# dcapp Installation and User Guide

version 1.1

NASA Johnson Space Center

## Table of Contents

# 1.0 Introduction

"dcapp" (pronounced “dee see app”) is a displays and controls software package designed for UNIX platforms, specifically MacOS and Linux. It is built upon standard UNIX technologies like OpenGL for graphics, libxml2 for input file parsing, and FreeType2 for font handling. For window management and event handling, it uses Cocoa on MacOS machines and X11 for Linux-based machines. It has built-in communication libraries to communicate with external Trick-based simulations (via trick_comm) and EDGE graphics (via EDGE’s remote commanding server (RCS)).

For more information, please contact:

Michael McFarlane
michael.r.mcfarlane@nasa.gov
ER7/Simulation and Graphics Branch
NASA Johnson Space Center

# 2.0 Installation

## 2.1 Mandatory Prerequisites

dcapp is designed to run on MacOS and Linux-based machines. Building dcapp requires a compiler that can accommodate the C++11 programming language. For all of the packages described hereafter, be sure to get “development” versions that include header files. These packages must be installed before building or running dcapp:

- OpenGL
- libxml2
- FreeType2

OpenGL is a standard environment for developing portable, interactive 2D and 3D graphics applications. It is a standard package on most MacOS and Linux installations, but it can be accessed at http://www.opengl.org if needed.

libxml2 is an XML file parser that is a standard package on most MacOS and Linux installations, but it can be accessed at http://xmlsoft.org if needed.

FreeType2 is a freely available software library for rendering fonts. It is capable of producing high-quality output (glyph images) of most vector- and bitmap- font formats. It is a standard package on most MacOS and Linux installations, but it can be accessed at http://www.freetype.org if needed.

## 2.2 Optional Prerequisites

If the user wants to use JPEG images in a dcapp display, then they should install either libjpeg or libjpeg-turbo prior to building dcapp. dcapp will build just fine without one of these packages, but it won't be able to process JPEG images unless one of the packages is properly installed.

Much of the PixelStream functionality associated with the MJPEG and VSM protocols is built upon curl. If the user needs this functionality, then they should install curl prior to building dcapp. dcapp will build without curl, but most of the MJPEG and VSM PixelStream capabilities will be unavailable to the user unless curl is properly installed.

If dcapp is to be run in conjunction with a Trick simulation, then the `trick-gte` command (or the `gte` command for Trick version 15 and earlier) must be accessible via the command line for dcapp to build correctly. Also, a stand-alone version of trick_comm must be successfully built prior to building dcapp. trick_comm is a Trick library that provides an interface to a Trick simulation via the Trick variable server. Note that some older versions of Trick do not automatically build the stand-alone version of trick_comm. If this is the case, install Trick, `cd` to ${TRICK_HOME}, and type "`make stand_alone`". Note that dcapp should work well with any Trick release numbered 10.2 or higher.

dcapp can be configured to monitor hardware inputs (dials, switches, etc.) via a controller area network (CAN) bus. CAN is a serial bus protocol used to connect individual systems and sensors over a single- or dual-wire networked data bus. Be sure that the CAN bus software is appropriately built and that the CANBUS_HOME environment variable is set to the directory containing the necessary header and library files.

dcapp can also monitor hardware inputs via a Hagstrom device. This requires the IDF package to be properly built and for the IDF_HOME environment variable to be set. If IDF_HOME isn't set, then dcapp will look for the IDF package at the same level in the directory tree as the dcapp package. If it still can't find IDF, then IDF functionality will not be available to the user.

## 2.3 dcapp

Extract the dcapp package if necessary, `cd` to the top level of the package, and type "`make`". This should build the dcapp executable within the dcapp.app/Contents/${OSSPEC} subdirectory, where OSSPEC is defined by the returned value of "`dcapp.app/Contents/dcapp-config --osspec`". On MacOS systems, OSSPEC is "MacOS". On other systems, it is typically set to a combination of `uname -s` (converted to lower case) followed by an underscore ("_") followed by `uname -m` (for instance, "linux_x86_64"). You should then add the returned value

of "`dcapp.app/Contents/dcapp-config --exepath`" to your $PATH environment variable if you intend to launch dcapp from the command line.

## 3.0 Activation

After following the instructions in section 2, simply type the following on the command line to activate dcapp:

```
dcapp file.xml [const=value...]
```

where `file.xml` is a full or partial path to a valid dcapp specfile (see section 4 for more information on dcapp specfiles). Note that the optional "`const=value`" constructs may be used as many times as needed to override the value of any constants defined within the specfile.

For instance, if a user wants to run dcapp with a specfile called myspec.xml but overriding the constants "WinWidth" and "WinHeight" with "480" and "640" respectively, the user would type the following command:

```
dcapp myspec.xml WinWidth=480 WinHeight=640
```

If a user chooses to run in debugging mode, which provides a wealth of debugging information to the console screen, the user may include a "-debug" argument anywhere on the command line after the specfile (`file.xml`).

Note that on MacOS, an alternative to launching dcapp via the command line is to use dcapp.app, which is automatically built during the "`make`" step described in section 2.3. dcapp.app can be launched like any MacOS application (double clicking it, launching it from the Dock, etc.). It brings up a simple user interface that requests the information described above from the user, then proceeds to launch dcapp accordingly.

## 4.0 Specfile

The dcapp specfile is a standard XML file used to customize the features and capabilities of dcapp. See http://www.w3.org/XML/ for more information about XML files, including valid file specifications, definition and usage of character entities, use of comments, etc. The elements contained within the dcapp specfile are detailed in this section.

### 4.1 Root Element

| Element | DCAPP |
|---|---|
| Parent | (none) |

| | |
|---|---|
| Children | (any) |
| Attributes | (none) |
| Description | All dcapp specfiles must contain this root element. All of the other elements, described in the following sections, must be enclosed within this root element. |

### 4.2 Universal Elements

These elements may appear anywhere within the dcapp specfile, and they may be embedded within any element that allows children.

| | |
|---|---|
| Element | Dummy |
| Parent | (any) |
| Children | (any) |
| Attributes | (none) |
| Description | This element does nothing besides allowing the user to group sub-elements. This is potentially useful when using XML's <xi:include> element, which requires included files to be "well-formed", which means, among other things, that the file must contain only one element at its root level. |

| | |
|---|---|
| Element | Include |
| Parent | (any) |
| Children | (any) |
| Attributes | (none) |
| Description | This element inserts the contents of a separate file into this portion of the specfile. The content of this element must point to a valid XML file containing valid dcapp data via an absolute path or a path relative to the current file. |

| | |
|---|---|
| Element | If |
| Parent | (any) |
| Children | True, False, (any) |
| Attributes | Operator, Value, Value1, Value2 |
| Description | This element applies the *Operator* (one of "eq", "ne", "gt", "lt", "ge", or "le") to *Value1* and *Value2* to evaluate a true or false condition. If no *Operator* is defined, then it simply tests *Value* to determine true or false. If the logic evaluates to true, then the sub-elements within the "True" element are processed, otherwise, the sub-elements within the "False" element are processed. If there is no "True" or "False" sub-element defined, the contents of this element are assumed to be contained within a virtual "True" element. |

| | |
|---|---|
| Element | True |
| Parent | If |

| | |
|---|---|
| Children | (any) |
| Attributes | (none) |
| Description | This element simply encloses sub-elements that are to be processed if the logic of the encompassing "If" element resolves to "true". |

| | |
|---|---|
| Element | False |
| Parent | If |
| Children | (any) |
| Attributes | (none) |
| Description | This element simply encloses sub-elements that are to be processed if the logic of the encompassing "If" element resolves to "false". |

| | |
|---|---|
| Element | Set |
| Parent | (any) |
| Children | (none) |
| Attributes | Variable, Operator, MinimumValue, MaximumValue |
| Description | This sets the value of *Variable* to a new value defined by the content of the element. The *Operator* is "=" by default, but may also be "+=" or "-=" if this element is to be used to increment or decrement *Variable* (usable only if *Variable* is a numeric type). *MinimumValue* and *MaximumValue* may optionally be set to bound the new numeric value. |

| | |
|---|---|
| Element | Animation |
| Parent | (any) |
| Children | Set, If |
| Attributes | Duration |
| Description | For each embedded "Set" element, this takes a snapshot of the current value and gradually sets it to the specified value over the course of the specified *Duration*. This is done linearly over each execution of dcapp until *Duration* is reached, at which point this element goes dormant until it is invoked again. |

## 4.3 Initialization Elements

These elements typically appear near the top of the dcapp specfile. They define the behavior of subsequent elements within the specfile.

### 4.3.1 Settings Elements

| | |
|---|---|
| Element | Constant |
| Parent | DCAPP |
| Children | (none) |
| Attributes | Name |
| Description | This allows a user to create a constant that can be accessed subsequently within the specfile. This is handy for setting values that |

| | |
|---|---|
| | are used frequently throughout the display. For instance, the user may set:<br>`<Constant Name="FontSize">24</Constant>`<br>The pre-processor will then replace all instances of "#FontSize" in the rest of the specfile with "24". |

| | |
|---|---|
| Element | Variable |
| Parent | DCAPP |
| Children | Type, InitialValue |
| Attributes | (none) |
| Description | This allows a user to create a variable that can be accessed subsequently within the specfile. The *Type* must be either "Float", "Integer", or "String". For instance, the user may set:<br>`<Variable Type="Integer">MyVar</Variable>`<br>Any subsequent elements may then use the associated value by specifying a value of "@MyVar". Note that if *InitialValue* is not specified, the default value is 0 for float and integer parameters and an empty string ("") for string parameters. |

| | |
|---|---|
| Element | Style |
| Parent | DCAPP |
| Children | (any) |
| Attributes | Name |
| Description | This allows a user to define a style, which defines attributes for any element that is used subsequently within the specfile. For instance, the user may set:<br>`<Style Name="mystyle">`<br>`  <String Size="28" Color="0 0 1"/>`<br>`</Style>`<br>Then, a subsequent "String" element that uses "mystyle" (`<String style="mystyle"...`) will be blue and use a font size of 28 by default. Note that multiple elements may be defined within a single "Style" element. |

| | |
|---|---|
| Element | Defaults |
| Parent | DCAPP |
| Children | (any) |
| Attributes | (none) |
| Description | This allows a user to define default attributes for any element that is used subsequently within the specfile. For instance, the user may set:<br>`<Defaults>`<br>`  <Rectangle LineWidth="2" LineColor="1 0 0"/>`<br>`</Defaults>`<br>Then, all subsequent "Rectangle" elements will be rendered with a red line that is 2 pixels thick by default. Note that multiple elements may be defined within a single "Defaults" element. |

### 4.3.2 Input/Output Elements

| | |
|---|---|
| Element | TrickIo |
| Parent | DCAPP |
| Children | FromTrick, ToTrick |
| Attributes | Host, Port, DataRate, DisconnectAction |
| Description | This construct specifies communication between dcapp and the Trick variable server. *Host* specifies the hostname upon which the Trick simulation is executing. If not specified, the default value is the hostname of the machine upon which dcapp is executing. *Port* specifies the port over which communication with the Trick variable server takes place. If not specified, the default value is 7000. *DataRate* specifies the data rate (in seconds) at which Trick will attempt to communicate with dcapp. If not specified, the default value is 1 second. *DisconnectAction* defines the action that dcapp takes if it loses connection with Trick. Options are "Terminate" or "Reconnect", with "Terminate" being the default action if none is specified. Note that the values for *Host* and *Port* may be overridden by the command-line arguments outlined in section 3. |

| | |
|---|---|
| Element | FromTrick |
| Parent | TrickIo |
| Children | TrickVariable |
| Attributes | (none) |
| Description | This contains a list of the "TrickVariable" elements that are used to over-write dcapp data with data from the attached Trick simulation. |

| | |
|---|---|
| Element | ToTrick |
| Parent | TrickIo |
| Children | TrickVariable |
| Attributes | (none) |
| Description | This contains a list of the "TrickVariable" elements that are used to over-write Trick simulation data with data from dcapp. |

| | |
|---|---|
| Element | TrickVariable |
| Parent | FromTrick, ToTrick |
| Children | (none) |
| Attributes | Name, Units |
| Description | This element attaches a dcapp "Variable" to the variable in the attached Trick simulation defined by *Name*. The user may optionally define the *Units* of the data within dcapp, which the Trick variable server will use to convert the data, if necessary. The *Units* must be a unit string recognizable by Trick. For instance:<br>`<TrickVariable Name="trickobj.var">MyVar</TrickVariable>` |

| | |
|---|---|
| Element | EdgeIo |
| Parent | DCAPP |
| Children | FromEdge, ToEdge |
| Attributes | Host, Port, DataRate |
| Description | This construct specifies communication between dcapp and EDGE via EDGE's remote commanding server server. *Host* specifies the hostname upon which EDGE is executing. If not specified, the default value is the hostname of the machine upon which dcapp is executing. *Port* specifies the port over which communication with EDGE takes place. If not specified, the default value is 5451. *DataRate* specifies the data rate (in seconds) at which EDGE will be polled by dcapp. If not specified, the default value is 1 second. |

| | |
|---|---|
| Element | FromEdge |
| Parent | EdgeIo |
| Children | EdgeVariable |
| Attributes | (none) |
| Description | This contains a list of the "EdgeVariable" elements that are used to over-write dcapp data with data from the attached EDGE instance. For instance:<br>`<EdgeVariable RcsCommand="doug.node Light set -lit_int">LightCmd</EdgeVariable>` |

| | |
|---|---|
| Element | ToEdge |
| Parent | EdgeIo |
| Children | EdgeVariable |
| Attributes | (none) |
| Description | This contains a list of the "EdgeVariable" elements that are used to over-write EDGE data with data from dcapp. |

| | |
|---|---|
| Element | EdgeVariable |
| Parent | FromEdge, ToEdge |
| Children | (none) |
| Attributes | RcsCommand |
| Description | This element attaches a dcapp "Variable" to the variable in the attached EDGE instance defined by *RcsCommand*. |

| | |
|---|---|
| Element | CAN |
| Parent | DCAPP |
| Children | (none) |
| Attributes | Network, ButtonID, ControlID |
| Description | This element assigns bezel keys to data associated with a CAN bus based upon *Network*, *ButtonID*, and *ControlID* of the unit associated with this instance of dcapp. The bezel keys are processed via the "Button" and/or "BezelEvent" elements. |

| | |
|---|---|
| Element | UEI |
| Parent | DCAPP |
| Children | (none) |
| Attributes | Host, Port, BezelID |
| Description | This element assigns bezel keys to data associated with a UEI controller based upon the *Host* and *Port* of the UEI and the *BezelID* of the unit associated with this instance of dcapp. The bezel keys are processed via the "Button" and/or "BezelEvent" elements. |

| | |
|---|---|
| Element | Hagstrom |
| Parent | DCAPP |
| Children | (none) |
| Attributes | SerialNumber |
| Description | This element assigns bezel keys to data associated with a Hagstrom device. The user may specify a *SerialNumber* of the Hagstrom device it wants to associate with in case multiple devices are connected to the computer. If *SerialNumber* isn't specified, then dcapp will attach to the first Hagstrom device it discovers. The bezel keys are processed via the "Button" and/or "BezelEvent" elements. |

### 4.3.3 Logic Element

| | |
|---|---|
| Element | DisplayLogic |
| Parent | DCAPP |
| Children | (none) |
| Attributes | (none) |
| Description | The content of this element specifies a shared object file to be linked into dcapp at execution time. See section 5.3 for more information about the format and content of this file. |

## 4.4 Display Setup

| | |
|---|---|
| Element | Window |
| Parent | DCAPP |
| Children | Panel |
| Attributes | X, Y, Width, Height, FullScreen, ActiveDisplay, ForceUpdate |
| Description | This defines the position (*X* and *Y*) and size (*Width* and *Height*) of the window containing the dcapp displays. If *FullScreen* is set to "true", "yes", or "on", the window will be rendered full screen regardless of *X*, *Y*, *Width*, and *Height* settings. The *ActiveDisplay* attribute allows the user to assign a variable to determine which display is active at any given time. If the value of this variable corresponds to the *DisplayIndex* of a given panel (see below), then that panel becomes the active display. By default, dcapp only updates when it senses an event |

| | |
|---|---|
| | (a mouse event, input data change, etc.), but the user may set *ForceUpdate* to specify an interval, in seconds, after which dcapp will automatically update. |

| | |
|---|---|
| Element | Panel |
| Parent | Window |
| Children | (display primitives) |
| Attributes | DisplayIndex, BackgroundColor, VirtualWidth, VirtualHeight |
| Description | This contains all of the display primitives for a given display panel. The *DisplayIndex* attribute is used to define when this display is the active display. *BackgroundColor* specifies the background color for the panel. See section 5.1 for information on specifying color. If not specified, the default color is black ("0 0 0"). *VirtualWidth* and *VirtualHeight* define the user-specified geometry of the display panel, which is used to render the position and size of the display primitives. If not specified, the default geometry is 100x100 units. |

## 4.5 Display Primitives

The display primitives are the building blocks that define how the individual display panels look, feel, and react to user input. They are grouped into two primary classifications: visual primitives, which are primitives that render data to the screen, and event primitives, which are primitives that handle user input.

### 4.5.1 Visual Primitives

| | |
|---|---|
| Element | Container |
| Parent | Panel, Container, Button, Active, Inactive, On, Transition, Off |
| Children | (display primitives) |
| Attributes | X, Y, OriginX, OriginY, Width, Height, HorizontalAlign, VerticalAlign, VirtualWidth, VirtualHeight, Rotate |
| Description | This redefines the coordinate frame for subsequent primitives by allowing the user to define a box of size *Width* by *Height* at position *X*, *Y* (with respect to *OriginX* and *OriginY*), and aligned by *HorizontalAlign* and *VerticalAlign*, within the current coordinate frame. The new coordinate frame can also be rotated by *Rotate* degrees from the current coordinate frame, and the new coordinate frame uses *VirtualWidth* and *VirtualHeight* to define the width and height of subsequent elements within the new frame. |

| | |
|---|---|
| Element | Line |
| Parent | Panel, Container, Button, Active, Inactive, On, Transition, Off |
| Children | Vertex |
| Attributes | LineWidth, Color |
| Description | This attaches the enclosed "Vertex" primitives to form a single, continuous line with the specified *LineWidth* and *Color*. |

| | |
|---|---|
| Element | Polygon |
| Parent | Panel, Container, Button, Active, Inactive, On, Transition, Off |
| Children | Vertex |
| Attributes | FillColor, LineColor, LineWidth |
| Description | This attaches the enclosed "Vertex" primitives to form a polygon. The polygon is filled with *FillColor* and outlined with a line of color *LineColor* and a width of *LineWidth*. If *FillColor* is not set, then the polygon is not filled. Likewise, if *LineColor* and *LineWidth* are not set, then the polygon is not outlined. Note that this primitive works well for convex polygons, but the behavior for polygons with concave vertices is undefined. |

| | |
|---|---|
| Element | Vertex |
| Parent | Panel, Container, Button, Active, Inactive, On, Transition, Off |
| Children | (none) |
| Attributes | X, Y, OriginX, OriginY |
| Description | This defines the *X* and *Y* coordinates (with respect to *OriginX* and *OriginY*) of a vertex within a "Line" or "Polygon" primitive. |

| | |
|---|---|
| Element | Rectangle |
| Parent | Panel, Container, Button, Active, Inactive, On, Transition, Off |
| Children | (none) |
| Attributes | X, Y, OriginX, OriginY, Width, Height, HorizontalAlign, VerticalAlign, Rotate, FillColor, LineColor, LineWidth |
| Description | This renders a rectangle based upon the location, origin, size, alignment, and orientation specified by the user. The rectangle is filled with *FillColor* and outlined with a line of color *LineColor* and a width of *LineWidth*. If *FillColor* is not set, then the rectangle is not filled. Likewise, if *LineColor* and *LineWidth* are not set, then the rectangle is not outlined. |

| | |
|---|---|
| Element | Circle |
| Parent | Panel, Container, Button, Active, Inactive, On, Transition, Off |
| Children | (none) |
| Attributes | X, Y, OriginX, OriginY, HorizontalAlign, VerticalAlign, Radius, Segments, FillColor, LineColor, LineWidth |
| Description | This renders a circle based upon the location, origin, radius, and alignment specified by the user. The user may also specify the number of straight-line segments used to render the circle via *Segments* (default is 80). The circle is filled with *FillColor* and outlined with a line of color *LineColor* and a width of *LineWidth*. If *FillColor* is not set, then the circle is not filled. Likewise, if *LineColor* and *LineWidth* are not set, then the circle is not outlined. |

| | |
|---|---|
| Element | String |
| Parent | Panel, Container, Button, Active, Inactive, On, Transition, Off |
| Children | (none) |
| Attributes | X, Y, OriginX, OriginY, Rotate, Size, HorizontalAlign, VerticalAlign, Color, BackgroundColor, ShadowOffset, Font, Face, ForceMono |
| Description | This renders a character string based on the location, origin, size, alignment, and rotation specified by the user. The user may also specify the *Font* and *Face*. Note that *Font* must point to a valid FreeType-accessible font file (most modern font files are FreeType-accessible) via an absolute path or a path relative to the current file. Different font files offer different options for *Face*, but typical options include "Bold", "Italic", etc., and if *Face* is not specified, the default face for the font is used. The *ForceMono* optional flag accommodates three possible values: "Numeric", "Alphanumeric", or "All". This allows the user to render some or all of the characters in a variable-width font as fixed width. The user may specify font *Color* with an optional *BackgroundColor*, and *ShadowOffset* allows the user to specify the offset of a shadow to be rendered behind the font (no shadow is rendered if *ShadowOffset* is not set). The content of this element is the string to be rendered, and it may contain static text, variable text, or both. dcapp variables may be accessed using the ampersand: "@MyVar", as well as an optional C-format specifier contained within parentheses: "@MyVar(%.2f)". |

| | |
|---|---|
| Element | Image |
| Parent | Panel, Container, Button, Active, Inactive, On, Transition, Off |
| Children | (none) |
| Attributes | X, Y, OriginX, OriginY, Width, Height, HorizontalAlign, VerticalAlign, Rotate |
| Description | This renders an image based on the location, origin, size, alignment, and rotation specified by the user. The content of this element must point to a graphical file in a format usable by dcapp (see section 5.3) via an absolute path or a path relative to the current file. |

| | |
|---|---|
| Element | PixelStream |
| Parent | Panel, Container, Button, Active, Inactive, On, Transition, Off |
| Children | (none) |
| Attributes | X, Y, OriginX, OriginY, Width, Height, HorizontalAlign, VerticalAlign, Rotate, Protocol, Host, Port, Path, SharedMemoryKey, File, TestPattern |
| Description | This renders a dynamic image based on the location, origin, size, alignment, and rotation specified by the user. The dynamic image must be generated by a separate package running a compatible PixelStream writer. *Protocol* is either "File", "MJPEG", "TCP", or "VSM" ("File" is typically best if the writer is on the same computer as dcapp, and "MJPEG" is typically best if the writer is on a remote computer. |

| | |
|---|---|
| | "VSM" is designed specifically for use with the video stream manager (VSM) application). Note that "MJPEG" and "VSM" only work *IF* the user has installed libjpeg (or libjpeg-turbo) and curl on their computer. For "File", the user must specify a *SharedMemoryKey*, which provides shared memory space for hand-shaking, and a *File*, which provides disk space containing image RGB information, that matches those settings for the writer. For "MJPEG" or "TCP", the user must provide the name of the remote *Host* (default is "localhost") and the *Port* number (default for "MJPEG" is "80") used by the PixelStream writer. Also for "MJPEG" the user may specify a *Path* in addition to *Host* and *Port*. "VSM" requires the user to specify a *Host* and *Port* for the VSM application as well as *Camera*, which specifies the name of the camera to be requested from VSM (this is typically a variable name). A user may also specify a *Path* for VSM that overrides the default path passed back by VSM. If the user specifies a *TestPattern* file in a graphical format usable by dcapp (see section 5.3) via an absolute path or a path relative to the current file, then the image associated with this file will be displayed if the PixelStream is not currently connected to a PixelStream source. |

| | |
|---|---|
| Element | ADI |
| Parent | Panel, Container, Button, Active, Inactive, On, Transition, Off |
| Children | (none) |
| Attributes | X, Y, OriginX, OriginY, Width, Height, HorizontalAlign, VerticalAlign, OuterRadius, BallRadius, ChevronWidth, ChevronHeight, BallFile, CoverFile, Roll, Pitch, Yaw, RollError, PitchError, YawError |
| Description | This renders an attitude direction indicator (ADI), or 8-ball, used in flying vehicles to show attitude (pitch/yaw/roll) information. It is rendered with the location, origin, size, and alignment information provided by the user. The user may also customize *OuterRadius*, *BallRadius*, *ChevronWidth*, and *ChevronHeight*. *BallFile* and *CoverFile* allow the user to specify an image to overlay on the 8-ball and an image for the instrument face. *Roll*, *Pitch*, *Yaw*, *RollError*, *PitchError*, and *YawError* point to variables used to drive the information on the ADI. |

### 4.5.2 Event Primitives

| | |
|---|---|
| Element | Button |
| Parent | Panel, Container |
| Children | Active, Inactive, On, Transition, Off, OnPress, OnRelease, (display primitives) |
| Attributes | X, Y, OriginX, OriginY, Width, Height, HorizontalAlign, VerticalAlign, Rotate, Type, Key, KeyASCII, BezelKey, Variable, On, Off, SwitchVariable, SwitchOn, SwitchOff, IndicatorVariable, IndicatorOn, ActiveVariable, ActiveOn |

| Description | |
|---|---|

| Element | Active |
|---|---|
| Parent | Button |
| Children | (display primitives) |
| Attributes | (none) |
| Description | This contains a list of primitives to be rendered when a Button is in the “active” state (when ActiveVariable is set to the ActiveOn value). |

| Element | Inactive |
|---|---|
| Parent | Button |
| Children | (display primitives) |
| Attributes | (none) |
| Description | This contains a list of primitives to be rendered when a Button is in the “inactive” state (when ActiveVariable is not set to the ActiveOn value). |

| Element | On |
|---|---|
| Parent | Button |
| Children | (display primitives) |
| Attributes | (none) |
| Description | This contains a list of primitives to be rendered when a Button is in the “on” state (when IndicatorVariable is set to the IndicatorOn value). |

| Element | Transition |
|---|---|
| Parent | Button |
| Children | (display primitives) |
| Attributes | (none) |
| Description | This contains a list of primitives to be rendered when a Button is in the “transition” state (when IndicatorVariable and SwitchVariable are in different states). |

| Element | Off |
|---|---|
| Parent | Button |
| Children | (display primitives) |
| Attributes | (none) |
| Description | This contains a list of primitives to be rendered when a Button is in the “off” state (when IndicatorVariable is not set to the IndicatorOn value). |

| Element | MouseMotion |
|---|---|
| Parent | Panel, Container |
| Children | (none) |
| Attributes | XVariable, YVariable |
| Description | This element provides the user with the current X and Y position of |

| | |
|---|---|
| | the mouse within the context of the Panel or Container within which it exists. Note that if this element is active, it will sense any mouse motion as an event, which will trigger a display update, which can be computationally expensive. |

| | |
|---|---|
| Element | MouseEvent |
| Parent | Panel, Container |
| Children | OnPress, OnRelease, Set, If, Animation |
| Attributes | X, Y, OriginX, OriginY, Width, Height, HorizontalAlign, VerticalAlign |
| Description | This element sets up a listener to react when the mouse is pressed or released in a bounding volume specified by this element's attributes. The listener then executes the elements contained within its "OnPress" and "OnRelease" elements. If there is no "OnPress" or "OnRelease" sub-element defined, the contents of this element are assumed to be contained within a virtual "OnPress" element. |

| | |
|---|---|
| Element | KeyboardEvent |
| Parent | Panel, Container |
| Children | OnPress, OnRelease, Set, If, Animation |
| Attributes | Key, KeyASCII |
| Description | This element sets up a listener to react when a specified key on a keyboard is pressed or released. The key can be specified either with *Key* (for instance "a", "b", "Q", "3", "$", etc.) or with *KeyASCII* if the user wishes to specify a key that is not easily specifiable in an XML file (for instance, "8" to represent the "backspace" key). The listener then executes the elements contained within its "OnPress" and "OnRelease" elements. If there is no "OnPress" or "OnRelease" sub-element defined, the contents of this element are assumed to be contained within a virtual "OnPress" element. |

| | |
|---|---|
| Element | BezelEvent |
| Parent | Panel, Container |
| Children | OnPress, OnRelease, Set, If, Animation |
| Attributes | Key |
| Description | This element sets up a listener to react when a specified bezel *Key* is pressed or released. The listener then executes the elements contained within its "OnPress" and "OnRelease" elements. If there is no "OnPress" or "OnRelease" sub-element defined, the contents of this element are assumed to be contained within a virtual "OnPress" element. |

| | |
|---|---|
| Element | OnPress |
| Parent | Button, MouseEvent, KeyboardEvent, BezelEvent |
| Children | Set, If, Animation |
| Attributes | (none) |

| | |
|---|---|
| Description | This element defines a list of actions to take if the parent element senses a “press” condition. |

| | |
|---|---|
| Element | OnRelease |
| Parent | Button, MouseEvent, KeyboardEvent, BezelEvent |
| Children | Set, If, Animation |
| Attributes | (none) |
| Description | This element defines a list of actions to take if the parent element senses a “release” condition. |

# 5.0 Technical Details

## 5.1 Color Format Specification

When specifying color formats for any dcapp display elements, the following format must be used:

```
red_level green_level blue_level alpha_level
```

where each level is expressed as a number between 0 (full off) and 1 (full on). Note that if `alpha_level` isn’t specified, dcapp assumes a value of 1 (fully opaque). Examples may include: black specified as “0 0 0”, white specified as “1 1 1”, blue specified as “0 0 1”, grey specified as “0.5 0.5 0.5”, etc.

## 5.2 Origin Specification

The origin attributes (*OriginX* and *OriginY*) specify which side of the current region to establish as the base for *X* and *Y*. For instance, if a user wants a primitive 10 units from the left, they would set *OriginX* to “Left” and *X* to “10”. Likewise, if they want a primitive 10 units from the right, they would set *OriginX* to “Right” and *X* to “10”. Options for *OriginX* are “Left” and “Right”. Options for *OriginY* are “Bottom” and “Top”. If a user doesn’t specify an origin, “Left” and “Bottom” are used as default values.

## 5.3 Alignment Specification

The alignment attributes (*HorizontalAlign* and *VerticalAlign*) specify the direction in which a primitive is rendered with respect to the specified *X* and *Y* values. Options for *HorizontalAlign* are “Left”, “Center”, and “Right”. Options for *VerticalAlign* are “Bottom”, “Middle”, and “Top”. If a user doesn’t specify alignment, “Left” and “Bottom” are used as default values.

## 5.4 Graphic File Formats

dcapp can currently handle graphic files in TARGA (.tga) and bitmap (.bmp) formats. TARGA files should be saved uncompressed with a "bottom left" origin. Bitmap files should be saved in 24-bit format, although files saved in other valid bitmap formats may work. dcapp also handles JPEG (.jpg or .jpeg) formats *IF* the user has installed libjpeg (or libjpeg-turbo) on their computer.

## 5.5 Display Logic File

While dcapp sets the value of its variables automatically based upon user specifications provided within the XML specfile, it also provides the capability for a user to modify those variables using custom C++ software. This is useful if the user wants the display to operate in non-standard or complex ways. To facilitate this, dcapp provides a command-line tool called dcapp_genheader, which is activated with the following syntax:

```
dcapp_genheader file.xml [output_file]
```

where `file.xml` is a full or partial path to a valid dcapp specfile and `output_file` is the name of the C++ header file to be created (the default value is "dcapp.h"). This tool mines the specfile, finds all of the dcapp variables, and provides them to the user as pointers (double *, int *, or std::string *, depending upon how they're defined in the specfile) in `output_file`.

From there, the user may create one or more C++ files to manipulate these variables. The user must #include `output_file` in the C++ files and include C++ logic within one of these three "void" routines:

- DisplayInit
- DisplayLogic
- DisplayClose

Note that these routines do not accept arguments, and the user need only include the ones that are needed for their logic. DisplayInit is executed once at dcapp startup and DisplayClose is executed once at dcapp shutdown. DisplayLogic is executed whenever dcapp senses that a display should be updated, which includes a keyboard or mouse event, a variable update, etc.

Once the user creates the logic files, they should be compiled into a shared object. Doing so can differ per operating system but typically looks something like:

```
c++ -fPIC -shared mylogic.cc -o mylogic.so
```

From there, the user instructs dcapp to use this customized logic by using the DisplayLogic element (see section 4.3.3) within the specfile.

## 5.6 Element Values

### 5.6.1 Constants

### 5.6.2 Variables

### 5.6.3 Environment Variables

The user may access the value of any available environment variable from within the specfile by prepending the name of the environment variable with a dollar sign ("$"). For instance, the following String element shows the value of the USER environment variable:

```
<String>my user name is $USER</String>
```

Note that dcapp makes the following environment variables available to the user:

- dcappOSTYPE
- dcappOSSPEC
- dcappOBJDIR
- dcappBINDIR

These values are set to the return values of the dcapp.app/Contents/dcapp-config script with the following arguments respectively: --ostype, --osspec, --objdir, and --bindir. Note that using the above environment variables in the specfile in conjunction with the corresponding return values from the dcapp.app/Contents/dcapp-config script in build files are a great way of ensuring that the specfile and build files are pointing to the same files and directories.

Other environment variables provided by dcapp include:

- dcappDisplayHome
- dcappVersion

The dcappDisplayHome variable is set to the directory containing the specfile. It is useful for specifying a path to a Font, Image, etc., with respect to the specfile instead of to the current file, which may change if Include elements are used. The dcappVersion provides a way to test the current version of dcapp from within the specfile in the event that a display needs to run with two or more versions of dcapp that may or may not be fully compatible from a programming interface perspective.

Besides the above variables, dcapp makes the following environment variables available to the user from within the specfile by setting them if they haven't already been set:

- USER

- LOGNAME
- HOME
- OSTYPE
- MACHTYPE
- HOST

## 6.0 Release Information

While dcapp strives to maintain full compatibility between releases, there are often changes from release to release that require a user to make modifications to their specfiles and/or logic files. Below is a list of such items associated with each release:

- 1.0:
  - The pre-release versions of dcapp provided the user with "String" variables of type "char *" for use in their logic files. With this release, those variables are now of type "std::string", so some re-writing of logic files may be needed since calls like strlen, strcmp, etc. do not work with variables of type std::string.
- 1.1:
  - Prior to dcapp 1.1, a user could use negative values for X and Y to render primitives from the right or top of the current region, which was often confusing and non-intuitive. This is no longer the case. This release provides the explicit attributes OriginX and OriginY to achieve this instead.