EDML
and
PJML, VIML, CAML
Reference Manual

XML Languages for the ED-Projector System, including the Projector, the Vision Interface, and the Camera Control Modules

version 1.01
June 19, 2002

Rick Kjeldsen
Anthony Levas
Gopal Pingali
Claudio Pinhanez

Copyright 2002, IBM Co.
Syntax Summary

Basic Syntax
All commands, methods, and objects are written in standard XML format. To describe the statements recognized by the EDML system modules, we use the following syntax symbols:

```xml
<a y="*" />
```

means that the attribute ‘y’ of the component ‘a’ accepts any string as its value.

```xml
<a y="#" />
```

means that the attribute ‘y’ of the component ‘a’ accepts any integer number as its value.

```xml
<a y="www|zzz|*" />
```

means that the strings ‘www’ and ‘zzz’ are particularly recognized values of the attribute ‘y’ of the component ‘a’, but any other string is also accepted.

```xml
<a y="[www]|zzz|*" />
```

means that the strings ‘www’ and ‘zzz’ are particularly recognized values of the attribute ‘y’ of the component ‘a’, but any other string is also accepted, and ‘www’ is the default value.

```xml
<a y="[www]|zzz|#" />
```

means that the strings or numbers ‘www’ and ‘zzz’ are particularly recognized values of the attribute ‘y’ of the component ‘a’, but any other integer number is also accepted, and ‘www’ is the default value.

```xml
<a * />
```

means that the statement can contain any attribute or element of ‘a’.

```xml
<xxx>
    * <a />
</xxx>
```

means that multiple ‘a’ components can be listed inside a single ‘xxx’ statement.

```xml
<xxx>
    + <a />
</xxx>
```

means that just one component ‘a’ can appear in a single ‘xxx’ statement.
Syntax Summary

```
<xxx>
  | <a />
  <b />
</xxx>
```

means that either the component ‘a’ or the component ‘b’ (one of the two) can appear in a single ‘xxx’ statement.

```
<xxx>
  * <a />
  * <b />
</xxx>
```

means that multiple ‘a’ and ‘b’ components can be listed inside a single ‘xxx’ statement. In the case of methods, parallelization of the components may occur during execution.
Ex: In `<xxx> <a /> <b /> </xxx>` ‘a’ and ‘b’ may be performed in parallel;
while in `<xxx> <a /> </xxx> <xxx> <b /> </xxx>` ‘a’ is followed by ‘b’.

**Method Syntax**

Methods are always specified as the topmost component in the XML command, immediately following the language specification.

```
<__ml * >
  <method_x * >
    <a * />
  </method_x>
</__ml>
```

means that ‘method_x’ is applied to the ‘a’ object.

Methods are always applied to the most specific object, unless otherwise stated in the method definition.

```
<method_x>
  <a * >
    <b * />
  </a>
</method_x>
```

means that ‘method_x’ is applied only to the ‘b’ component of ‘a’, unless otherwise stated in the method definition.

Methods are applied to all objects of the same level of specificity, unless otherwise stated in the method definition.

```
<method_x>
  <a * >
    <b * />
    <c * />
  </a>
</method_x>
```

means that ‘method_x’ is applied both to ‘b’ and ‘c’ components of ‘a’, unless otherwise stated in the method definition.
# Table of Contents

1 OVERVIEW.............................................................................................................................................. 7
  1.1 Architecture Overview .............................................................................................................. 7
  1.2 EDML Modules ......................................................................................................................... 7
  1.3 Object Creation and Persistency ............................................................................................. 9
  1.4 Object Activation ..................................................................................................................... 9
  1.5 Name Spaces .......................................................................................................................... 9

2 BASIC COMMANDS, OBJECTS, METHODS, AND EVENTS FOR EDML LANGUAGES ........................................................................................................................... 10
  2.1 Application Commands ........................................................................................................... 10
  2.2 Application Objects ................................................................................................................ 10
    2.2.1 EDapplicationInfo ........................................................................................................... 10
  2.3 Application Methods .............................................................................................................. 11
    2.3.1 open .................................................................................................................................... 11
    2.3.2 close ................................................................................................................................... 11
    2.3.3 set ...................................................................................................................................... 12
    2.3.4 get ...................................................................................................................................... 12
  2.4 Other Objects .......................................................................................................................... 13
    2.4.1 EDimage ........................................................................................................................... 13
    2.4.2 EDdesktop ......................................................................................................................... 13
    2.4.3 EDcamera ........................................................................................................................ 14
    2.4.4 EDmovie .......................................................................................................................... 14
  2.5 Return Messages ..................................................................................................................... 15
  2.6 Generated Events .................................................................................................................... 16

3 PJML: COMMANDS, OBJECTS, METHODS, AND EVENTS FOR THE PROJECTOR/MIRROR SYSTEM ................................................................................................................................. 17
  3.1 Commands ............................................................................................................................... 17
  3.2 Objects ..................................................................................................................................... 17
    3.2.1 PJsystem ............................................................................................................................ 17
    3.2.2 PJmirror ............................................................................................................................ 18
    3.2.3 PJprojector ......................................................................................................................... 18
    3.2.4 PJrendering ........................................................................................................................ 18
    3.2.5 PJsurface .......................................................................................................................... 18
    3.2.6 PJmirrorState ..................................................................................................................... 19
    3.2.7 PJprojectorState ................................................................................................................ 19
    3.2.8 PJprojCorrection ................................................................................................................ 19
    3.2.9 PJposition3D ....................................................................................................................... 20
    3.2.10 PJhomography .................................................................................................................. 21
    3.2.11 PJdisplay .......................................................................................................................... 21
    3.2.12 PJimage ............................................................................................................................ 22
    3.2.13 PJstream ........................................................................................................................... 22
  3.3 Methods ..................................................................................................................................... 23
Table of Contents

3.3.1 use .................................................................................................................................23
3.3.2 release .................................................................................................................................24
3.3.3 set .........................................................................................................................................24
3.3.4 get .........................................................................................................................................25
3.3.5 activate .................................................................................................................................26
3.3.6 deactivate ............................................................................................................................27
3.3.7 sample ...................................................................................................................................27
3.4 Return Messages ....................................................................................................................28

4 VIML: COMMANDS, OBJECTS, METHODS, AND EVENTS FOR THE VISION INTERFACE .................................................................................................................................29
4.1 Commands ..............................................................................................................................29
4.2 Objects ....................................................................................................................................29
  4.2.1 VIsystem .............................................................................................................................30
  4.2.2 VIsurface ..........................................................................................................................30
  4.2.3 VIconfiguration ..................................................................................................................30
  4.2.4 VIButton ..........................................................................................................................31
  4.2.5 VIslider ...............................................................................................................................31
  4.2.6 VItrackArea ........................................................................................................................32
  4.2.7 VIgestureArea ...................................................................................................................32
  4.2.8 VIwidget ...........................................................................................................................33
4.3 Methods ..................................................................................................................................33
  4.3.1 use .......................................................................................................................................33
  4.3.2 release ..................................................................................................................................34
  4.3.3 set .........................................................................................................................................35
  4.3.4 get .........................................................................................................................................36
  4.3.5 activate ..................................................................................................................................36
  4.3.6 deactivate .............................................................................................................................37
4.4 Return Messages ....................................................................................................................38
4.5 Generated Events ...................................................................................................................38
  4.5.1 VItementTouch ......................................................................................................................39
  4.5.2 VItementValueChange .........................................................................................................39
  4.5.3 VItementPositionChange .....................................................................................................40
  4.5.4 VItementGesture ................................................................................................................40
  4.5.5 VItementType ......................................................................................................................40

5 CAML: COMMANDS, OBJECTS, METHODS, AND EVENTS FOR THE CAMERA CONTROL .................................................................................................................................41
5.1 Commands ..............................................................................................................................41
5.2 Objects ....................................................................................................................................41
  5.2.1 CAsystem ............................................................................................................................41
  5.2.2 CAsurface ...........................................................................................................................42
  5.2.3 CApatternPosition .............................................................................................................42
5.3 Methods ..................................................................................................................................43
  5.3.1 use .......................................................................................................................................43
  5.3.2 release ..................................................................................................................................43
Table of Contents

5.3.3 set...........................................................................................................................................43
5.3.4 get .........................................................................................................................................44
5.3.5 activate ..............................................................................................................................45
5.3.6 deactivate ..........................................................................................................................45
5.3.7 getImage ............................................................................................................................45
5.3.8 findPattern ........................................................................................................................46
5.4 Return Messages ...................................................................................................................46

6 REFERENCE................................................................................................................................47
6.1 List of Return Values ..............................................................................................................47

REFERENCES................................................................................................................................48

INDEX...........................................................................................................................................49
1 Overview

This manual covers the XML communication languages used to control the ED-projector, the Vision Interface system, and the pan/tilt camera. The PJML languages is used to communicate with the projector, the VIML language is used with the Vision Interface system, and the CAML language with the camera. The whole group of languages is called EDML.

All commands follow XML syntax. The current implementation requires the issuing of these commands through the HTTP protocol, embedded in “POST” messages. To receive return information and events generated by the module, the application has to run their own XML/HTTP servers, able to accept “POST” messages from the different modules that compose the system described in this manual. The application has also to parse and interpret the return messages and the generated events, following the syntax described in the manual.

For convenience, a Java API is provided where Java methods issue the appropriate commands in XML to the modules. The Java package also includes an EDML server/parser/interpreter that listens to HTTP messages and translates them into Java events.

1.1 Architecture Overview

One of our primary objectives in our system architecture is to enable a specific user interaction to occur on any calibrated surface in a room without requiring the customization of this interaction for every surface. This leads to a clear separation of the abstract definition of an interaction from the actual surface upon which it occurs.

In our system, the projector is responsible for rendering the view, while the vision module provides the controller input events. The definition layer of Figure 1 shows the separation of the display and interaction areas that are responsible for the definition of the view and the interaction respectively. Display and interaction will often overlap as shown by the square overlapping buttons on the right of the display area. Provision is also made, however, for allowing interaction to occur in a region where nothing is displayed, as shown in the large rectangular region on the right of the interaction area.

Every interaction surface in a physical space is named and initially calibrated by the projection and vision subsystems yielding the transforms $H$ and $H_v$, respectively (see also [1]). $H$ corrects the projected image while $H_v$ provides the transformation to the camera image, for a specific surface as described in the prior sections. Figure 1demonstrates the flow of definitions for display images and interaction data (widget data) from the definition layer to the mapping layer. During run-time, the system software automatically maps (through the warping functions $H$ and $H_v$) the definition layer data into the projected image and the camera view associated with the active surface. The result is a projected image that generates application events when user gestures occur in the areas defined in the camera view, as shown in the integration layer of Figure 1.

1.2 EDML Modules

An application can issue EDML commands to many different systems. Each system is called an EDML module, and corresponds to a camera, a projector system, or a vision interface system. Each EDML module is identified by a name, the name of the machine it runs on, and the port number used to listen commands. The application only needs to know the machine name and the port number of an EDML module to communicate with it. In practice, this corresponds to the URL of the POST web server run by the module, used to listen to POST messages from the application.
The same application can use multiple EDML modules. Multiple applications and programs can use simultaneously multiple EDML modules, as long as they manage the usage of the resources to avoid conflicts.

To receive return messages and EDML events, an application has to run POST web servers. The URLs of these two web servers are sent to the EDML modules in the very first message to the module, an 'open' method on an 'EDapplicationInfo' object (see 2.2.1 and 2.3.1). All return messages in EDML are asynchronous, i.e., are received as POST messages in the return server, and not through the return mechanism of the original POST message.
1.3 Object Creation and Persistency

Objects in the EDML have to be declared explicitly (see the ‘use’ methods of each section) and cannot be deleted by XML messages. Actual deletion can only be performed through the GUI interface of each module. If objects do not exist when an application tries to use them, they are automatically created with default values (except if special attributes of the ‘use’ method are employed — see section 3.3.1, for instance). Objects that are explicitly named are easily accessed by the reference to the name.

All named objects (except ‘EDapplicationInfo’) in the EDML system are persistent, i.e., they are stored in the EDML modules and can be recalled at any time by an application. Examples of such objects are PJsurface, VIsurface, VIconfiguration, and CAsurface. Persistency is necessary in our system because location-specific and calibration parameters are used by these objects. Although persistent objects are stored, they still need to be accessed in the initialization phase of any application through the ‘use’ method.

1.4 Object Activation

All named objects (except “applications”) in EDML must be activated in order to become the object currently being used by the ED-projector system. Activation is accomplished through the ‘activate’ method.

1.5 Name Spaces

Any string of characters that is a valid filename in the local OS can be the name of an EDML object. In addition, objects can be grouped in “directories”, by using a directory like prefix, a string of characters followed by a slash “/” mark.

Examples of different valid names are: “table”, “browser/table”, “paint/table”, etc.
2 Basic Commands, Objects, Methods, and Events for EDML Languages

These are the commands, objects, methods, and events that are accepted by all EDML languages.

2.1 Application Commands

A EDML command is a list of basic EDML methods sent to EDML objects.

Syntax:

```
<edml key="*" application="*" >
  * <open * />
  * <close * />
  * <set * />
  * <get * />
</edml>
```

Attributes:

- **key**: string with a key to be matched and verified by the receiving module. If not present, or if it does not match the module key, the command is ignored.
- **application**: the name of the application to which the returns should be sent. If it is not present, the command is ignored.

Return values:

All returns of EDML commands are asynchronously sent to the URL of the application XML server, according to the definition of the application, at the time of completion of the command. Returns of commands listed in the same VIML command are sent one by one and marked with the id of the command.

2.2 Application Objects

2.2.1 EDapplicationInfo

An “EDapplicationInfo” object describes an application of the EDML system. This is the URL where events and notifications are sent.

Syntax:

```
<edapplicationInfo name="*" returnServer="[none]|*" eventServer="[none]|*"
  other="[]|*"
/>```

Attributes:

- **name**: name of the application.
- **returnServer**: the name of a URL where return values are sent. If ‘none’, it is assumed that the application does not run a web server and therefore cannot receive return messages generated by EDML modules.
- **eventServer**: the name of a URL where events are sent. If 'none', it is assumed that the application does not run a web server and therefore cannot receive events generated by EDML modules.
- **other**: placeholder for other attributes.


2.3 Application Methods

2.3.1 open

This method opens a connection to an EDML system. This command must be issued before any other command can be accepted from the application.

Syntax:

```xml
<open id="[unknown]|*" other="[]|*" />
```

Attributes:

- **id**: identification number of the call for this method, generated by the application program.
- **other**: placeholder for other method attributes.

Return values: see section 2.5.

Example:

The following command tries to open a connection from the application ‘xPlayer’ to an VIML module called ‘vi1’. The application runs on the machine called ‘mymachine’, and reserves port 1000 for the HTTP/XML servelet that handles return messages, and port 1010 for the HTTP/XML servelet processes events.

```xml
<edml key="EDML" application="xPlayer">
  <open id="127">
    <EDapplicationInfo name="xPlayer" returnServer="http://mymachine:1000" eventServer="http://mymachine:1010" />
  </open>
</edml>
```

If successful, a ‘POST’ XML message is going to be sent to the return server containing:

```xml
<edml vimlName="vi1">
  <return id="127" value="200" valueText="OK." method="open" />
</edml>
```

If an error occurs, for instance, the event server URL is invalid, a ‘POST’ XML message is going to be sent to the return server containing:

```xml
<edml vimlName="vi1">
  <return id="127" value="24" valueText="Invalid URL." method="open">
    <EDapplicationInfo name="xPlayer" returnServer="http://mymachine:1000" eventServer="http://mymachine:1010" />
  </return>
</edml>
```

2.3.2 close

This method closes a connection with the EDML system. After this command is issued, no commands sent to any module of the EDML system will be accepted. If ‘name’ is an empty string or contains an invalid name in any ‘application’ descriptor, that application is ignored. If ‘name="all"’ then all applications are closed.
Syntax:

```xml
<close id="[unknown]|*" other="[[]]|*" >
  * <EDapplicationInfo name="[[]]|all|*" />
</close>
```

**Attributes:**

- id: identification number of the call for this method, generated by the application program.
- other: placeholder for other method attributes.

**Return values:** see section 2.5.

### 2.3.3 set

This method sets the current state of the attributes of applications. Any attribute or element of “application” can be set this way.

Syntax:

```xml
<set id="[unknown]|*" other="[[]]|*" >
  * <EDapplicationInfo * />
</set>
```

**Attributes:**

- id: identification number of the call for this method, generated by the application program.
- other: placeholder for other method attributes.

**Return values:** see section 2.5.

### 2.3.4 get

This method gets the current state of the attributes of an application. If ‘name’ is an empty string or an invalid name in any ‘application’ descriptor, that application is ignored. If ‘name=”all”’ then information about all applications are returned. All other attributes and components of applications are ignored.

Syntax:

```xml
<get id="[unknown]|*" other="[[]]|*" >
  * <EDapplicationInfo name="[all]|*" />
</get>
```

**Attributes:**

- id: identification number of the call for this method, generated by the application program.
- other: placeholder for other method attributes.

**Return values:**

In addition to the return value, as described in section 2.5, ‘get’ returns the information about the requested applications inside the ‘return’ object, according to this syntax:

```xml
<edml * >
  <return * >
    * <EDapplicationInfo * />
  </return>
</edml>
```
2.4 Other Objects

2.4.1 EDimage
This object describes an image file available from a web server.

Syntax:

```xml
<EDimage src="[black.bmp]|*" left="[leftMost]|#" right="[rightMost]|#"
       top="[topMost]|#" bottom="[bottomMost]|#" other="[]|*" />
```

Attributes:

- `src`: the name, path, or URL of an image file.
- `left`: position, in relation to the image file coordinate system, of the left edge of the area where the image should be sampled from; “leftMost” corresponds to the leftmost position on the image file.
- `right`: position, in relation to the image file coordinate system, of the right edge of the area where the image should be sampled from; “rightMost” corresponds to the rightmost position on the image file.
- `top`: position, in relation to the image file coordinate system, of the top edge of the area where the image should be sampled from; “topMost” corresponds to the topmost position on the image file.
- `bottom`: position, in relation to the image file coordinate system, of the bottom edge of the area where the image should be sampled from; “bottomMost” corresponds to the bottommost position on the image file.
- `other`: placeholder for other attributes.

2.4.2 EDdesktop
This object describes a server that provides images from the desktop of a computer. Whenever sampled, it returns a copy of a rectangular portion of the desktop.

Syntax:

```xml
<EDdesktop src="[this]|*" left="[leftMost]|#" right="[rightMost]|#"
           top="[topMost]|#" bottom="[bottomMost]|#" other="[]|*" />
```

Attributes:

- `src`: URL of the desktop grab server; if “this”, grabs from the ED-projector's desktop.
- `left`: position, in relation to the desktop coordinate system, of the left edge of the area where the desktop image should be sampled from; “leftMost” corresponds to the leftmost position on the desktop.
- `right`: position, in relation to the desktop coordinate system, of the right edge of the area where the desktop image should be sampled from; “rightMost” corresponds to the rightmost position on the desktop.
- `top`: position, in relation to the desktop coordinate system, of the top edge of the area where the desktop image should be sampled from; “topMost” corresponds to the topmost position on the desktop.
- `bottom`: position, in relation to the desktop coordinate system, of the bottom edge of the area where the desktop image should be sampled from; “bottomMost” corresponds to the bottommost position on the desktop.
- `other`: placeholder for other attributes.
2.4.3 **EDcamera**

This object describes a server that provides images captured by a camera. Whenever sampled, it returns a copy of a rectangular portion of the camera image. Images are obtained using the “getImage” method described in section 5.3.5.

Syntax:

```xml
<EDcamera src="*" left="[leftMost]|#" right="[rightMost]|#"
   top="[topMost]|#" bottom="[bottomMost]|#" other="[]|*" />
```

Attributes:

- **src**: URL of the camera image server.
- **left**: position, in relation to the camera coordinate system, of the left edge of the area where the camera image should be sampled from; “leftMost” corresponds to the leftmost position on the camera.
- **right**: position, in relation to the camera coordinate system, of the right edge of the area where the camera image should be sampled from; “rightMost” corresponds to the rightmost position on the camera.
- **top**: position, in relation to the camera coordinate system, of the top edge of the area where the camera image should be sampled from; “topMost” corresponds to the topmost position on the camera.
- **bottom**: position, in relation to the camera coordinate system, of the bottom edge of the area where the camera image should be sampled from; “bottomMost” corresponds to the bottommost position on the camera.
- **other**: placeholder for other attributes.

2.4.4 **EDmovie**

This object defines a movie file. The source file can be either located in a web server or transferred directly as part of the message.

Syntax:

```xml
<EDmovie src="*" left="[leftMost]|#" right="[rightMost]|#"
   top="[topMost]|#" bottom="[bottomMost]|#" other="[]|*" />
```

Attributes:

- **src**: the name, path, or URL of a movie file.
- **left**: position, in relation to the image file coordinate system, of the left edge of the area where the image should be sampled from; “leftMost” corresponds to the leftmost position on the image file.
- **right**: position, in relation to the image file coordinate system, of the right edge of the area where the image should be sampled from; “rightMost” corresponds to the rightmost position on the image file.
- **top**: position, in relation to the image file coordinate system, of the top edge of the area where the image should be sampled from; “topMost” corresponds to the topmost position on the image file.
- **bottom**: position, in relation to the image file coordinate system, of the bottom edge of the area where the image should be sampled from; “bottomMost” corresponds to the bottommost position on the image file.
other: placeholder for other attributes.

2.5 Return Messages

Return messages describe the results of methods sent to objects. These messages are sent asynchronously to the URL of the application that sent the corresponding command. The header __ml has to be substituted by edml, pjml, viml, or caml.

Only one return object is sent for each method, even if the method is applied to multiple objects. If the application of all methods succeeds, then the return object simply declares the success of the whole method. If the methods fails in at least one object, then the parsing of the method is stopped and the error codes for that object are returned, together with the offending object.

Syntax:

```xml
<__ml __mlName="*" >
  <return value="#" valueText="*" methodId="[unknown]|*" method="*"
    other="[]|*" >
    * [method dependent stuff]
  </return>
</__ml>
```

Attributes:

- __mlName: name of the issuing module (i.e., 'pj', 'vi', 'ca').
- value: a number from a list of error numbers.
- valueText: a textual description of the error.
- methodId: identification number of the incoming method.
- method: name of the method that generated the return, composed by the concatenation of the statement headers corresponding to the return value (see example).
- other: placeholder for other return attributes.

Example:

If an application sends the following pseudo-command to the PJML module 'pj02':

```xml
<edml key="EDML" >
  <xxx id="1" > <a /> <b /> </xxx>
  <yyy id="2" > <c /> <d /> </yyy>
</edml>
```

If the application of the method ‘xxx’ to ‘a’ and ‘b’ succeeds, and the application of method ‘yyy’ to ‘c’ works but fails in ‘d’, a module returns a message such as:

```xml
<edml pjmlName="pj02" >
  <return id="1" value="200" valueText="OK" method="xxx" />
  <return id="2" value="0" valueText="General failure" method="yyy" >
    <d />
  </return>
</edml>
```

List of Return Values:

Generic return numbers are numbered between 0 and 999.
Success return values are numbered between 200 and 299.
200          OK.
210          OK, but application is already open.
0            General failure.
1            Syntax error.
10           Cannot create object: object already exists.
11           Cannot delete object: object does not exist.
20           Invalid parameter.
21           Invalid attribute.
23           Invalid object name.
24           Invalid URL.
28           Invalid application.
50           Parameter beyond range.

2.6 Generated Events

An ‘event’ object that progressively lists the objects that generated the event and the event itself.

Syntax:

```ml
<__ml __mlName="*" >
  <event id="*" >
    * [module dependent stuff]
  </event>
</__ml>
```

Attributes:

__Name: name of the issuing module (i.e., ‘pj’, ‘vi’, ‘ca’).

id: identification number generated by the module.
3 PJML: Commands, Objects, Methods, and Events for the Projector/Mirror System

This chapter describes the interface to the projector/mirror module.

3.1 Commands

A PJML command is a list of basic PJML methods sent to objects.

Syntax:

```xml
<pjml key="*" application="*">
  * <use * />
  * <release * />
  * <set * />
  * <get * />
  * <activate * />
  * <deactivate * />
  * <sample * />
</pjml>
```

Attributes:

- **key**: string with a key to be matched and verified by the receiving module. If not present, or if it does not match the module key, the command is ignored.
- **application**: the name of the EDapplicationInfo to which the returns should be sent. If not present, the command is ignored.

Return values:

All returns of PJML commands are asynchronously sent to the URL of the application XML server, at the time of completion of the command. Returns of commands listed in the same PJML command are sent one by one and marked with the id of the command.

3.2 Objects

The creation of projected surfaces is accomplished through the combination of two basic objects: PJsurface and PJdisplay. The PJsurface object refers to an actual planar surface in the real world. The object contains all the information needed to drive the mirror to the appropriate position and to correct the distortion caused by oblique projection. It also defines the four corners of a quadrilateral area on the surface to where the PJdisplay object will be mapped onto (see Figure 1, page 8).

While the PJsurface object defines where things are projected, the PJdisplay object determines what is projected. PJdisplays define the spatial configuration of images and streams of images (movies, desktop captures, etc). The position and size of these objects are given in respect to the PJdisplay coordinate system. The coordinate system of each PJdisplay object is defined by the application.

3.2.1 PJsystem

This object describes the basic projector/mirror system. Only one of these objects exists in each projector/system module.
Syntax:

```xml
<PJsystem other="[]|"*/
  + <PJmirror * />
  + <PJprojector * />
  + <PJrendering * />
</PJsystem>
```

**Attributes:**

other: placeholder for other attributes.

### 3.2.2 PJmirror

Object that defines mirror controls. Mostly used to activate or deactivate the mirror movement.

**Syntax:**

```xml
<PJmirror other="[]|"/> 
```

**Attributes:**

other: placeholder for other attributes.

### 3.2.3 PJprojector

Object that defines projector controls. Mostly used to activate or deactivate the projector control.

**Syntax:**

```xml
<PJprojector other="[]|"/> 
```

**Attributes:**

other: placeholder for other attributes.

### 3.2.4 PJrendering

Object that defines rendering controls. Mostly used to activate or deactivate the rendering of the display.

**Syntax:**

```xml
<PJrendering other="[]|"/> 
```

**Attributes:**

other: placeholder for other attributes.

### 3.2.5 PJsurface

This object describes the attributes of a PJsurface to where a PJdisplay can be mapped. It is composed of three objects, PJmirrorState, PJprojectorState, and PJprojCorrection, which describes the method and parameters used to correct for projection distortion.

**Syntax:**

```xml
<PJsurface name="[control]|" other="[]|"*/
  + <PJmirrorState * />
  + <PJprojectorState * />
  + <PJprojectorCorrection * />
</PJsurface>
```
Attributes:

name: name of a “PJsurface”; the “control” surface corresponds to a surface that does not store values, but that can be used to create PJsurfaces dynamically.

other: placeholder for other attributes.

3.2.6 PJmirrorState
This objects describes a state of the mirror of the ED-projector.

Syntax:

```xml
<PJmirrorState pan="[0]|#" tilt="[0]|#" panSpeed="[1000]|#"
   tiltSpeed="[1000]|#" accuracy="[standard]|maximum" other="[\]|*" />
```

Attributes:

pan: pan value of the mirror, in a scale from –1000 to 1000.
tilt: tilt value of the mirror, in a scale from –1000 to 1000.
panSpeed: how fast the mirror pans, in a scale from 0 to 1000.
tiltSpeed: how fast the mirror tilts, in a scale from 0 to 1000.
accuracy: determines whether the positioning of the mirror has to be very accurate or not; “maximum” accuracy may require a longer time to move the mirror.

other: placeholder for other attributes.

3.2.7 PJprojectorState
This objects describes a state of the projector of the ED-projector.

Syntax:

```xml
<PJprojectorState zoom="[0]|#" focus="[0]|#" zoomSpeed="[1000]|#"
   focusSpeed="[1000]|#" accuracy="[standard]|maximum" other="[\]|*" />
```

Attributes:

zoom: zoom value of the projector, in a scale from 0 to 1000.
focus: focus value of the mirror, in a scale from 0 to 1000.
zoomSpeed: how fast the projector zoom moves, in a scale from 0 to 1000.
focusSpeed: how fast the projector focus moves, in a scale from 0 to 1000.
accuracy: determines whether the positioning of the projector has to be very accurate or not; “maximum” accuracy may require a longer time to change the state of the projector.

other: placeholder for other attributes.

3.2.8 PJprojCorrection
This object determines the warping function used to correct oblique projection problems. It uses either a PJposition3D object or a PJhomography object to establish a projective reference coordinate system on the LCD plane of the projector. Then, it uses the attributes of this object to position and size a projective rectangle on that reference system.
Syntax:

```xml
<PJprojCorrection rotationZ="[0]\" scaleX="[100]\" scaleY="[100]\"
 translationX="[0]\" translationY="[0]\" other="[]\" />
```

Attributes:

- `rotationZ`: rotation of the rectangle in reference to the surface coordinate system, in 1/10\textsuperscript{th} degrees (i.e., it goes from -1800 to 1800); 0 corresponds to the Y-axis, and angles are measured CW.

- `scaleX`: scale in the direction of the X-axis of the surface coordinate system, from 0 to 9999; 100 corresponds to a surface with the default size in the case of PJposition3D, and to a surface that covers completely the four reference points in the case of PJhomography.

- `scaleY`: scale in the direction of the Y-axis of the surface coordinate system, from 0 to 9999; 100 corresponds to a surface with the default size in the case of PJposition3D, and to a surface that covers completely the four reference points in the case of PJhomography.

- `translationX`: translation in the direction of the X-axis of the surface coordinate system, from -9999 to 9999; 0 aligns the x-coordinate of the left corner of the rectangle to the center of the coordinate system; 1000 corresponds the default size in the case of PJposition3D, and to a surface that is exactly outside the reference points in the case of PJhomography.

- `translationY`: translation in the direction of the Y-axis of the surface coordinate system, from -9999 to 9999; 0 aligns the y-coordinate of the top corner of the rectangle to the center of the coordinate system; 1000 corresponds the default size in the case of PJposition3D, and to a surface that is exactly outside the reference points in the case of PJhomography.

- `other`: placeholder for other attributes.

### 3.2.9 PJposition3D

This object describes the fundamental 3D aspects of a square surface in 3D space. This basic surface defines the reference coordinate system of the PJSurface, and its bottom-left point corresponds to the origin (0,0) of the coordinate system, and the top-right point to the point (1000,1000).

Syntax:

```xml
<PJposition3D rotationX="[0]\" rotationY="[0]\"
 lensAngle="[100]\" other="[]\" />
```

Attributes:

- `rotationX`: rotation of the rectangle in reference to the X-axis of the 3D plane, in 1/10\textsuperscript{th} degrees (i.e., it goes from -1800 to 1800); 0 corresponds to the Y-axis, and angles are measured following the right-hand rule.

- `rotationY`: rotation of the rectangle in reference to the Y-axis of the 3D plane, in 1/10\textsuperscript{th} degrees (i.e., it goes from -1800 to 1800); 0 corresponds to the X-axis, and angles are measured following the right-hand rule.

- `lensAngle`: angle of view of the CG camera in 3D space, in 1/10\textsuperscript{th} degrees (i.e., it goes from 0 to 1800).

- `other`: placeholder for other attributes.
3.2.10 PJhomography

This object describes the four points in the LCD projector plane that define the reference coordinate system. The top-left point corresponds to the origin (-1000,-1000) of the coordinate system, and the bottom-right point to the point (1000,1000).

Syntax:

```xml
<PJhomography leftBottomX="[leftMost]|#" leftBottomY="[bottomMost]|#"
    leftTopX="[leftMost]|#" leftTopY="[topMost]|#"
    rightTopX="[rightMost]|#" rightTopY="[topMost]|#"
    rightBottomX="[rightMost]|#" rightBottomY="[bottomMost]|#"
    other="[]|*" />
```

Attributes:

- `leftBottomX`, `leftBottomY`, `leftTopX`, `leftTopY`, `rightTopX`, `rightTopY`, `rightBottomX`, `rightBottomY`: LCD projector plane coordinates of the quadrilateral to where the PJdisplay is mapped onto (see section 3.2.11); the constants “leftMost”, “rightMost”, “topMost”, and “bottomMost” correspond to edge coordinates of the LCD projector plane.
- `other`: placeholder for other attributes.

3.2.11 PJdisplay

A PJdisplay is a rectangular region of application space that contains multiple image sources, either static or dynamic. The origin and extent of the rectangular area is determined by the application through “use” or “set” commands. When a PJdisplay is mapped onto a PJsurface (through an ‘activate’ command), the region is mapped into a quadrilateral on the LCD projector plane so that the coordinate system defined by application is appropriately mapped on the projected surface.

Image sources are stacked on top of each other, according to the order described in the PJdisplay object. The bottommost object is always the “black” image source. Image sources are only displayed when activated by an “activate” method. If a set of image source objects is activated with the “overlay” option set to “true”, then the activated sources are stacked on top of the existing sources. Otherwise, the previously displayed sources are deactivated automatically, the “black” image source is loaded, and the newly activated sources are stacked on the order described in the activation method.

Syntax:

```xml
<PJdisplay name="[control]|*" left="[0]|#" right="[800]|#" top="[0]|#"
    bottom="[600]|#" other="[]|*"
    >
    * <PJimage * />
    * <PJstream * />
</PJdisplay>
```

Attributes:

- `name`: name of the PJdisplay; the “control” display corresponds to a display that does not store values, but that can be used to create PJdisplays dynamically.
- `left`: x-coordinate of the leftmost value inside the display.
- `right`: x-coordinate of the rightmost value inside the display.
- `top`: y-coordinate of the topmost value inside the display.
- `bottom`: y-coordinate of the bottommost value inside the display.
other: placeholder for other attributes.

### 3.2.12 PJimage

A PJimage object defines a static source of imagery. Either the source can be a file available in a web server (using the object EDimage) or the image can be transferred directly as part of the message (EDimageData). There are four pre-defined images, “black”, “calibration”, “logo”, and “arrow_cursor”. The image is positioned according to the coordinate system defined by the currently activated PJdisplay object.

**Syntax:**

```
<PJimage name="*|black|calibration|logo|arrow_cursor|*"
   left="[leftMost]|#" right="[rightMost]|#"
   top="[topMost]|#" bottom="[bottomMost]|#" other="[]|*"
  >
    <EDimage * />
  </PJimage>
```

**Attributes:**

- **name**: name of the PJimage; “black” refers to a completely blank image; “calibration” contains a special calibration pattern; “logo” refers to the ED logo; and “arrow_cursor” contains a simple arrow that can be used as a cursor.
- **left**: position, in relation to the PJdisplay coordinate system, where the left edge of the image should be placed; “leftMost” corresponds to the leftmost position on the PJdisplay.
- **right**: position, in relation to the PJdisplay coordinate system, where the right edge of the image should be placed; “rightMost” corresponds to the rightmost position on the PJdisplay.
- **top**: position, in relation to the PJdisplay coordinate system, where the top edge of the image should be placed; “topMost” corresponds to the topmost position on the PJdisplay.
- **bottom**: position, in relation to the PJdisplay coordinate system, where the bottom edge of the image should be placed; “bottomMost” corresponds to the bottommost position on the PJdisplay.
- **other**: placeholder for other attributes.

### 3.2.13 PJstream

This object defines a stream of images to be displayed, grabbed from a stream server of desktop images, a camera server, or a movie file. The image is positioned according to the coordinate system defined by the currently activated PJdisplay object. Image frames are refreshed in real-time, i.e., as fast as the system can do it given its own and the server’s limitations, only when a “sample” method is called (see section 3.3.5), or every time a certain number of milliseconds has passed.

**Syntax:**

```
<PJstream name="*" sampling="[once]|real-time|#"
   left="[leftMost]|#" right="[rightMost]|#
   top="[topMost]|#" bottom="[bottomMost]|#" other="[]|*"
  >
    | <EDdesktop * />
    | <EDcamera * />
    | <EDmovie * />
  </PJstream>
```

**Attributes:**
name: name of the PJstream.

sampling: defines how often the system refreshes the displayed image: “real-time” makes the system refresh the projected display as fast as possible; “once” corresponds to a single sampling, taken immediately whenever the object is activated; any numbered entered in this field corresponds to the minimum number of milliseconds between two consecutive samplings.

left: position, in relation to the PJdisplay coordinate system, where the left edge of the stream should be placed; “leftMost” corresponds to the leftmost position on the PJdisplay.

right: position, in relation to the PJdisplay coordinate system, where the right edge of the stream should be placed; “rightMost” corresponds to the rightmost position on the PJdisplay.

top: position, in relation to the PJdisplay coordinate system, where the top edge of the stream should be placed; “topMost” corresponds to the topmost position on the PJdisplay.

bottom: position, in relation to the PJdisplay coordinate system, where the bottom edge of the stream should be placed; “bottomMost” corresponds to the bottommost position on the PJdisplay.

other: placeholder for other attributes.

3.3 Methods

3.3.1 use

This method is used by an application to communicate to PJML module about which objects it will use. Any attribute of objects but ‘name’ are ignored by the ‘use’ method. An application cannot send a command to an object to which it has not issued an 'use' command, or after releasing the object: this generates an error.

The ‘use’ method can be used to simultaneously use objects of different levels. When applied to an object composed of other objects, the method is applied both on the “parent” object and on the “children” objects.

Syntax:

```
<use id="[unknown]|*" other="[]|*" >
* <PJsurface name="*" />
* <PJdisplay name="*" >
  * <PJimage name="*" >
    <EDimage src="*" />  
  </PJimage>
* <PJstream name="*" >
  | <EDdesktop src="*" />  
  | <EDcamera src="*" />  
  | <EDmovie src="*" />  
</PJstream>
</PJdisplay>
</use>
```

Attributes:

- id: identification number of the call for this method, generated by the application program.
- other: placeholder for other method attributes.

Return values:

For general values, see section 3.4. If the object already existed, a return value of 200 (“OK.”) is returned (see section 2.5).
### 3.3.2 release

The 'release' method removes a PJ object (such as PJdisplay, PJsurface, etc.) from the list of objects in a PJML that are accessible from the issuing application. Any attribute of objects but 'name' are ignored by the 'use' method. An application cannot send a command to an object to which it has not issued an 'use' command, or after releasing the object: this generates an error.

If the “name” attribute is empty or contains an invalid name, no object is released. A “name= "all"” attribute releases all objects of that type from the application list. If a name is specified, and the corresponding object is active, the object is deactivated.

**Syntax:**

```
<release id="[unknown]|*" other="[]|*" >
  * <PJsurface name="[]|all|*" />
  * <PJdisplay name="[]|all|*" >
    * <PJimage name="[]|all|*" />
    * <PJstream name="[]|all|*" />
  </PJdisplay>
</release>
```

**Attributes:**

- **id:** identification number of the call for this method, generated by the application program.
- **other:** placeholder for other method attributes.

**Return values:** see section 3.4.

### 3.3.3 set

The “set” method updates the attributes of an ED object (such as PJdisplay, PJsurface, etc.). The object must exist in the current list of objects of that type. If it exists, the attributes are set according to the values in the method. If multiple types of objects are set by one “set” method, paralelization may occur. For example, the mirror and the projector states can be set in parallel. Persistent EDML objects (i.e., objects with name) should be always explicitly referred by their names in “set” methods.

The ‘set’ method can be used to simultaneously set attributes in objects of different level. When applied to an object composed of other objects, the method is applied both on the “parent” object and on the “children” objects.
Syntax:

```xml
<set id="[unknown]|*" other="[]|*">
+ <PJsystem />
* <PJsurface >
  + <PJmirrorState />
  + <PJprojectorState />
  + <PJprojCorrection >
  | <PJposition3D />
  | <PJhomography />
</PJprojCorrection>
</PJsurface>
* <PJdisplay >
  * <PJimage >
  <EDimage />
</PJimage>
  * <PJstream >
    | <EDdesktop />
    | <EDcamera />
    | <EDmovie />
</PJstream>
</PJdisplay>
</set>
```

**Attributes:**

- **id**: identification number of the call for this method, generated by the application program.
- **other**: placeholder for other method attributes.

**Return values:** see section 3.4.

### 3.3.4 get

The “get” method queries the ED-projector and returns the names and attributes of ED objects (PJdisplay, PJsurface, etc.). If “state="active"” or “state="inactive"”, it returns only the list of active, respectively inactive, objects of that type. If “name” is the empty string or contains an invalid name, no information is returned. If “name="all"” then “get” returns information about all the objects of that particular type. If an object name is specified, then only information about that particular object, if it exists, is returned. The information is returned in the same format as objects are created and set.

Syntax:

```xml
<get id="[unknown]|*" state="[all]|active|inactive|*" other="[]|*">
+ <PJsystem >
  + <PJmirror />
  + <PJprojector />
  + <PJrendering />
</PJsystem>
* <PJsurface name="[]|all|*">
* <PJdisplay name="[]|all|*">
  * <PJimage name="[]|all|*">
  * <PJstream name="[]|all|*">
</PJdisplay>
</get>
```

**Attributes:**

- **id**: identification number of the call for this method, generated by the application program.
state: filter for obtaining only the active or inactive objects of the type listed in the body of the method.
other: placeholder for other method attributes.

Return values:
In addition to the return value, as described in section 3.4, “get” returns the list of objects requested inside the “return” object, according to this syntax:

```xml
<pjml * >
 <return * >
  * [list of objects returned]
 </return>
</pjml>
```

3.3.5 activate
The ‘activate’ method is used to activate an object. Any attribute of objects but ‘name’ are ignored by the ‘use’ method. An application cannot activate an object that was not accessed through a ‘use’ command, or after releasing the object.

Only one ‘PJsurface’ and one ‘PJdisplay’ are active at each moment of time, but multiple ‘PJimages’ and ‘PJstreams’ can be active at the same time. Whenever a ‘PJsurface’ or a ‘PJdisplay’ is activated, the previous active object is first deactivated. Activating a ‘PJdisplay’ without specifying any of its component ‘PJimages’ or ‘PJstreams’ causes just the erasing of the previous ‘PJdisplay’.

Syntax:

```xml
<activate id="[unknown]|*" other="[]|*" >
 + <PJsystem >
  + <PJmirror />
  + <PJprojector />
  + <PJrendering />
 </PJsystem>
* <PJsurface name="[]|all|*" />
* <PJdisplay name="[]|all|*" >
  * <PJimage name="[]|all|*" />
  * <PJstream name="[]|all|*" />
 </PJdisplay>
</activate>
```

Special Methods:

```xml
<activate id="[unknown]|*" hideMovement="[true]|false" other="[]|*" >
  * <PJsurface * />
</activate>
```

```xml
<activate id="[unknown]|*" overlay="[true]|false" other="[]|*" >
  * <PJdisplay name="[]|all|*" >
    * <PJimage name="[]|all|*" />
    * <PJstream name="[]|all|*" />
 </PJdisplay>
</activate>
```

Attributes:

id: identification number of the call for this method, generated by the application program.
other: placeholder for other method attributes.
hideMovement: when “activate” is applied to a PJsurface object, this attribute specify whether the movement to the new surface should be seen or not; in the “true” case, the PJimage object called “black” is activated automatically, the movement is performed, and the previous PJdisplay configuration is restored after the movement. If multiple PJsurfaces are listed, the method is sequentially applied to each PJSurface, and each transition is governed by this attribute.

overlay: if “false”, the active display is erased and the sources are loaded, in the order they were activated, the last being shown on the top; if “true”, the list is added to the top (last) of the active list of image sources being displayed. If multiple PJdisplays are listed, the method is sequentially applied to each PJdisplay, and each transition is governed by this attribute.

Return values: see section 3.4.

3.3.6 deactivate
The ‘deactivate’ method is used to deactivate an object. Any attribute of objects but ‘name’ are ignored by the ‘use’ method. An application cannot deactivate an object that was not accessed through a ‘use’ command, or after releasing the object.

If the “name” attribute is empty or contains an invalid name, no object is deactivated. A “name="all"” attribute deactivates all objects of that type from the application list.

Syntax:

    <deactivate id="[unknown]" other="[]" />
    + <PJsystem>
        + <PJmirror />
        + <PJprojector />
        + <PJrenderer />
    </PJsystem>
    * <PJsurface name="[]" all="[]" />
    * <PJdisplay name="[]" all="[]" />
        * <PJimage name="[]" all="[]" />
        * <PJstream name="[]" all="[]" />
    </PJdisplay>
</deactivate>

Attributes:

id: identification number of the call for this method, generated by the application program.

other: placeholder for other method attributes.

Return values: see section 3.4.

3.3.7 sample
The “sample” method can be applied only to PJstream objects, and it is effectively only if the object has the attribute “sampling="once"”. The application of this method causes a new image to be acquired from that stream. After sampling, the PJstream is always shown on top of other display objects.

Syntax:

    <sample id="[unknown]" other="[]" />
    * <PJdisplay name="[]" />
        * <PJstream name="[]" all="[]" />
    </PJdisplay>
</sample>
Attributes:

id: identification number of the call for this method, generated by the application program.

other: placeholder for other method attributes.

Return values: see section 3.4.

3.4 Return Messages

For syntax of return messages, see section 2.5.

Return values specific to the PJML language have the prefix 30**.

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>0</td>
<td>General failure.</td>
</tr>
<tr>
<td>1…999</td>
<td>General error, see section 2.5.</td>
</tr>
<tr>
<td>3010</td>
<td>Invalid PJdisplay name.</td>
</tr>
<tr>
<td>3011</td>
<td>Invalid PJsurface name.</td>
</tr>
<tr>
<td>3012</td>
<td>Invalid PJimage name.</td>
</tr>
<tr>
<td>3013</td>
<td>Invalid PJstream name.</td>
</tr>
<tr>
<td>3050</td>
<td>PObject parameter beyond range.</td>
</tr>
<tr>
<td>3051</td>
<td>Invalid server name.</td>
</tr>
<tr>
<td>3052</td>
<td>Invalid file name.</td>
</tr>
<tr>
<td>3053</td>
<td>Invalid data structure.</td>
</tr>
</tbody>
</table>
4 VIML: Commands, Objects, Methods, and Events for the Vision Interface

This chapter describes the interface to the Vision Interface module.

4.1 Commands

A VIML command is a list of basic VIML methods sent to objects.

Syntax:

```xml
<viml key="*" application="*">
  * <use * />
  * <release * />
  * <set * />
  * <get * />
  * <activate * />
  * <deactivate * />
</viml>
```

Attributes:
- **key**: string with a key to be matched and verified by the receiving module. If not present, or if it does not match the module key, the command is ignored.
- **application**: the name of the EDapplicationInfo to which the returns should be sent. If not present, the command is ignored.

Return values:
All returns of VIML commands are asynchronously sent to the URL of the application XML server, at the time of completion of the command. Returns of commands listed in the same VIML command are sent one by one and marked with the id of the command.

4.2 Objects

User actions are sensed by activating VIconfigurations on VIsurfaces. A VIconfiguration is a collection of widgets that are active concurrently. Each widget can sense a specific user action. Examples of VIconfigurations might be a menu of color buttons or a tracking region with a “done” button. Each widget (VIbutton, VItrackArea, etc) generates a stream of VIevents (e.g. VIeventTouch). To define a VIconfiguration, widgets are arranged in an Interaction Area within application space in the definition layer. The interaction area can reside on top of the projector's display area or separately. Widgets outside the display area can have no projected representation. This separation can be used to provide a control interaction that takes place on top of some artifact in the environment removed from the projection, for example a control panel for a presentation. If the interaction area lies on top of the display area, then the interaction will take place within the projection.

A VIsurface corresponds to a location within the camera's field of view where an interaction takes place. Because the camera's view is determined by the CAML, A VIsurface is generally only meaningful when used with a particular CASurface. Of course, a CASurface may have multiple VIsurfaces associated with it. The EDML user must maintain the correspondence between the surfaces.

A VIsurface is defined by a quadrilateral Interaction Region in the camera image. When a VIconfiguration is activated on a VIsurface, the interaction area of the configuration is mapped to the interaction region for the surface. This mapping allows the vision system to locate widgets in the camera's field of view. Each
VIsurface may also contain information about how it expects users to appear in the camera's field of view, such as the expected size or orientation of the user's hands. This information will be gathered during a separate calibration step (details TBD).

4.2.1 VIsystem
This object describes the basic vision interface system. Only one of these objects exists in each vision interface module. This object is mostly used to activate and deactivate the whole sensing system.

Syntax:

```xml
<VIsystem other="[]|*" />
```

Attributes:

- other: placeholder for other attributes.

4.2.2 VIsurface
This object describes the attributes of a VIsurface to where a VIconfiguration can be mapped.

Syntax:

```xml
<VIsurface name="*" cameraModule="*" cameraSurface="*"
    leftBottomX="[leftMost]|#" leftBottomY="[bottomMost]|#"
    leftTopX="[leftMost]|#" leftTopY="[topMost]|#"
    rightTopX="[rightMost]|#" rightTopY="[topMost]|#"
    rightBottomX="[rightMost]|#" rightBottomY="[bottomMost]|#"
    other="[]|*" />
```

Attributes:

- name: name of a VIsurface.
- cameraModule: name of the camera module that controls the camera.
- cameraSurface: name of the position of the camera associated with this surface.
- leftTopX, leftTopY, rightTopX, rightTopY, leftBottomX, leftBottomY, rightBottomX, rightBottomY: camera image coordinates of the quadrilateral to where the VIconfiguration is mapped onto (see section 4.2.3); the constants "leftMost", "rightMost", "topMost", and "bottomMost" correspond to edge coordinates of the camera image plane.
- other: placeholder for other attributes.

4.2.3 VIconfiguration
A VIconfiguration is a rectangular region of application space that contains multiple Visual Interface widgets for an interaction. The origin and extent of the rectangular area is determined by the application through "create" or "set" commands. When a VIconfiguration is mapped onto a VIsurface (through an 'activate' command), the region is mapped into a quadrilateral in the camera image so that the location of its widgets within that camera image can be determined.

The location of the camera image quadrilateral is set during calibration. When a VIconfiguration/VIsurface pair is activated, the camera image is examined for user interactions with the widgets, and interaction events are returned to the application.

Syntax:
<VIconfiguration name="*" left="[0]|#" right="[800]|#" top="[0]|#" bottom="[600]|#" other="[]|*">
  * <VIbutton * />
  * <VIslider * />
  * <VItrackArea * />
  * <VIwidget * />
  * <VIgestureArea * />
</VIconfiguration>

**Attributes:**

- **name:** name of the VIconfiguration.
- **left:** x-coordinate of the leftmost value inside the configuration.
- **right:** x-coordinate of the rightmost value inside the configuration.
- **top:** y-coordinate of the topmost value inside the configuration.
- **bottom:** y-coordinate of the bottommost value inside the configuration.
- **other:** placeholder for other attributes.

### 4.2.4 VIbutton

This object describes a VIbutton, a widget that detects touch-like gestures on a circular area of the camera image, generating ‘VIeventTouch’ events.

**Syntax:**

```
<VIbutton name="*" x="[400]|#" y="[300]|#" size="[100]|#" other="[]|*" />
```

**Attributes:**

- **name:** name of the VIbutton.
- **x:** x-coordinate of the center of the button, relative to the VIconfiguration coordinate system.
- **y:** y-coordinate of the center of the button, relative to the VIconfiguration coordinate system.
- **size:** diameter of the button, relative to the size of the VIconfiguration.
- **other:** placeholder for other attributes.

### 4.2.5 VIslider

This object describes a VIslider, a widget that returns ‘VIeventValueChanged’ events.

**Syntax:**

```
<VIslider name="*" x="[400]|#" y="[300]|#" size="[100]|#" orientation="horizontal|vertical" other="[]|*
  minValue="[0]|#" maxValue="[100]|#" initValue="[0]|#" />
```

**Attributes:**

- **name:** name of the VIslider.
- **x:** x coordinate of the center of the slider, relative to the VIconfiguration coordinate system.
- **y:** y coordinate of the center of the slider, relative to the VIconfiguration coordinate system.
size: length of the slider, relative to the size of the application space units.
orientation: determines whether the main axis of the slider is positioned horizontally or vertically.
minValue: value associated to the bottom (left) of the slider.
maxValue: value associated to the top (right) of the slider.
initValue: initial value of the slider (only used when the slider is created).
other: placeholder for other attributes.

4.2.6 VItrackArea
This object describes a VItrackArea, a widget that tracks position in a rectangular area of the VIconfiguration area, and returns 'VIeventPositionChange' events whenever there is a change. The position of these events is returned relative to the coordinate system defined by the attributes leftValue, topValue, rightValue, and bottomValue.

Syntax:

```xml
<VItrackArea name="*" left="[0]\|#" right="[800]\|#" top="[0]\|#"
            bottom="[600]\|#" leftValue="[0]\|#" rightValue="[800]\|#" topValue="[0]\|#"
            bottomValue="[600]\|#" other="[ ]\|*">
</VItrackArea>
```

Attributes:

name: name of the VItrackArea.

left: x-coordinate of the leftmost point in the VItrackArea, relative to the VIconfiguration coordinate system.

right: x-coordinate of the rightmost point in the VItrackArea, relative to the VIconfiguration coordinate system.

top: y-coordinate of the topmost point in the VItrackArea, relative to the VIconfiguration coordinate system.

bottom: y-coordinate of the bottommost point in the trackArea, relative to the VIconfiguration coordinate system.

leftValue: value associated to the x-coordinate of the leftmost point in the trackArea.

rightValue: value associated to the x-coordinate of the rightmost point in the trackArea.

topValue: value associated to the y-coordinate of the topmost point in the trackArea.

bottomValue: value associated to y-coordinate of the the bottommost point in the trackArea.

other: placeholder for other attributes.

4.2.7 VIgestureArea
This object describes a VIgestureArea, a widget that tracks position in a rectangular area of the VIconfiguration area, and returns 'VIeventGesture' events whenever it detects a particular user gesture.
Syntax:

```xml
<VIGestureArea name="*" left="[0]|#" right="[800]|#" top="[0]|#"
bottom="[600]|#" other="[]|*"
/>```

Attributes:

- **name**: name of the VIGestureArea.
- **left**: coordinate of the leftmost point in the VIGestureArea, relative to the VIconfiguration coordinate system.
- **right**: coordinate of the rightmost point in the VIGestureArea, relative to the VIconfiguration coordinate system.
- **top**: coordinate of the topmost point in the VIGestureArea, relative to the VIconfiguration coordinate system.
- **bottom**: coordinate of the bottommost point in the VIGestureArea, relative to the VIconfiguration coordinate system.
- **other**: placeholder for other attributes.

### 4.2.8 VIWidget

This object defines a generic widget of type `type`. This is to be used to extend the set of widgets of handled by the VI system. It can generate all types of VIevents.

Syntax:

```xml
<VIwidget name="*" type="*" x="[400]|#" y="[300]|#" size="[100]|#"
other="[]|*"
/>```

Attributes:

- **name**: name of the widget.
- **type**: the type of the widget. This attribute has to be understood by the VI system.
- **x**: x coordinate of the center of the widget, relative to the VIconfiguration coordinate system.
- **y**: y coordinate of the center of the widget, relative to the VIconfiguration coordinate system.
- **size**: diameter of the widget, relative to the size of the application space units.
- **other**: placeholder for other attributes, including all attributes needed to create and modify a particular type of widget.

### 4.3 Methods

#### 4.3.1 use

This method is used by an application to communicate to VIML module about which objects it will use. Any attribute of objects but ‘name’ are ignored by the ‘use’ method. An application cannot send a command to an object to which it has not issued an ‘use’ command, or after releasing the object: this generates an error.

The ‘use’ method can be used to simultaneously use objects of different levels. When applied to an object composed of other objects, the method is applied both on the “parent” object and on the “children” objects.
Syntax:

```xml
<use id="[unknown]|*" other="[]|*" >
  * <VIsurface name="*" />
  * <VIconfiguration name="*" >
    * <VIbutton name="*" />
    * <VIslider name="*" />
    * <VItrackArea name="*" />
    * <VIgestureArea name="*" />
    * <VIwidget name="*" />
  </VIconfiguration >
</use>
```

Attributes:

- **id**: identification number of the call for this method, generated by the application program.
- **other**: placeholder for other method attributes.

Return values:

For general values, see section 4.4. If the object already existed, a return value of 200 (“OK.”) is returned (see section 2.5).

Examples:

This command tries to habilitate the application ‘xPlayer’ to use the VIsurface called ‘table’.

```xml
<viml application="xPlayer" key="EDML">
  <use id="3212">
    <VIsurface name="table" />
  </use>
</viml>
```

This command tries to habilitate the application ‘xPlayer’ to use the VIconfiguration called ‘control’ and its Vibuttons ‘b1’ and ‘b2’. Even if the stored VIconfiguration contains other widgets, only ‘b1’ and ‘b2’ will be accessible to ‘xPlayer’.

```xml
<viml application="xPlayer" key="EDML">
  <use id="3213">
    <VIconfiguration name="control" >
      <VIbutton name="b1" />
      <VIbutton name="b2" />
    </VIconfiguration>
  </use>
</viml>
```

4.3.2 release

The ‘release’ method removes a VI object (such as VIconfiguration, VIwidget, etc.) from the list of objects in a VIML that are accessible from the issuing application. Any attribute of objects but ‘name’ are ignored by the ‘use’ method. An application cannot send a command to an object to which it has not issued an ‘use’ command, or after releasing the object: this generates an error.

If the “name” attribute is empty or contains an invalid name, no object is released. A “name="all"” attribute releases all objects of that type from the application list. If a name is specified, and the corresponding object is active, the object is deactivated.
4.3.3 set

The "set" method updates the attributes of an VI object (such as VIconfiguration, VIwidget, etc.). An application cannot send a command to an object to which it has not issued an 'use' command, or after releasing the object: this generates an error.

If the object exists, the attributes are set according to the values in the method. If multiple types of objects are set by one "set" method, paralelization may occur. Persistent EDML objects (i.e., objects with name) should be always explicitly referred by their names in "set" methods. If the object is active, the change in the attributes occurs immediately.

The 'set' method can be used to simultaneously set attributes in objects of different levels. When applied to an object composed of other objects, the method is applied both on the "parent" object and on the "children" objects.

Syntax:

```xml
<set id="[unknown]|*" other="[]|*" >
  + <VIsystem * />  
  * <VIconfiguration * >
    * <VIsurface * />  
    * <VIslider * />  
    * <VIsurface * />  
    * <VIsurface * />  
  * <VIslider * />  
  * <VIsurface * />  
  * <VIsurface * />  
  * <VIsurface * />  
  * <VIsurface * />  
  * <VIsurface * />  
  * <VIsurface * />  
</VIconfiguration >
</set>
```

Attributes:

- **id**: identification number of the call for this method, generated by the application program.
- **other**: placeholder for other method attributes.

Return values: see section 4.4.
### 4.3.4 get

The “get” method queries the Visual Interface and returns the names and attributes of Visual Interface objects (VIconfiguration, VIwidget, etc.). Any attribute of objects but ‘name’ are ignored by the ‘use’ method. An application cannot send a ‘get’ method to an object to which it has not issued an 'use' command, or after releasing the object: this generates an error.

If “state="active"” or “state="inactive"”, it returns only the list of active, respectively inactive, objects of that type. If “name” is the empty string or contains an invalid name, no information is returned. If “name="all"” then “get” returns information about all the objects of that particular type. If an object name is specified, then only information about that particular object, if it exists, is returned. The information is returned in the same format as objects are created and set, except if the attribute ‘state’ is equal to ‘active’ or ‘inactive’, when only the object names are returned.

**Syntax:**

```xml
<get id="[unknown]|*" state="[all]|active|inactive|*" other="[]|*" >
  + <VI system />
  * <VI surface name="[]|all|*"/>
  * <VI configuration name="[]|all|*" >
    * <VI button name="[]|all|*" />
    * <VI slider name="[]|all|*" />
    * <VI trackArea name="[]|all|*" />
    * <VI gestureArea name="[]|all|*" />
    * <VI widget name="[]|all|*" />
</VI configuration>
</get>
```

**Attributes:**

- id: identification number of the call for this method, generated by the application program.
- state: filter for obtaining only the active or inactive objects of the type listed in the body of the method.
- other: placeholder for other method attributes.

**Return values:**

In addition to the return value, as described in section 4.4, “get” returns the list of objects requested inside the “return” object, according to this syntax:

```xml
<viml * >
  <return * >
    * [list of objects returned]
  </return>
</viml>
```

### 4.3.5 activate

The ‘activate’ method is used to activate an object. Any attribute of objects but ‘name’ are ignored by the ‘use’ method. An application cannot activate an object that was not accessed through a 'use' command, or after releasing the object.
Syntax:

```
<activate id="[unknown]|*" other="[]|*" >
  + <VIsystem />
  * <VIsurface name="[]|*" />
  * <VIconfiguration name="[]|*" >
    * <VIbutton name="[]|*" />
    * <VIslider name="[]|*" />
    * <VItrackArea name="[]|*" />
    * <VIgestureArea name="[]|*" />
    * <VIwidget name="[]|*" />
  </VIconfiguration>
</activate>
```

Attributes:

- **id**: identification number of the call for this method, generated by the application program.
- **other**: placeholder for other method attributes.

Return values: see section 4.4.

Examples:

This command activates a VIML module.

```
<viml application="xPlayer" key="EDML" reverse="[]">
  <activate id="209A">
    <VIsystem />
  </activate>
</viml>
```

This command activates a VIconfiguration called ‘control’ and all of its widgets.

```
<viml application="xPlayer" key="EDML" reverse="[]">
  <activate id="210V">
    <VIconfiguration name="control" />
  </activate>
</viml>
```

This command activates the VIbuttons ‘b1’ and ‘b2’ of the VIconfiguration called ‘control’.

```
<viml application="xPlayer" key="EDML" reverse="[]">
  <activate id="210V">
    <VIconfiguration name="control" />
    <VIbutton name="b1" />
    <VIbutton name="b2" />
  </activate>
</viml>
```

4.3.6 **deactivate**

The ‘deactivate’ method is used to deactivate an object. Any attribute of objects but ‘name’ are ignored by the ‘use’ method. An application cannot deactivate an object that was not accessed through a ‘use’ command, or after releasing the object.

If the “name” attribute is empty or contains an invalid name, no object is deactivated. A “name="all"” attribute deactivates all objects of that type from the application list.
Syntax:

```xml
<deactivate id="[unknown]|*" other="[]|*" >
  + <VIsystem />
* <VIconfiguration name="[]|all|*" >
  * <VIbutton name="[]|all|*" />
  * <VIslider name="[]|all|*" />
  * <VIttrackArea name="[]|all|*" />
  * <VIgestureArea name="[]|all|*" />
* <VIwidget name="[]|all|*" />
</VIconfiguration >
</deactivate>
```

Attributes:

- `id`: identification number of the call for this method, generated by the application program.
- `other`: placeholder for other method attributes.

Return values: see section 4.4.

4.4 Return Messages

For syntax of return messages, see section 2.5.

Return values specific to the VIML language have the prefix 40**.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>0</td>
<td>General failure.</td>
</tr>
<tr>
<td>1…999</td>
<td>General error, see section 2.5.</td>
</tr>
<tr>
<td>4010</td>
<td>Invalid VIconfiguration name.</td>
</tr>
<tr>
<td>4011</td>
<td>Invalid VI surface name.</td>
</tr>
<tr>
<td>4012</td>
<td>Invalid VI widget name.</td>
</tr>
<tr>
<td>4050</td>
<td>VI object parameter beyond range.</td>
</tr>
</tbody>
</table>

4.5 Generated Events

A Vlevent is an ‘event’ object that progressively lists the configuration that generated the event, the widget that generated the event, and the event itself.
Syntax:

```xml
<viml vimlName="*" >
  <event id="*" >
    <VIconfiguration name="*" >
      | <VIbutton name="*" > <VIeventTouch * /> </VIbutton>
      | <VIslider name="*" > <VIeventValue * /> </VIslider>
      | <VItrackArea name="*" > <VIeventPosition * /> </VItrackArea>
      | <VIgestureArea name="*" >
        <VIeventGesture * />
      </VIgestureArea>
      | <VIwidget name="*" type="*" >
        <VIeventTouch * />
        <VIeventValueChange * />
        <VIeventPositionChange * />
        <VIeventGesture * />
        <VIeventType * />
      </VIwidget>
    </VIconfiguration>
  </event>
</viml>
```

Attributes:
- `vimlName`: name of the module of the vision interface.
- `id`: identification number generated by the VI system.

### 4.5.1 VIEventTouch
Events generated by confirmation widgets such as VIbutton.

Syntax:

```xml
<VIEventTouch generator="[]|*" certainty="[100]|#" other="[]|*" />
```

Attributes:
- `generator`: name of the event generator within the widget.
- `certainty`: degree of certainty of the event occurrence.
- `other`: placeholder for other event attributes.

### 4.5.2 VIEventValueChange
Events generated by valuator widgets such as VI slider.

Syntax:

```xml
<VIEventValueChange value="#" generator="[]|*" certainty="[100]|#" other="[]|*" />
```

Attributes:
- `value`: value associated with type 'value'.
- `generator`: name of the event generator within the widget.
- `certainty`: degree of certainty of the event occurrence.
other: placeholder for other event attributes.

4.5.3 **VIeventPositionChange**

Events generated by pointing widgets such as VItrackArea.

**Syntax:**

```xml
<VIeventPositionChange x="#" y="#" generator="[\*\*" certainty="[100]\#" other="[\*\*" />
```

**Attributes:**

- **x, y**: position where the event occurred, relative to the coordinate system defined by the widget (normally by VItrackArea).
- **generator**: name of the event generator within the widget.
- **certainty**: degree of certainty of the event occurrence.
- **other**: placeholder for other event attributes.

4.5.4 **VIeventGesture**

Events generated by gesture widgets such as VIgestureArea.

**Syntax:**

```xml
<VIeventGesture gestureType="[\*\*" generator="[\*\*" certainty="[100]\#" other="[\*\*" />
```

**Attributes:**

- **gestureType**: label for the gesture.
- **generator**: name of the event generator within the widget.
- **certainty**: degree of certainty of the event occurrence.
- **other**: placeholder for other event attributes.

4.5.5 **VIeventType**

Events generated by special widgets.

**Syntax:**

```xml
<VIeventType type="[\*\*" generator="[\*\*" certainty="[100]\#" other="[\*\*" />
```

**Attributes:**

- **type**: type of the event.
- **generator**: name of the event generator within the widget.
- **certainty**: degree of certainty of the event occurrence.
- **other**: placeholder for other attributes.
5 CAML: Commands, Objects, Methods, and Events for the Camera Control

This chapter describes the interface to the camera control module. The camera control interface performs three basic tasks: a) moving the camera to a pre-calibrated position; b) capturing and sending of images; and c) finding a calibration pattern on the view and return the coordinates of its extremities.

5.1 Commands

A CAML command is a list of basic CAML methods sent to objects.

Syntax:

```xml
<caml key="*" application="*" >
  * <use * />
  * <release * />
  * <set * />
  * <get * />
  * <activate * />
  * <deactivate * />
  * <getImage * />
  * <findPattern * />
</caml>
```

Attributes:

- **key**: string with a key to be matched and verified by the receiving module. If not present, or if it does not match the module key, the command is ignored.
- **application**: the name of the EDapplicationInfo to which the returns should be sent. If not present, the command is ignored.

Return values:

All returns of CAML commands are asynchronously sent to the URL of the application XML server, at the time of completion of the command. Returns of commands listed in the same CAML command are sent one by one and marked with the id of the command.

5.2 Objects

The basic object created by the camera control system is a CAsurface object, which determines the parameters needed to appropriately generate a stream of images from a camera: pan, tilt, zoom, focus, iris, etc. CAML also uses (only for return) the CApatternPosition object that describes the extremities of a pattern as detected by the camera control software.

5.2.1 CAsystem

This object describes the basic camera system. Only one of these objects exists in each camera module. This object is mostly used to activate or deactivate the whole camera system.

Syntax:

```xml
<CAsystem other="[] | *" />
```
Attributes:
other: placeholder for other attributes.

5.2.2 CAsurface
This object describes a state of the camera.

Syntax:
```xml
<CAsurface name="[control]|*" pan="[0]|#" tilt="[0]|#" zoom="[0]|#"
  focus="[auto]|#" accuracy="[standard]|maximum"
gain="[auto]|#" iris="[0]|#" shutter="[0]|#"
  other="[ ]|*" />
```

Attributes:
name: name of the CAsurface; the “control” surface corresponds to a surface that does not store values, but that can be used to create CAsurfaces dynamically.
pan: pan value of the camera, in a scale from –1000 to 1000.
tilt: tilt value of the camera, in a scale from –1000 to 1000.
zoom: zoom value of the camera, in a scale from 0 to 1000.
focus: focus value of the camera, in a scale from 0 to 1000; “auto” means automatic focus is on.
accuracy: determines whether the positioning of the camera has to be very accurate or not; “maximum” accuracy may require a longer time to move the camera.
gain: gain value of the camera, in a scale from –1000 to 1000; “auto” means automatic gain control (AGC) is on.
iris: iris value of the camera, in a scale from 0 to 1000; 0 is completely closed.
shutter: shutter speed of the camera, in a scale from 0 to 1000; 0 is the longest possible aperture time.
other: placeholder for other attributes.

5.2.3 CApatternPosition
This object describes the four corners of a pattern as detected by the camera, i.r.t. the camera image plane coordinate system. This object is used only to describe return values as requested by “findPattern” methods.

Syntax:
```xml
<CApatternPosition leftBottomX="[unknown]|#" leftBottomY="[unknown]|#"
  leftTopX="[unknown]|#" leftTopY="[unknown]|#"
  rightTopX="[unknown]|#" rightTopY="[unknown]|#"
  rightBottomX="[unknown]|#" rightBottomY="[unknown]|#"
  other="[ ]|*" />
```

Attributes:
leftBottomX, leftBottomY, leftTopX, leftTopY, rightTopX, rightTopY, rightBottomX, rightBottomY: camera image coordinates of the quadrilateral corresponding to the position of the calibration pattern; the “unknown” correspond to situations where the corner was not detected.
other: placeholder for other attributes.

5.3 Methods

5.3.1 use
This method is used by an application to communicate to CAML module about which objects it will use. Any attribute of objects but 'name' are ignored by the 'use' method. An application cannot send a command to an object to which it has not issued an 'use' command, or after releasing the object: this generates an error.

The 'use' method can be used to simultaneously use objects of different levels. When applied to an object composed of other objects, the method is applied both on the "parent" object and on the "children" objects.

Syntax:
```xml
<use id="[unknown]|*" other="[\[]|*" >
  * <CAsurface name="*" />
</use>
```

Attributes:
- id: identification number of the call for this method, generated by the application program.
- other: placeholder for other method attributes.

Return values:
For general values, see section 5.4. If the object already existed, a return value of 200 ("OK.") is returned (see section 2.5).

5.3.2 release
The 'release' method removes a VI object (such as VIconfiguration, VIwidget, etc.) from the list of objects in a VIML that are accessible from the issuing application. Any attribute of objects but 'name' are ignored by the 'use' method. An application cannot send a command to an object to which it has not issued an 'use' command, or after releasing the object: this generates an error.

If the "name" attribute is empty or contains an invalid name, no object is released. A "name="all" attribute releases all objects of that type from the application list. If a name is specified, and the corresponding object is active, the object is deactivated.

Syntax:
```xml
<release id="[unknown]|*" other="[\[]|*" >
  * <CAsurface name="[\[]|all|*" />
</release>
```

Attributes:
- id: identification number of the call for this method, generated by the application program.
- other: placeholder for other method attributes.

Return values: see section 5.4.

5.3.3 set
The "set" method updates the attributes of an CA object (such as CAsurface). The object must exist in the current list of objects of that type. If it exists, the attributes are set according to the values in the method. If
multiple types of objects are set by one “set” method, parallelization may occur. Persistent EDML objects (i.e., objects with name) should be always explicitly referred by their names in “set” methods.

The ‘set’ method can be used to simultaneously set attributes in objects of different levels. When applied to an object composed of other objects, the method is applied both on the “parent” object and on the “children” objects.

Syntax:

```
<set id="[unknown]|*" other="[]|*" >
  + <CAsystem * />
  * <CAsurface * />
</set>
```

Attributes:

- id: identification number of the call for this method, generated by the application program.
- other: placeholder for other method attributes.

Return values: see section 5.4.

5.3.4 get

The “get” method queries the camera control module and returns the names and attributes of CA objects (such as CAsurface). If “state="active"” or “state="inactive"”, it returns only the list of active, respectively inactive, objects of that type. If “name” is the empty string or contains an invalid name, no information is returned. If “name="all"” then “get” returns information about all the objects of that particular type. If an object name is specified, then only information about that particular object, if it exists, is returned. The information is returned in the same format as objects are created and set.

Syntax:

```
<get id="[unknown]|*" state="[all]|active|inactive|*" other="[]|*" >
  + <CAsystem />
  * <CAsurface name="[]|all|*" />
</get>
```

Attributes:

- id: identification number of the call for this method, generated by the application program.
- state: filter for obtaining only the active or inactive objects of the type listed in the body of the method.
- other: placeholder for other method attributes.

Return values:

In addition to the return value, as described in section 5.4, “get” returns the list of objects requested inside the “return” object, according to this syntax:

```
<caml * >
  <return * >
    * [list of objects returned]
  </return>
</caml>
```
5.3.5 activate

The ‘activate’ method is used to activate an object. Any attribute of objects but ‘name’ are ignored by the ‘use’ method. An application cannot activate an object that was not accessed through a ‘use’ command, or after releasing the object.

Syntax:

```xml
<activate id="[unknown]|*" other="[|]*" >
  + <CAsystem />
  * <CAsurface name="[|]*" />
</activate>
```

Attributes:

**id**: identification number of the call for this method, generated by the application program.

**other**: placeholder for other method attributes.

**Return values**: see section 5.4.

5.3.6 deactivate

The ‘deactivate’ method is used to deactivate an object. Any attribute of objects but ‘name’ are ignored by the ‘use’ method. An application cannot deactivate an object that was not accessed through a ‘use’ command, or after releasing the object.

If the “name” attribute is empty or contains an invalid name, no object is deactivated. A “name=all” attribute deactivates all objects of that type from the application list.

Syntax:

```xml
<deactivate id="[unknown]|*" other="[|]*" >
  + <CAsystem />
  * <CAsurface name="[|all]*" />
</deactivate>
```

Attributes:

**id**: identification number of the call for this method, generated by the application program.

**other**: placeholder for other method attributes.

**Return values**: see section 4.4.

5.3.7 getImage

This method is sent to the camera control module itself. It returns information about how to access the current image viewed by the camera as an image object, as described in sections Error! Reference source not found..

Syntax:

```xml
<getImage id="[unknown]|*" other="[|]*" >
  <CAsystem />
</getImage>
```

Attributes:

**id**: identification number of the call for this method, generated by the application program.

**other**: placeholder for other method attributes.
Return values:
In addition to the return value, as described in section 5.4, “getImage” returns an image object inside the ‘return’ object, according to this syntax. This object provides information about the URL or path of the server that has the image and the image name.

```
<return *
    <EDimage */
  </return>
</caml
```

5.3.8 findPattern
This method is sent to the camera control module itself. It returns the four corners of a calibration pattern, if present in the current image viewed by the camera.

Syntax:
```
<findPattern id="[unknown]|*" patternName="[calibration]|*" other="[]|*" >
  <CAsystem />
</findPattern>
```

Attributes:
- id: identification number of the call for this method, generated by the application program.
- patternName: name of the pattern to be found in the image; “calibration” refers to the standard calibration pattern.
- other: placeholder for other method attributes.

Return values:
In addition to the return value, as described in section 5.4, “findPattern” returns a CApatternPosition object inside the ‘return’ object, according to this syntax:

```
<return *
    <CApatternPosition */
  </return>
</caml
```

5.4 Return Messages
For syntax of return messages, see section 2.5.

Return values specific to the CAML language have the prefix 50**.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>0</td>
<td>General failure.</td>
</tr>
<tr>
<td>1…999</td>
<td>General error, see section 2.5.</td>
</tr>
<tr>
<td>5010</td>
<td>Invalid CAsurface name.</td>
</tr>
<tr>
<td>5050</td>
<td>CAobject parameter beyond range.</td>
</tr>
<tr>
<td>5100</td>
<td>Cannot send image.</td>
</tr>
<tr>
<td>5150</td>
<td>Cannot find pattern.</td>
</tr>
</tbody>
</table>
6 Reference

6.1 List of Return Values

For syntax of return messages, see section 2.5.

Generic return values are numbered between 0 and 999.

Success return values are numbered between 200 and 299.

Return values specific to the PJML language have the prefix 30**.

Return values specific to the VIML language have the prefix 40**.

Return values specific to the CAML language have the prefix 50**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>210</td>
<td>OK, but application is already open.</td>
</tr>
<tr>
<td>0</td>
<td>General failure.</td>
</tr>
<tr>
<td>1</td>
<td>Syntax error.</td>
</tr>
<tr>
<td>10</td>
<td>Cannot create object: object already exists.</td>
</tr>
<tr>
<td>11</td>
<td>Cannot delete object: object does not exist.</td>
</tr>
<tr>
<td>20</td>
<td>Invalid parameter.</td>
</tr>
<tr>
<td>21</td>
<td>Invalid attribute.</td>
</tr>
<tr>
<td>23</td>
<td>Invalid object name.</td>
</tr>
<tr>
<td>24</td>
<td>Invalid URL.</td>
</tr>
<tr>
<td>28</td>
<td>Invalid application.</td>
</tr>
<tr>
<td>50</td>
<td>Parameter beyond range.</td>
</tr>
<tr>
<td>3010</td>
<td>Invalid PJdisplay name.</td>
</tr>
<tr>
<td>3011</td>
<td>Invalid PJsurface name.</td>
</tr>
<tr>
<td>3012</td>
<td>Invalid PJimage name.</td>
</tr>
<tr>
<td>3013</td>
<td>Invalid PJstream name.</td>
</tr>
<tr>
<td>3050</td>
<td>PJobject parameter beyond range.</td>
</tr>
<tr>
<td>3051</td>
<td>Invalid server name.</td>
</tr>
<tr>
<td>3052</td>
<td>Invalid file name.</td>
</tr>
<tr>
<td>3053</td>
<td>Invalid data structure.</td>
</tr>
<tr>
<td>4010</td>
<td>Invalid VIconfiguration name.</td>
</tr>
<tr>
<td>4011</td>
<td>Invalid VIsurface name.</td>
</tr>
<tr>
<td>4012</td>
<td>Invalid VIwidget name.</td>
</tr>
<tr>
<td>4050</td>
<td>VIobject parameter beyond range.</td>
</tr>
<tr>
<td>5010</td>
<td>Invalid CASurface name.</td>
</tr>
<tr>
<td>5050</td>
<td>CAobject parameter beyond range.</td>
</tr>
<tr>
<td>5100</td>
<td>Cannot send image.</td>
</tr>
<tr>
<td>5150</td>
<td>Cannot find pattern.</td>
</tr>
</tbody>
</table>
References

# Index

**activate**
- CAML......................................................45
- PJML....................................................26
- VIML....................................................36
- CAML......................................................41
  - CApatternPosition..............................42
  - CASurface........................................42
  - CASystem..........................................41

**close**
- EDML......................................................11

**commands**
- CAML......................................................41
- EDML......................................................10
- PJML....................................................17
- VIML....................................................29

**deactivate**
- CAML......................................................45
- PJML....................................................27
- VIML....................................................37
- EDML......................................................10
  - basic objects........................................13
  - EApplicationInfo.................................10
  - EDcamera............................................14
  - EDdesktop..........................................13
  - EDimage.............................................13
  - EDmovie..........................................14
- Eevent..................................................14

**events**
- basic syntax and values..............................16
- VIML....................................................38
  - VIeventGesture....................................40
  - VIeventPositionChange............................40
  - VIeventTouch......................................39
  - VIeventType........................................39
  - VIeventValueChange...............................39

**example**
- activating VIML objects.............................37
- multiple returns......................................15
- opening an application...............................11
- using VIML objects....................................34

**findPattern**
- CAML......................................................46

**get**
- CAML......................................................44
- EDML......................................................12
- PJML....................................................25
- VIML....................................................36

**getImage**
- CAML......................................................45

**methods**
- CAML......................................................43
- EDML......................................................11
- PJML....................................................23
- VIML....................................................33

**objects**
- activation.............................................9
- CAML......................................................41
- EDML......................................................10
- persistency............................................9
- PJML....................................................17
- VIML....................................................29

**open**
- EDML......................................................11
- PJML....................................................17
- PJdisplay.............................................21
- PJhomography.........................................21
- PJimage...............................................22
- PJmirror..............................................18
- PJmirrorState.........................................19
- Pjposition3D.........................................20
- PJprojCorrection.....................................19
- PJprojector..........................................18
- PJprojectorState.....................................19
- PJrendering..........................................18
- PJstream.............................................22
- PJsurface............................................18
- PJsystem..............................................17

**release**
- CAML......................................................43
- PJML....................................................24
- VIML....................................................34

**return messages**
- basic syntax..........................................15
- CAML......................................................46
- complete list.........................................47
- PJML....................................................28
- VIML....................................................38
<table>
<thead>
<tr>
<th>Sample</th>
<th>PJML</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PJML</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>CAML</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>EDML</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>PJML</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>VIML</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use</th>
<th>CAML</th>
<th>43</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PJML</td>
<td>23</td>
</tr>
</tbody>
</table>

| VIML | 33 |
| VIButton | 29 |
| VIConfiguration | 31 |
| VIGestureArea | 30 |
| VISlider | 32 |
| VIInterface | 31 |
| VIOverlay | 30 |
| VISystem | 30 |
| VITrackArea | 32 |
| VIView | 33 |