the switch position.
/amp/left/mode	use an integer parameter to select the mode.
/amp/left/gain	use a floating point value.

OSC clients can send address strings that include wild characters, such as **?** and *****, used in address comparisons. A question mark matches a single character, and an asterisk matches a string of zero or more characters. An OSC server with two addresses /myunit/output/1/gain and /myunit/output/2/gain can be sent a message directed at /myunit/output/?/gain to set the same gain for both nodes.

There is also a special character code of **//** which acts a shortcut to a common element of a control address, enabling the shortened address of the form **//gain** or **//left/mode**.

OSC message Triggers in CasparCG Client



The SVT CasparCG client includes an OSC server that enables an external OSC client to trigger items in the active rundown. The screen grab at the left illustrates the entries in a rundown item that enable OSC triggering. The string entered in the UID box forms part of the trigger address.

The triggered OSC address is of the form **/control/<UID_name>/<action>** where <UID_name> is the target name entered in the UID box and <action> is the control process required. For example:

/control/headline1/play

The supported OSC <action> names are:

stop	play	playnow	load
pause	next	update	invoke
preview	clear	clearvideolayer	clearchannel

The user selected <UID_name> must follow general OSC rule (no spaces, none of the reserved characters). The OSC message must include a single integer parameter with value 1 to be recognised as a valid control string.

The OSC commands can be sent direct via UDP to the client or they can be sent as WebSocket JSON traffic.

CasparCG Client OSC Rundown Controls

The SVT client rundown can also be controlled by OSC messages. This control mechanism provides remote operation of the rundown as if the user were directly controlling the rundown playout from a keyboard. All the commands use an OSC address of the form **/control/<action>** where <action> is one of the entries in the table below.

stop	play	playnow	load
pause	next	update	invoke
preview	clear	clearvideolayer	clearchannel
down	up	playnowifchannel	playandautostep
playnowandautostep			

The majority of the control values are self-documenting. One, **playnowifchannel**, is less obvious. This value causes the SVT client to look at the item properties and to only issue the play action if the rundown item is a CasparCG control. If the playnowifchannel is actioned by a group, any command such as ATEM, TriCaster or GPI output are not triggered.

Using Web Sockets for OSC Control from a Browser

It is possible to implement simple control operations from a web page. The html code below is a basic implementation. It has one button that plays a remote control enabled clip with a control UID of “1”. It has four buttons that can move the cursor up and down the active rundown, play the selected item, or play the selected item and move the cursor down to highlight the following item. These four buttons operate if the rundown is remote control enabled (via Remote control enable item in the rundown menu).

```
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="UTF-8">
	<meta name="viewport" content="width=device-width, initial-scale=1.0">
	<title>Simple CasparCG Websocket control client</title>
</head>

<script>
// CCG control from a web page. Remember to enable the rundown Allow Remote
// Trigger to allow message cursor move processing by the rundown.

var ws
function onReady(){
	// Open the websocket connection to the CasparCG server port 4250 once the browser has
	// loaded all page elements. The tcp/ip address is the client host pc.
	ws = new WebSocket("ws://192.168.42.12:4250/");
}

function playClip1(){
	// This is the trigger for an item with UID of 1
	ws.send('{"path" : "/control/1/play", "args" : [1]}');
}

function doup(){
	ws.send('{"path" : "/control/up", "args" : [1]}');
}

function dodown(){
	ws.send('{"path" : "/control/down", "args" : [1]}');
}

function doplay(){
	ws.send('{"path" : "/control/play", "args" : [1]}');
}

function doplayandadvance(){
	ws.send('{"path" : "/control/playandautostep", "args" : [1]}');
}

</script>
<body onload="onReady()">
	<input id="Titles" type="button" value="slide 1" onclick="playClip1();"/><br>
	<input id="UP" type="button" value="Cursor Up" onclick="doup()"/>
	<input id="DOWN" type="button" value="Cursor Down" onclick="dodown();" />
	<input id="TAKE" type="button" value="Play at Cursor" onclick="doplay();" />
	<input id="TAKEAUTO" type="button" value="Play and Advance" onclick="doplayandadvance();"/>
</body>
</html>
```

CasparCG Server OSC Data Output

CasparCG server uses OSC to emit the current server status to both connected clients and fixed addresses defined in the server configuration file. Because this status data includes details of all active channels, the audio levels etc., the OSC bundle mechanism is used to combine several status items into a single network transmission. The data block below was captured using Wireshark exporting the UDP packet as both a hex dump and ASCII..

```
0000   00 e0 4c 36 57 79 1c 69 7a 02 fd 0a 08 00 45 00   ..L6Wy.iz.....E.
0010   00 e4 e0 7b 00 00 80 11 84 25 c0 a8 2a 0b c0 a8   ...{.....%..*...
0020   2a 0c e8 f0 18 6a 00 d0 7a 1e 23 62 75 6e 64 6c   *....j..z.#bundl
0030   65 00 00 00 00 00 00 00 00 01 00 00 00 44 2f 64   e............D/d
0040   69 61 67 2f 38 2f 74 65 78 74 00 00 00 00 2c 73   iag/8/text....,s
0050   00 00 66 66 6d 70 65 67 5b 48 61 72 62 6f 75 72   ..ffmpeg[Harbour
0060   5f 43 6c 69 70 2e 6d 6f 76 7c 37 32 30 78 35 37   _Clip.mov|720x57
0070   36 69 35 30 2e 30 30 7c 36 39 36 2f 31 35 31 32   6i50.00|696/1512
0080   5d 00 00 00 00 34 2f 63 68 61 6e 6e 65 6c 2f 31   ]....4/channel/1
0090   2f 73 74 61 67 65 2f 6c 61 79 65 72 2f 31 30 2f   /stage/layer/10/
00a0   70 72 6f 64 75 63 65 72 2f 74 79 70 65 00 2c 73   producer/type.,s
00b0   00 00 66 66 6d 70 65 67 00 00 00 00 00 34 2f 63   ..ffmpeg.....4/c
00c0   68 61 6e 6e 65 6c 2f 31 2f 73 74 61 67 65 2f 6c   hannel/1/stage/l
00d0   61 79 65 72 2f 31 30 2f 70 72 6f 66 69 6c 65 72   ayer/10/profiler
00e0   2f 74 69 6d 65 00 2c 66 66 00 00 00 00 00 3c a3   /time.,ff.....<.
00f0   d7 0a                                             ..
```

The start of the bundle is shown by the **blue** text. The bundle header is followed by the **time** at which the command should be actioned, in this instance immediately. The first OSC message length is 44 (hex) bytes long. The address field of the first message is **/diag/8/text** which has one null character string terminate bye and 3 padding bytes to make the string a multiple of 4-bytes long.

The value carried in the message is a string, with the string text being:

ffmpeg[Harbour_Clip.mov|720x576i50.00|696/1512]

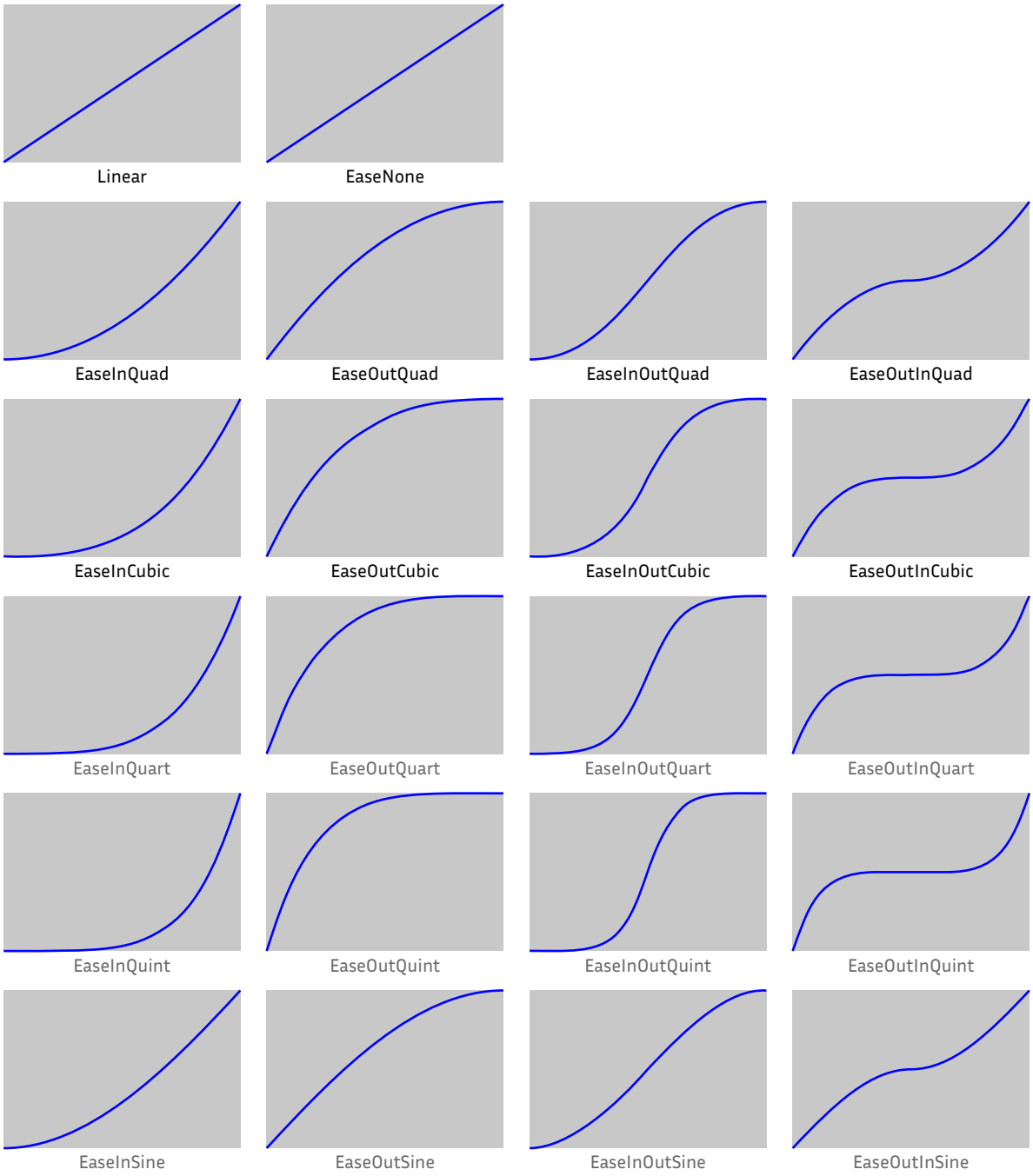
A client can extract the clip name and the playout progress which is frame 696 of 1512 frames total. The timebase is present in the message allowing a computation of the time remaining in the playout.

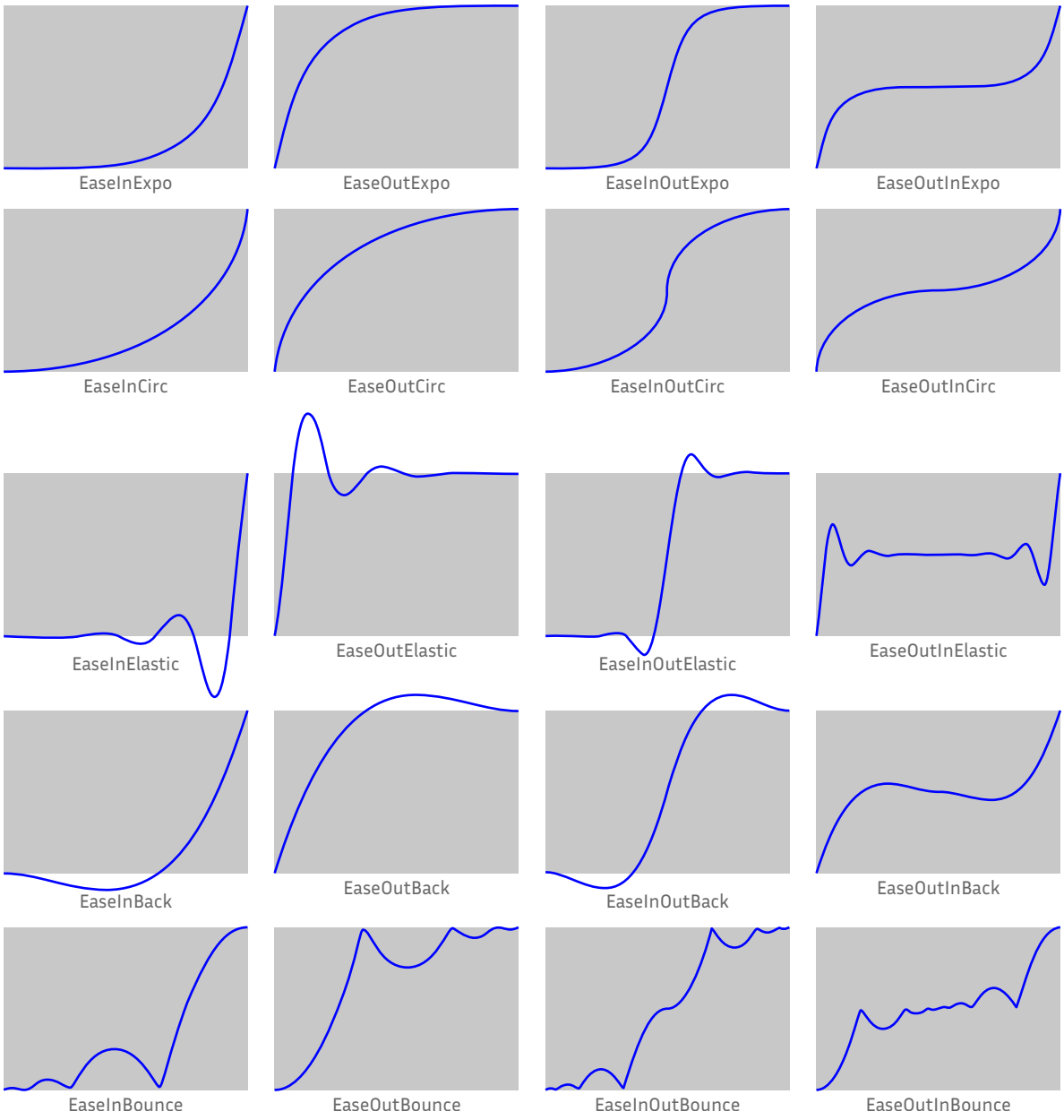
Appendix C - Transition Easing

CasparCG implements a standard set of transition easing functions published by Robert Penner, originally for use in flash animations. For more detail, including the equations, see http://robertpenner.com/easing/.

In the following graphs the vertical axis is the value change for the modified parameter (mix percentage, position etc). The horizontal axis is effect time expressed as a percentage (0% to 100%) of the user-set physical transition time.

The Penner equation implementations generally have an EaseNone function rather than the more descriptive Linear function. Linear combination is widely used in TV equipment for mixes etc. CasparCG server has added code that maps a request for a Linear transition to the EaseNone transition (actually called Linear by Penner!).




EaseInExpo
EaseOutExpo
EaseInOutExpo
EaseOutInExpo
EaseInCirc
EaseOutCirc
EaseInOutCirc
EaseOutInCirc
EaseInElastic
EaseOutElastic
EaseInOutElastic
EaseOutInElastic
EaseInBack
EaseOutBack
EaseInOutBack
EaseOutInBack
EaseInBounce
EaseOutBounce
EaseInOutBounce
EaseOutInBounce

Appendix D - Software Versions

There are several branches and versions of CasparCG server. At September 2022 the two most relevant versions are the outlined below.

Server Version 2.3.3_LTS (Long Term Support)

This branch is managed by the development team at SVT. This server computer for this version requires a graphics card that supports OpenGL version 4.5 or higher. The ffmpeg library and the OSC status outputs require a client version 2.2 to show clip playout progress.

The design targets in this branch are operational stability with a support life of more than two years from the initial release in mid-2020.

This version provides native support for Newtek NDI input and output streams via the NDI producer and NDI consumer.

There have been three incremental releases to the initial version 2.3.0 release:

Version 2.3.1 - January 2021

Enables Flash graphic templates to operate after Windows removed Flash support.

Version 2.3.2 - March 2021

Bug fixes -Intel Threading Library and ffmpeg producers.

Version 2.3.3 - March 2021

Added Image Scroll producer that was removed during early development of the LTS code base. This allows simple stills images to create roller and crawler captions.

Software Repository:

https://github.com/CasparCG/Server/releases

Server Version 2.1.12_NRK

This NRK supported fork of the server requires a graphics card that supports OpenGL version 3.0 or higher. The matching clients are versions 2.0.8 and 2.0.9 (which shows V2.0.8 on the About... display, but the changelog file shows the 2.0.9 developments).

A key target of this branch is stability of operation as it is deployed in live news playout operations. It has the ability to trigger server playout events based on EBU/SMPTE timecode connected to the physical input/output cards, or via the use of the computer's internal clock.

This code branch can create NDI output streams, labelled iVGA, an earlier name for NDI. This code branch also supports some templated operations using Adobe Photoshop files with the Adobe timeline.

Recording and Streaming Issue

There is a known bug in the **Windows** version this branch that causes the server to crash when the file producer or ffmpeg producer is removed from an output. The development team have used significant effort to locate the cause, but issue remains unsolved. There are two consequences:

- It is not possible to create recordings of a channel
- It is not possible to stream a live client preview

The development team report that switching the Microsoft VS2015 linker to /Incremental mode removes the issue, but there is a consequential performance loss. The SVT LTS version seems to have identical code base, but is built using a newer Microsoft compiler/linker. There is significant risk that porting the code to a newer Microsoft version of VS will be take a long time and suffer many issues as various libraries will also need updating. The bug report for the issue is available here.

The Linux version of the server records as expected.

Software Repository:
https://github.com/nrkno/tv-automation-casparcg-server/releases

Appendix E - Toggling Software Versions

Server Version

The elements in the casparcg.config file are described earlier in this document. The major differences between the configuration files for the NRK and SVT branches are:

- There are less folder definitions in the <paths> tag block. The <font> and <thumbnail> path definitions are not required for SVT branch.
- Definitions are required for the media scanner used in the SVT version.
- The default audio mode is stereo in the NRK version and 8-channel passthrough in the SVT version.

The other key difference between the servers is the client requirement - version 2.0.8/2.0.9 for the NRK branch, and version 2.2.0 for the SVT version. Where client time values set in frames are used, such as fade durations, the values use the channel picture rate count in version 2.0.9 and progressive frame count in version 2.2.0. Thus a version 2.2 client on a channel operating at 1080i5000 uses a count of 25 for half a second.

A key change in the SVT 2.2 and later branches is splitting the media scanner and thumbnail generation out from the server code and using a new support tool to provide the services. Clients still request the information via AMCP commands directed to the server, but the server requests the data from the media scanner tool. The SVT distribution includes a batch file that runs the scanner, then runs the server. Whilst the two elements can be manually started in sequence the batch file is the better process to use. The NRK branch also has a batch file, but the only thing it runs is the server.

The server command line can include the name of the configuration file to use, allowing easy support where different channel configurations are required - for example one for SD-SDI and one for HD-SDI operations. The provided batch files do not provide support for this, although it could be added to the NRK version, but the media scanner used in the SVT server does not support a command line change of configuration file name to use.

The simplest process to manage both code branch configurations is keeping the required configuration files in their server folder, but with an added name element. For example, for HD and SD operations have two files called casparcg_SD.config and casparcg_HD.config.

Create one or more batch files whose function is copying the desired configuration file into the standard casparcg.config file name.

Keep shortcuts to the server startup batch files on the computer desktop as well as in the main folder of the server code. Ensure all users know that the system must ***only*** be started via the batch files.

The system installer chooses the server LOG file folder by an entry in the <paths></paths> tag pair in the server configuration file. The log files for the different server versions can use a common folder. Using a single folder simplifies the log management process (see ***here*** for more detail).

Client Version

The default client configuration uses a single folder that has the client configuration database file, with a sub-folder for client log files. A Windows client host uses a folder called .CasparCG that is stored in the logged on users home folder on the primary disk drive. If the user is called casparcg the full path to the configuration and log store is:

C:\Users\casparcg\.CasparCG

There is a sub-folder called Client which has the database called **Database.s3db** and a folder called **Logs**.

The operational issue is that the fields in the database file have different content for client version 2.0.9 and client version 2.2. Operating more than one client therefore needs tight operational discipline.

The client program checks it's startup command line looking for any options that may be present. One option is to choose a different database file name or path. The simplest mechanism to implement this change of file name is to use a shortcut with the command line of the format:

```
"C:/CasparCG/ClientV2d0d9/casparCG Client.exe" -t c:/casparclient/usedb209.s3db
```

where -**t** is the option tag for client database name.

If someone now directly runs a client EXE application the database uses the standard location and name, and does not corrupt the normal working file for a specific client.

Good operational practice requires that after a client is fully configured a copy of the configuration is taken and stored in a "safe" location. If there is a subsequent problem with the working copy, the safety copy is used to restore basic configuration. It may still be necessary to use the client configuration tool to delete thumbnail pictures from the database, then allow the thumbnails for the current media set to be re-loaded.

An alternate method of managing the use of two clients is making a manual copy of each client configuration database after every configuration change, then copying that file to the default name and location before starting the client.

Appendix F - Media Database Alignment

Symptom
Media file list on disc does not match the list(s) in the client library boxes.

Software tools required
1. Windows File Explorer

2. DB Browser for SQLite

Background
Server version 2.2 uses an associated task to scan for media and media changes. This task stores the scan results in a Pouch database stored on the CasparCG server host computer. This database information is sent to the client when requested, such as when the client connects to the server. The client stores the received information in a SQLite database on the client host computer.

The software design assumes that the scanner is always active, and hence it detects changes in the media folder, updating the Pouch database as needed, passing the changes to the connected client(s).

If the scanner is not running when media is added to or deleted from the media store, the media ***may*** fail to be detected when the scanner next starts. Media type detection as video still or audio sometimes fails. When the media is re-scanned and correctly detected the change does not get reported to the client.

The simplest fix for these issues is to force a re-build of the scanner database then the client database..

Step 1 - Stopping Tasks
On the CasparCG server computer(s) stop the server executable and the scanner. The server is closed by typing kill<cr> in the DOS server window. The scanner DOS style window can be closed by the standard windows close icon in the title bar.

On all client machines, stop the CasparCG client using the standard windows close mechanism.

Step 2 - Server Computer(s)
On each computer locate the folder that has the CasparCG server executable. Delete **_media** and **pouch__all_dbs__** folders.

Step 3 - Client Computer(s)
Locate the folder that stores the client database file.

This may be the default location, or a user selected storage location. If the client is using a user selected location this will be shown on the Target line of the shortcut that starts the client. A user database path and location is entered after a **-t** identifier. The default is in a sub-folder of the Users home folder. For a user called “myname” the database file is:

```
C:\Users\myname\.CasparCG\Client\Database.s3db
```

Run the DB Browser for SQLite program. Click the Open Database button on the shortcuts bar. Set the file type to All Files and locate the database file you identified, then click the Open button.

Select the Browse Data tab, then choose the Library table via the drop-down list box. Select all the records in the table, then click the delete record icon (table with a red circle containing a minus sign) to delete all records. Choose the Thumbnail table, select all records and delete them.

Close the database (File menu) then close the DB Browser program.

Step 4 - Restart CasparCG Server(s)

Start the media scanner and CasparCG server using a desktop shortcut or the **casparcg_auto_restart.bat** batch file in the server home folder.

This rebuilds the pouch database on the server computer. There is a lot of activity in the scanner window. When this completes move to step 5.

Step 5 - Restart CasparCG Client

Use any appropriate shortcut on the client computer, as this selects the correct local client database. The client contacts the servers defined in the client configuration server list, updating the client media database and thumbnails.

Appendix G - Client Record Presets

This appendix documents the process of editing three client source code files to add new record presets. It assumes the reader is familiar with downloading file sets from GitHub, installing compiler software on a host PC, and running application build processes.

The distribution version of client version 2.2 includes two sets of preset control properties in the File Recorder tool. The build process is documented in the development paragraph of the client GitHub repository readme.md file at https://github.com/CasparCG/client. When the source code edits are complete, run the build operation using the target platform batch file included in the client distribution. For example, the Windows application is built using the batch file called set-variables-and-build-windows.bat.

Test and prove the new record properties using a custom command via the distribution version of the client. Save the working parameters list used in the command into a simple text file that will be pasted into the client source code during the edit. Three files need editing:

1. src/common/Global.h

2. src/Widgets/Inspector/InspectorFileRecorderWidget.cpp

3. src/Widgets/Inspector/InspectorFileRecorderWidget.ui

File: Global.h
Open src/common/Global.h in a suitable editor such as Visual Studio Code. Use the editor search tool to locate the segment that starts `namespace FileRecorder`. This block defines the name of the preset that will be seen in the drop down list and the associated command string. In May 2022 the distribution code was:

```
namespace FileRecorder
{
    static const QString DEFAULT_OUTPUT = "Output.mxf";
    static const QString DEFAULT_DNXHD_CODEC = "dnxhd";
    static const QString DEFAULT_H264_CODEC = "h264";
    static const QString DEFAULT_DNXHD_PRESET =
    "-codec:v dnxhd -b:v 120M -flags:v +ilme+ildct -threads:v 4 -filter:v
     format=yuv422p,interlace";
    static const QString DEFAULT_H264_PRESET =
    "-codec:v libx264 -flags:v +ilme+ildct -threads:v 4 -filter:v interlace
     -preset:v veryfast";
    static const QString DEFAULT_CODEC = DEFAULT_DNXHD_CODEC;
    static const QString DEFAULT_PRESET = DEFAULT_DNXHD_PRESET;
    static const bool DEFAULT_WITH_ALPHA = false;
}
```

Each new preset requires two lines adding to the namespace block, one defining the name and the second defining the properties. Here are two example strings for a progressive-scan dnxhd recording:

```
static const Qstring DEFAULT_DNXHDPROG_CODEC = “dnxhd_progressive”

static const QString DEFAULT_DNXHDPROG_PRESET =
    "-codec:v dnxhd -b:v 120M -threads:v 4 -filter:v format=yuv422p";
```

File: InspectorFileRecorderWidget.cpp
Open src/Widgets/Inspector/InspectorFileRecorderWidget.cpp in the text editor. Locate the c++ function:

```
void InspectorFileRecorderWidget::codecChanged(QString codec)
```

Add an extra “else if” clause for each extra preset setting.

The following example adds the dnxhd progressive preset described earlier in this document to the list of presets. The first line of the action code adds the correct file extent to the record file name set by the user (or added to Output if the default name has not been edited). The result is stored in the File Recorder instance Inspector **name** field. The second line of the action code copies the new properties string into the preset field of the item inspector. An example of the edited function is shown below, with the extra code coloured red.

```
void InspectorFileRecorderWidget::codecChanged(QString codec)
{
    this->command->setCodec(codec);

    QFileInfo output(this->command->getOutput());
    if (codec == FileRecorder::DEFAULT_DNXHD_CODEC)
    {
        this->lineEditOutput->setText(output.completeBaseName() + ".mxf");
        this->lineEditPreset->setText(FileRecorder::DEFAULT_DNXHD_PRESET);
    }
    else if (codec == FileRecorder::DEFAULT_H264_CODEC)
    {
        this->lineEditOutput->setText(output.completeBaseName() + ".mp4");
        this->lineEditPreset->setText(FileRecorder::DEFAULT_H264_PRESET);
    }
    else if (codec == FileRecorder::DEFAULT_DNXHDPROG_CODEC)
    {
        this->lineEditOutput->setText(output.completeBaseName() + ".mxf");
        this->lineEditPreset->setText(FileRecorder::DEFAULT_DNXHDPROG_PRESET);
    }
}
```

When the item is run the values are extracted from the item fields, formatted as needed, and sent to the selected server.

File: src/Widgets/Inspector/InspectorFileRecorderWidget.ui
This xml file is read by the Qt software that creates the computer monitor display. The user edit adds new entries to the the drop-down lists. Locate the property chain shown below, and add one extra item per new preset.

```
<item row="1" column="1">
    <widget class="QComboBox" name="comboBoxCodec">
     <property name="sizePolicy">
      <sizepolicy hsizetype="Expanding" vsizetype="Fixed">
       <horstretch>0</horstretch>
       <verstretch>0</verstretch>
      </sizepolicy>
     </property>
     <item>
      <property name="text">
       <string>dnxhd</string>
      </property>
     </item>
     <item>
      <property name="text">
       <string>h264</string>
      </property>
     </item>
    </widget>
   </item>
```

Add the new entries after the item that has the <string>h264</string>. The example progressive preset requires the lines below: Note the <string> property ***must*** match the string defined in file global.h namespace FileRecorder.

```
<item>
 <property name="text">
  <string>dnxhd_progressive</string>
 </property>
</item>
```

Save all edited file, run the build process, test the built instance of the client.

Deploy the updated client to user workstations.

Appendix H - Example Lower Third

The code in this appendix between <!**doctype**> and </**html**> is a simple static lower third template for CasparCG. All required elements, including CSS, Javascript and document layout are inline.

When the template is played it displays a solid red bar over which two lines of white text are displayed. The text uses Arial as there should be a .TTF for this typeface on all host machines. The displayed text is sent by the client using the identifying keys **f0** (upper line) and **f1** (lower line). The template supports updates to the text content, implementing changes as simple cuts.

The sizes of all elements use viewport relative sizes, where the viewport width is 100vw and the viewport height is 100vh. The template is therefore channel resolution independent.

HTML Code

```
<!doctype html>
<html lang="en">
<head>
   <meta charset="UTF-8">

<!--
   Further meta items about the Template. The values can be read by a suitable application.
   None of the items are required to operate the template, they are only defined here for
   potential future use.
-->
   <meta name="ccg_template" content="Static"> <!-- or "Dynamic" -->
   <meta name="ccg_author" content="programmer name here">
   <meta name="ccg_contact" content="programmer email">
   <meta name="ccg_programme" content="programme name">  <!-- Programme that uses template -->
   <meta name="ccg_designdate" content="Graphic Design first use month and year - e.g. July 2019">
   <meta name="ccg_summary" content="Short Description of the template">
   <meta name="ccg_resolution" content="All">   <!-- PAL/720/1080 etc. -->
   <meta name="ccg_licence" content="MIT">
   <meta name="ccg_fontlist" content="Arial">

   <title>CasparCG Template - Example Static 01</title>
<!--
   This CasparCG HTML template was created by <enter name here>.
   Contact <Enter email here>

   Reason for template/programme applications etc.

   This template displays a lower third with a name field and a designation field, both
   shown on top of a simple colour strap.
   Expected field names from CasparCG:
      f0      Name to display
      f1      Designation/Job Title

   The template accepts user parameters wrapped in either XML or JSON.
-->

<style type="text/css">
/*
   Define some "CSS Variables" values that may be needed in multiple CSS selectors. Embedded
   Chromium in CasparCG is a webkit based browser, so some selectors must be stated in both
   Standard and webkit formats.
*/
:root {
   --scrTop:0vh; --scrLeft:0vw; --scrWidth:100vw; --scrHeight:100vh;
   --gfxContentEdge:10vw; --gfxContentWidth:80vw;
   --strapColour:#b80000; --nameColour:#f0f0f0;
}
/* CSS Variables that define strap properties relative to straptop element */
:root {
   --strap1top: 0vh;
   --strap1left: var(--gfxContentEdge);
   --strap1width: var(--gfxContentWidth);
   --strap1height: 10vh;
}
```

```
html {
   Box-Sizing:border-box;           /* Use all available space */
   backface-visibility: hidden;     /* Useful for 3d Manipulations */
   -webkit-box-sizing: border-box;
   -webkit-backface-visibility: hidden;
   -webkit-transition: translate3d(0,0,0);
}

html, body {
   Margin:0;
   Padding:0;
   Background:transparent;    /* Useful when using a web browser during development */
   Overflow:hidden;           /* Prevent scroll bars from minor overflows of content */
   -webkit-font-smoothing:antialiased !important;
}

.hideMe { opacity:0; }

/* Define a selector that controls the vertical placement of the coloured bar. */
.straptop {
   position:absolute; top: 80vh; width:var(--scrWidth);
}

.strap01 {
   position:absolute; top:var(--strap1top); left:var(--strap1left);
   width:var(--strap1width); height:var(--strap1height);
   background-color: var(--strapColour);
}

.f0_posn {
   position:absolute; top:0vh; left:1vw;
   width:var(--strap1width); height:var(--strap1height);
}

.f0_font {
   font-family: Arial; font-weight: bold;
   font-size: 4.6vh; font-kerning:normal; color:var(--nameColour);
   white-space:pre; text-align: left;
    vertical-align: top;
}

.f1_posn {
   position:absolute; top:5vh; left:1vw;
   width:var(--strap1width); height:var(--strap1height);
}

.f1_font {
   font-family: Arial; font-weight:normal; font-style: italic;
   font-size: 3.6vh; font-kerning:normal; color:var(--nameColour);
   white-space:pre; text-align: left;
}
</style>

<script type="text/javascript">
'use strict';
function init(){
   // Insert code in this function to perform any required post-load initialisations.
}

// Module Wide Variables
let dataCaspar = {}; // Used in parsing data from client,
let updateCalled = false;

/*
   ================================================================================================
   CasparCG support library functions that parse data from a client, and insert the data into
   the template field placeholders.
*/
```

```
/*
   function escapeHtml(srcString) returns a string in which any HTML reserved characters
   in the user srcString are replaced by their HTML markup equivalent. For example the left elbow
   bracket '<' is converted to '&lt;'. This enables text containing a < to be displayed by an
   HTML element using code of the form:

      let ptr = document.getElementById("f0");
      ptr.innerHTML = escapeHtml(srcString);

   Alterately use the textContent object method that includes the HTML escape function:

      let ptr = document.getElementById("f0");
      ptr.textContent = "myString";
*/
function escapeHtml(unsafe) {
   return unsafe.replace(/&/g, "&amp;").replace(/</g, "&lt;").replace(/>/g, "&gt;")
                .replace(/"/g, "&quot;").replace(/'/g, "&#039;");
}

/*
   function XML2JSON(node) converts the XML string passed as a parameter 'node' into a JSON
   object containing the field names and their values. For example, if there are two fields
   called 'f0' and 'f1' in the user "node" data, the returned JSON object takes the form:

   { f0: 'Input value 1 from client', f1: 'Input value 2 from client' }

   This function require zero pre-knowledge of the number of fields nor their associated names.
*/
function XML2JSON(node) {
   let data = {};   // result object
   for (let k=0; k<node.length; k++)   {
      let idCaspar = node[k].getAttribute("id");
      let valCaspar = node[k].childNodes[0].getAttribute("value");
      if (idCaspar != undefined && valCaspar != undefined) {
         data[idCaspar] = valCaspar;
      };
   }
   return data;
}

/*
   function parseCaspar(str) takes the user data from the client wrapped in either XML
   or stringified JSON, producing a globally accessible JSON object called 'dataCaspar'

   When the user data is sent in an XML wrapped format, the string contains XML of the form:

   <templateData>
      <componentData id="#idCaspar#">
         <data id="text" value="#valCaspar#" />
      </componentData>
         :
         :
      <componentData id="#idCaspar#">
         <data id="text" value="#valCaspar#" />
      </componentData>
   </templateData>

   where #idCaspar# is the name of the user field, for example f0, and #valCaspar# is the
   associated value for that field.

   The string from the client does not contain any line breaks, and all reserved
   characters are escaped by a backslash, thus a quotation mark (") is sent as
   backslash+quote (\")

   For JSON emitting capable clients (including custom clients) the structure of the
   client string has the form:

   "{\"f0\":\"Value of f0\",\"f1\":\"Value of f1\"}"

   As shown above, escape characters are used to preserve the meaning of the outer quote marks
   that wrap the user data.
```

```
   function parseCaspar(str) uses the first character of the data string to determine the wrapping
   Process used. If the first character is a left angle bracket (<) the string is XML wrapped.
*/
function parseCaspar(str)   {
   let xmlDoc;
   let parser;
   if (str != undefined) {    // Check the str parameter has a value, even if it is null
      if (str.length != 0) {  // Check if the string is a null string (zero length)
         if (str[0] === '<') {
            // Convert XML wrapped data into JSON via conversion of text string into a
            // Document Object Model.
            if (window.DOMParser) {
               parser = new DOMParser();
               xmlDoc = parser.parseFromString(str,"text/xml");
            }
            dataCaspar = XML2JSON(xmlDoc.documentElement.childNodes);
         }
         else {
            // Parse stringified JSON into a JSON object
            dataCaspar = JSON.parse(str);
         }
      }
   }
}

/*
   function dataCaspar(objectName) looks for HTML elements whose id matches a user data field name.
   If dataCaspar contains a field called f0 this function reads the DOM tree looking for an
   element identifier that is of the form:

      <div id="f0"></div> or <label id="f0"></label>

   If the element exists the user data is inserted into the element after conversion
   of reserved HTML characters such as '<' to their HTML description.
*/
function dataInsert(dataCaspar)   {
   for (let idCaspar in dataCaspar) {
      let idTemplate = document.getElementById(idCaspar);
      if (idTemplate != undefined) {
         idTemplate.innerHTML = escapeHtml(dataCaspar[idCaspar]);
      }
   }
}

/*
   ****************************************
   * CasparCG Standard Interface Functions *
   ****************************************
*/

/*
   function play() is called by the CasparCG template host module when a AMCP "CG PLAY" or
   "CG ADD" with Autoplay flag set is received from the client. The purpose of the function
   is to run any input animation required by the template design then pause the animation
   'timeline' at the end of the input animation.

   Note: There are differences between server 2.0.7 and later versions.
   =====

   In version 2.0.7 the user data string from the client is passed directly to the play
   function, so the function call is play(userString). The play function must parse the
   string, then run any input animation.

   Later versions of the server first call the update(userString) function, then call
   the play() function without any user parameter. Supporting all versions of server
   requires the play function to test for the presence of a string parameter, processing
   as required.
*/
function play(str) {
   if (str != undefined) {
      update(str);
   }
```

```
   if (updateCalled) {
      // User data has been loaded into the placeholders. We are ready to show the caption.
      // No input animation in this boilerplate template code, so remove the hideMe class.
      document.getElementById("strapholder").classList.remove("hideMe");
   }
}

/*
   function stop() is called when the user client sends a CG STOP command. This enables
   an output animation to be run before the template is removed by the template host.

   In this boilerplate code there is no output animation. The code hides the uppermost
   element of the strap by adding a CSS class, thus hiding all children as well. It
   checks if the template is running in CasparCG or in a conventional browser, calling
   the appropriate window close function.
*/
function stop() {
      document.getElementById("strapholder").classList.add("hideMe");

   setTimeout( () => {
      if (window.caspar) window.remove();
      else window.close();
   }, 800);
}

/*
   function next() is called without parameters. It is used in templates that have multiple
   timeline states to their animation. When next() is called, it causes the animation timeline
   to re-start from current position, playing forward until the next pause point.
*/
function next() {
   // Dummy function in this template. No timeline implementation
}

/*
   function update(userString) is called when the user client sends a play command (eg by pressing
   the F2 key in the SVT client), and when the users sends a CG UPDATE command (SVT client F6 key).
   This allows the instance data shown by the template to be updated without any
   'template remove - template start' disturbances.

   Where multiple user fields are present, the template must select the actions that happen when
   only a sub-set of fields are updated. For example, in a news lower third there may be a two-line
   information display - line 1 with a name, line 2 with a job or organisation description.

   How should the template process a change when just the job description changes? The answer is
   specific to each deployment.

   In this boilerplate template the update function implemnts a cut operation on all data fields
   sent in the user data block. Thus if there is existing display content for keys f0 and f1, but
   the new user string only contains data for the f1 key the f0 content remains unchanged on the
   Display.

   Update(str) becomes a much more complex code block when dynamic template operations are used.
   For example changing just the f0 placeholder text require three process steps:
   1) Animate the existing text content off the display
   2) Change the text content to the new value whilst the text field is hidden
   3) Animate the text field onto the display
*/
function update(str) {
   if (str != undefined) {
      parseCaspar(str);
      if (dataCaspar != undefined) {
         dataInsert(dataCaspar);
      }
   }

   updateCalled = true; // Record that update function has been called.
}

</script>

</head>
```

```
<!-- Define the page layout for the content -->
<!-- Define the body giving a function name that is called when all loading is complete -->
<body onload="init()">
   <!-- Place the strap and it's children on the display. -->
   <!-- Hide the content of strap and children until function play() is called -->
   <div id="strapholder" class="straptop hideMe">
      <div id="myStrap" class="strap01">
         <div id="f0" class="f0_font f0_posn"></div>
         <div id="f1" class="f1_font f1_posn"></div>
      </div>
   </div>
</body>

</html>
```

How to copy and test the template

To use the above text as a template in either a browser or in CasparCG server select all text starting at <!Doctype html> up to and including </html> and copy to the clipboard. Open a simple text editor or program editor such as Visual Studio Code, pasting the clipboard into an empty document. Use a paste mode that copies just the text without any highlighting.

NOTE: Remove any page numbers that copied as part of the cut and paste operation.

Save the file under a suitable name such as **demotemplate.html**.

To use the template in a CasparCG server copy the file into the server template folder. If a CasparCG client is already running when the file copy occurs, use the library refresh tool in the client to add the new template to the list of templates. The template can be instanced in a rundown, data entered for the f0 and f1 keys then run.

The template also runs in a browser, such as Chrome, by selecting the file in the finder list, right clicking the name and selecting the target program as Chrome browser. When the browser page shows as a blank display, right-click in the display area and choose Inspect which opens a second window that includes a Javascript console. Locate the console tab to make it the focus zone. Enter a command on the console similar to:

```
play('{"f0": "My name shows here", "f1": "My job title shows here"}')
```

When entering the above command take great care with the placement of the single and double quotation marks. Some people find the data entry is simplified by using a command that uses the built-in function that converts a JSON object to stringified JSON form:

```
play(JSON.stringify({f0:"CasparCG", f1:"Graphics made Cool"}))
```

Chrome was suggested as the browser, as this is closest to the embedded chromium browser used in CasparCG. Other browsers also work to show the template, but some error messages may be seen on the console display unless they are webkit based browsers.

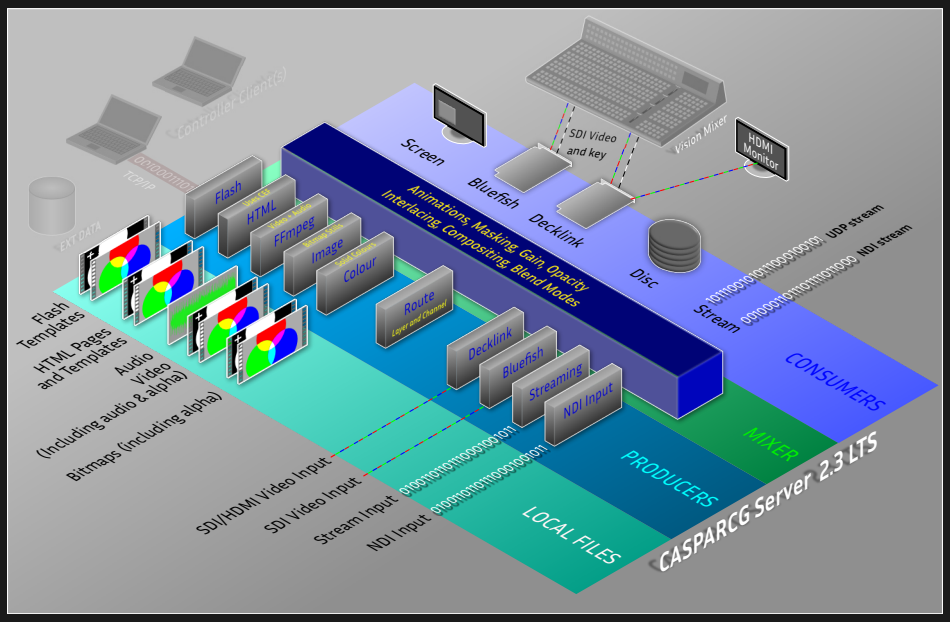
The content update process is invoked in a similar fashion:

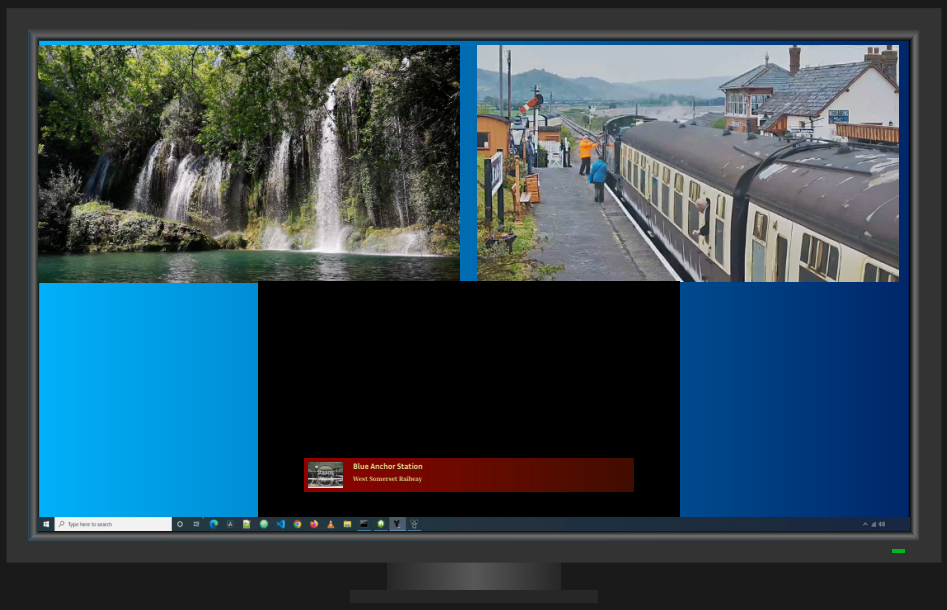
```
update(JSON.stringify({f0:"CasparCG User Forum", f1:"https://casparcgforum.org/"}))
```

Finally the template can be removed by calling the stop function:

```
stop()
```

This hides the text, and about 800 ms later closes the browser tab or window.


Controller Client(s)
TCP/IP
OSC/OSCQUERY
EXT DATA
Flash
HTML
FFmpeg
Image
Colour
Route
Layer and Channel
Animations, Masking, Gain, Opacity, Interlacing, Compositing, Blend Modes
Screen
Bluefish
Decklink
SDI Video and key
Vision Mixer
HDMI Monitor
Disc
Stream
UDP stream
NDI stream
Flash Templates
HTML Pages and Templates
Audio
Video (including audio & alpha)
Bitmaps (including alpha)
Decklink
Bluefish
Streaming
NDI Input
CONSUMERS
MIXER
PRODUCERS
LOCAL FILES
SDI/HDMI Video Input
SDI Video Input
Stream Input
NDI Input
CASPARCG Server 2.3 LTS


Blue Anchor Station
West Somerset Railway

AMW tech