Advanced Programmer's Guide

for

pV3 Rev. 2.05

Bob Haimes

December 12, 2001

DISCLAIMER: This programming interface is maturing. The goal is to mimic **Visual3***s advanced programming interface as much as possible.*

Sections marked with this change-bar are have had a major change from last release (Rev 2.00) and may require some programming modifications.

License

Contents

1 Introduction

The control of the **pV3** interactive and post-processing visualization session is driven by the user. Much time and effort were placed in making that interaction *real-time*. But there are many instances when you may want the control of what is rendered on the screen to be specified by a program. Examples of this are demonstrations, visual results from expert-systems, macros and etc.

In that the **pV3** interactive server and the **pV3** post-processing viewer are **X** windows applications, control is driven by **X** events. Figure 1 shows these task's normal processing loop. In general, all **X** events are extracted from the **X** event queue and the internal **pV3** state information is adjusted appropriately. When all events are exhausted, any changes to state that require getting data from the application cause the server to request various extracts from the clients in the interactive session. After this information collection phase is done, the windows that require updating are redrawn during the rendering phase. Then the event queue is checked for more user interaction, and this continues for the life of the **pV3** server or viewer.

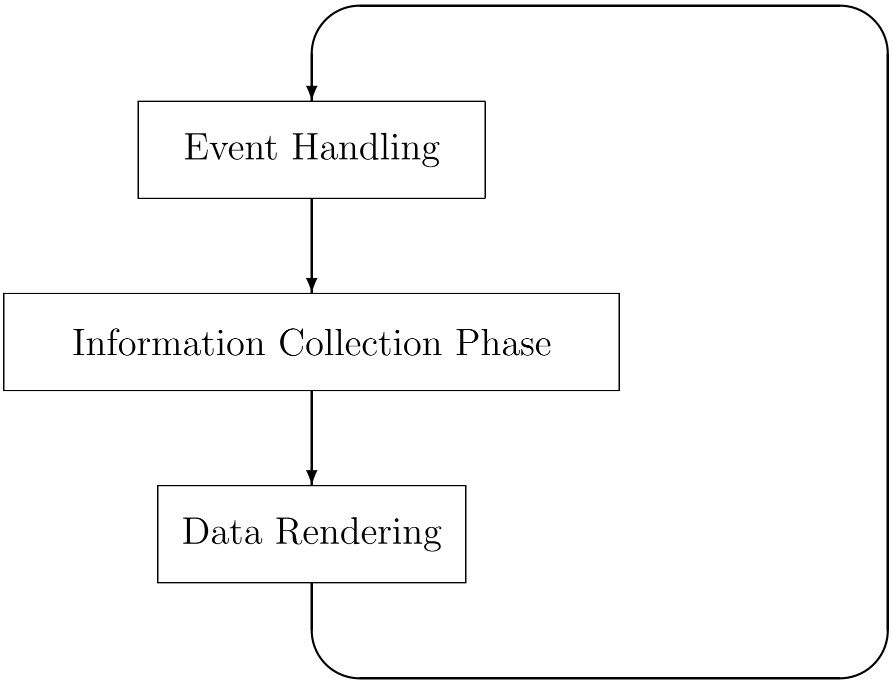

Figure 1: The **pV3** Server's Internal Loop

Also, to simplify the data handling, **pV3** only exposed it's internal data to the programmer when it required data on the client side. The data generated to produce cut surfaces, streamlines and other objects was hidden. This document describes how the advanced **pV3** programmer can access this extract information on the server and client sides.

2 pV3 Server & Viewer Event Handling

One of the goals of allowing programming control in the **pV3** suite is to maintain the current action as the default. Also this must allow the programmer to either modify the effects of an event or completely take over.

Before attempting to program **pV3** at this level it is important to understand the interactive server's and post-processing viewer's program structure. This includes the internals of the event handling phase.

The details of the **pV3** server and **pV3** viewer event handling phase can be seen in Figure 2. First, an **X** event is pulled of the **X** event queue. The event is described by a window ID - $IWIN$ (the window in which the event occurred), the type of event (e.g., KeyPress, ButtonPress, and etc.), the location in the window of the event, and finally any state data (e.g., which button or key was pressed). There is a special window ID flag of -1 which indicates that the event queue was empty.

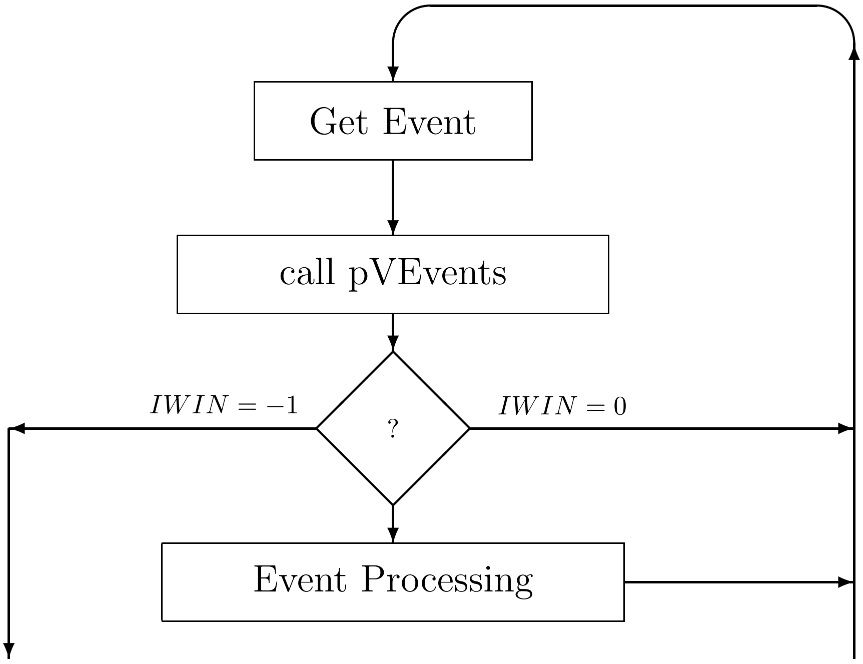

Figure 2: **pV3** Server and Viewer Event Handling

Next a programmer-supplied routine **pVEvents** is called with the event data. **pVEvents** has the option of passing the event on (doing nothing), performing some action based on the event, changing the event (by modifying the arguments) or having the event ignored (by setting $IWIN$, the window ID, to 0). The other routines described in this document may be called within **pVEvents** to inquire and set many of the **pV3** internal state variables.

On the basis of the value of $IWIN$, as returned by **pVEvents**; the event processing section of **pV3** is called when $IWIN > 0$, the event is ignored if **pVEvents** sets $IWIN = 0$, or this phase is terminated with $IWIN = -1$. When changing the action of the **pV3** server it is important to remember that nothing will be updated in the windows until **pVEvents** returns with $IWIN = -1$. Also, if **pVEvents** was called with $IWIN = -1$ and an event is constructed and passed through to the event processing section, **pVEvents** will be re-entered (possibly with $IWIN \neq -1$), before the screen gets updated.

Examples of **pVEvents**, that perform various functions, are contained within the distribution.

3 Use of Pointers & FORTRAN in the pV3 Server Suite

Some of the data that gets returned from **pV_GetPointer**, **pV_GetExtract** and **pV_GetSub** is in the form of pointers to blocks of memory. There are two modes that the pointers may be handled:

- %val

  A mechanism exists on all major workstation's FORTRAN (but NOT on CRAYs) to allow the pointer to be passed to a SUBROUTINE or FUNCTION and then have the memory treated as a normally declared vector or array. This is done by the VAX extension '%val(pointer)' used in the CALL or function invocation. When the pointer is passed to the sub-program by 'value', it is equivalent to passing a variable by 'reference' (the FORTRAN method). That is, in both cases, the address of the memory of interest is placed in the stack!

  In this case, the pointers are treated as INTEGER variables. This causes a problem in portability. The pointers must be treated as INTEGER*4 variables on all 32 bit machines. On 64 bit machines (ALPHAs and SGIs with R8000, R10000 or R12000 chips, in native mode) the pointers are INTEGER*8.

- POINTER statment

  The POINTER statement supported by most FORTRANs removes the portability problem. This syntax allows the coupling of a pointer with an array.

Using either mechanism, sophisticated **pV3** enhancements may be performed using FORTRAN. To complete this picture, three additional entries to the server have been added to allow the FORTRAN programmer to allocate and free up blocks of memory.

3.1 MALLOCPV

PTR = MALLOCPV(NBYTES)
This function is equivalent to the C routine 'malloc'. It allocates a block of memory and returns the pointer to the block. On 64 bit machines, this function may need to be declared INTEGER*8.

NBYTES:i: I	The number of bytes to allocate.
PTR:o: I	The address of the block (0 is an error indicator).

3.2 FREEV3

FREEPV(PTR)
This function is equivalent to the C routine 'free'. It deallocates a block of memory. NOTE: Only free up blocks of memory that YOU allocate!

PTR:i: I	The address of the block.

3.3 REALLOCPV

PTRN = REALLOCPV(PTRO, NBYTES)

This function is equivalent to the C routine ‘realloc’. It re-allocates a block of memory and returns the pointer to the new block. On 64 bit machines, this function may need to be declared INTEGER*8.

PTRO:i: I	The address of the block.
NBYTES:i: I	The number of bytes to allocate.
PTRN:o: I	The address of the new block (0 is an error indicator).

4 Server Side Extracts

The following section describes the internal data stored in the **pV3** server suite (which includes *pV3Server, pV3Batch* and *pV3Viewer*). This is the data used to produce the graphics objects that get rendered to make the scene. Each tool generates a different type of *extract* from the 3D data in the client(s). The data gets transmitted to the server and is stored for as long as it is needed (or is written to disk in the case of the batch server). Each *extract* consists of a number of *sub-extract* types, and there is a complete collection of *sub-extracts* for each client. Note: each client's data is stored separately.

4.1 Surfaces

This data is generated by the **pV3** scalar tools (planar cuts, structured block planes, programmed cut surfaces, iso-surfaces and domain surfaces). This data is exposed so that new 'probes' may be easily generated. The size of many of these arrays (and therefore the pointers) will change during the execution of **pV3**, so when using this data, get the current pointers before accessing the memory.

Extract	Type	Valid Sub-Extracts
2	Planar Cut	0 1 2 3 4 5 6 7 9 10*
3	Block Plane	1 2 3 4 5 6 7 8 9 11
4	Geometric Cut	0 1 2 3 4 5 6 7 8 9 10 11
5	Domain Surface	0 1 2 3 4 5 6 7 8 9 10 11
7	Iso-Surface	0 1 2 3 4 5 6 7 10 11

4.1.1 0 - Surface Sub-Extract *Tris*

The following data defines the disjoint triangle space. Where the number of triangles in the structure is KTRI.

TRIS: I(4,KTRI) disjoint triangle definitions.

TRIS(1,n) = first node index for the triangle.

TRIS(2,n) = second node index for the triangle.

TRIS(3,n) = third node index for the triangle.

TRIS(4,n) = the parent 3D cell number (in the client).

4.1.2 1 - Surface Sub-Extract *Quads*

The following data defines the disjoint quadrilateral space. Where the number of quadrilaterals in the structure is KQUAD.

QUADS: I(5,KQUAD)	disjoint quadrilateral definitions.
	QUADS(1,n) = first node index for the quadrilateral.
	QUADS(2,n) = second node index for the quadrilateral.
	QUADS(3,n) = third node index for the quadrilateral.
	QUADS(4,n) = fourth node index for the quadrilateral.
	QUADS(5,n) = the parent 3D cell number (in the client).

4.1.3 2 - Surface Sub-Extract *XYZ*

The following data defines the 3D coordinates for the nodes (and therefore also the number of nodes) that support the surface. The number of nodes in the structure is KXYZ. If KXYZ is 1 this is usually the indication of a sub-extract place-holder and not actual data.

XYZ: R(3,KXYZ)	(x, y, z)-coordinates for the nodes.

4.1.4 3 - Surface Sub-Extract *Mesh*

The following data defines the disjoint lines that make-up the intersection of the cell edges and the cutting surface. The number of line segments in the structure is KFACE.

FACE: I(2,KFACE)	disjoint line definitions.
	FACE(1,n) = first node index for the line.
	FACE(2,n) = second node index for the line.

4.1.5 4 - Surface Sub-Extract *Outline*

The following data defines the disjoint lines that make-up the outline of the surface. The number of line segments in the structure is KEDGE.

EDGE: I(3,KEDGE)	disjoint line definitions.
	EDGE(1,n) = first node index for the line.
	EDGE(2,n) = second node index for the line.
	EDGE(3,n) = the parent surface face number (in the client).

4.1.6 5 - Surface Sub-Extract *Scalar*

The following data defines the current scalar for the nodes (and therefore also the number of nodes) that support the surface. The number of nodes in the structure is KS and is the same as KXYZ.

S: R(KS)	scalar functional values for the nodes.

4.1.7 6 - Surface Sub-Extract *Vector*

The following data defines the current vector for the nodes (and therefore also the number of nodes) that support the surface. The number of nodes in the structure is KV and is the same as KXYZ.

V: R(3,KV)	vector values (Vx, Vy, Vz) for the nodes.

4.1.8 7 - Surface Sub-Extract *Threshold*

The following data defines the current threshold values for the nodes that support the surface. The number of nodes in the structure is KT and is the same as KXYZ.

T: R(KT)	threshold functional values for the nodes.

4.1.9 8 - Surface Sub-Extract *2D Mapping*

The following data defines the 2D mapping for the nodes that support the surface. The number of nodes in the structure is KXY and is the same as KXYZ.

XY: R(2,KXY)	raw (x', y')-coordinates as specified by the client.

Notes:
(1) The 2D mapping for planar cuts is implicit and not required from the client.
(2) There is no 2D mapping for iso-surfaces.

4.1.10 9 - Surface Sub-Extract *Transformed 2D Mapping*

The following data defines the 2D mapping used to draw the surface into the 2D window, where the viewed range is between -1.0 and 1.0 in both x' and y'. This is a simple transformation of XY. The number of nodes in the structure is KXYP and is the same as KXYZ.

XYP: R(2,KXYP)	transformed (x', y')-coordinates.

Notes:
(1) This data is generated at the server and not transfered from the clients.
(2) If KYXP is zero, the transformation has not yet been computed.
(3) There is no 2D mapping for iso-surfaces.

4.1.11 10 - Surface Sub-Extract *Tri normals*

The following data defines the normals used (for each disjoint triangle). This data is used in the lighting model for drawing the surface in the 3D window. The number of normals in the structure is KTRIN and is the same as KTRI.

TRIN: R(3,KTRIN)	(x, y, z)-normals for the triangle.

Notes:
(1) This data is generated at the server and not transfered from the clients.
(2) If KTRIN is zero, the normals have not yet been computed.
(3) The normals for planar cuts are implicit and the same for all triangles and therefore this sub-extract is not used (but the index is reused for clipping).

4.1.12 11 - Surface Sub-Extract *Quad normals*

The following data defines the normals used for each disjoint quadrilateral. This data is used in the lighting model for drawing the surface in the 3D window. The number of normals in the structure is KQUADN and is the same as KQUAD.

QUADN: R(3,KQUADN)	(x, y, z)-normals for the quadrilateral.

Notes:
(1) This data is generated at the server and not transfered from the clients.
(2) If KQUADN is zero, the normals have not yet been computed.
(3) The normals for planar cuts are implicit and the same for all quadrilaterals and therefore this sub-extract is not used.

4.1.13 10* - Surface Sub-Extract *Planar clipping*

The following data defines the clipping index used for each node. The number of entries in the structure is KCLIP and is the same as KXYZ.

CLIP: I(KCLIP)	0 - not clipped, otherwise the following are additive:
	1 - transformed x' less than -1.0
	2 - transformed x' greater than 1.0
	4 - transformed y' less than -1.0
	8 - transformed y' greater than 1.0

Notes:
(1) This data is generated at the server and not transfered from the clients.
(2) If KCLIP is zero, the clipping index has not yet been computed.
(3) This sub-extract index is only used for planar cuts so that there is no conflict with triangle normals.

4.2 StreamLines

This data is generated by the **pV3** clients during the integration of instantaneous streamlines. The size of many of these arrays (and therefore the pointers) will change during the execution of **pV3**, so when using this data, get the current pointers before accessing the memory. Unlike all other Extracts, the number of sub-extracts is not a function of the number of clients but of the maximum allotted streamline segments (that is greater than the number of clients). This allows a streamline to reenter a client more than once.

4.2.1 0 - StreamLine Sub-Extract *Cell*

The following data contains the 3D cell number for the position of the point for this segment (used for the point probe). The number of entries in the structure is KCELL and is the same as KXYZ.

CELL: I(KCELL) the parent 3D cell number (in the client).

4.2.2 1 - StreamLine Sub-Extract *Time*

The following data defines the integration pseudo-time for the point (used for streamline animation). Where the number of elements in the structure is KTIME and is the same as KXYZ.

TIME: R(KTIME) integration time (from the seed position).

4.2.3 2 - StreamLine Sub-Extract *XYZ*

The following data defines the 3D coordinates for the points that support this poly-line segment. The number of nodes in the structure is KXYZ.

XYZ: R(3,KXYZ) (x, y, z)-coordinates for the points.

4.2.4 3 - StreamLine Sub-Extract *Div*

The following data defines the cross-flow divergence felt by each point during the integration. Where the number of elements in the structure is KDIV and this is the same as KXYZ.

DIV: R(KDIV) used for streamtube rendering, where the size of the tube is based on a starting size mutiplied by **e** to this power.

4.2.5 4 - StreamLine Sub-Extract *Angle*

The following data contains the curl for each point, calculated during the integration, in this segment of the streamline Where the number of entries in the structure is KANG and this is the same value as KXYZ.

ANG: R(KANG) angle of the twist for ribbons in degrees.

4.2.6 5 - StreamLine Sub-Extract *Scalar*

The following data defines the current scalar for the points that support the line in this segment. The number of points in the structure is KS and this is the same as KXYZ.

S: R(KS) scalar functional values for the points.

4.2.7 6 - StreamLine Sub-Extract *Vector*

The following data defines the current vector for the points that make up this segment of the streamline. The number of elements in the structure is KV and this is the same as KXYZ.

V: R(3,KV) vector values (Vx, Vy, Vz) for the points.

4.2.8 7 - StreamLine Sub-Extract *Threshold*

The following data defines the current threshold values for the points that support the poly-line. The number of entries in the structure is KT and is the same as KXYZ.

T: R(KT) threshold functional values for the points.

4.3 Particles

This data is updated by the **pV3** clients during the bubble integration at each time-step. The size of many of these arrays (and therefore the pointers) will change during the execution of **pV3**, so when using this data, get the current pointers before accessing the memory.

4.3.1 0 - Particle Sub-Extract *Number*

The following data contains the unique particle number for each bubble in that client. The number of entries in the structure is KNUM and this is the same as KXYZ.

NUM: I(KNUM) the global particle number.

4.3.2 1 - Particle Sub-Extract *Time*

The following data defines the start time for each bubble. The number of elements in the structure is KTIME and this number is the same as KXYZ.

TIME: R(KTIME) bubble simulation time when the particle was seeded.

4.3.3 2 - Particle Sub-Extract *XYZ*

The following data defines the current 3D coordinates for the particles. The number of nodes in the structure is KXYZ.

XYZ: R(3,KXYZ) (x, y, z)-coordinates for the bubbles.

4.3.4 3 - Particle Sub-Extract *Div*

The following data defines the cross-flow divergence currently felt by each bubble. Where the number of elements in the structure is KDIV and this is the same as KXYZ.

DIV: R(KDIV) optionally used for bubble rendering, where the size of the particle is based on a starting size mutiplied by **e** to this power.

4.3.5 5 - Particle Sub-Extract *Scalar*

The following data defines the current scalar for the particles in this client. The number of points in the structure is KS and this is the same as KXYZ.

S: R(KS) scalar functional values for the bubbles.

4.3.6 6 - Particle Sub-Extract *Vector*

The following data defines the current vector for the particles. The number of elements in the structure is KV and this number is the same as KXYZ.

V: R(3,KV) vector values (Vx, Vy, Vz) for the bubbles.

4.3.7 7 - Particle Sub-Extract *Threshold*

The following data defines the current threshold values for the particles. The number of entries in the structure is KT and is the same as KXYZ (the number of bubbles).

T: R(KT) threshold functional values for the bubbles.

4.3.8 10 - Particle Sub-Extract *Group Index*

The following data defines the current group number for the particles. The number of entries in the structure is KGI and is the same as KXYZ (the number of bubbles).

GI: I(KGI) group index for the bubbles (used for *time lines*).

4.4 Vector Clouds

4.4.1 2 - VC Sub-Extract *XYZ*

The following data defines the coordinates for the 3D nodes that satisfy the threshold limits within each client. The number of nodes in the structure is KXYZ.

XYZ: R(3,KXYZ) (x, y, z)-coordinates for the vector cloud.

4.4.2 5 - VC Sub-Extract *Scalar*

The following data defines the current scalar for the vector cloud The number of points in the structure is KS and this number is the same as KXYZ.

S: R(KS) scalar functional values for the 3D nodes.

4.4.3 6 - VC Sub-Extract *Vector*

The following data defines the current vector for each node in the client that satisfies the threshold limits. The number of elements in the structure is KV and this number is the same as KXYZ.

V: R(3,KV) vector values (Vx, Vy, Vz) for the vector cloud.

4.5 Points

This data is updated by the **pV3** clients after a Point extract is created. See the **pV3** Server Builder's Guide.

4.5.1 0 - Point Sub-Extract *Index*

The following data contains the index (bias 1) from the original list for this point. The number of entries in the structure is KINDX and this is the same as KXYZ.

INDX: I(KINDX) the point index.

4.5.2 1 - Point Sub-Extract *Cell Index*

The following data contains the cell index in the client that contains the point. The number of entries in the structure is KICEL and this is the same as KXYZ.

ICEL: I(KICEL) the cell index.

4.5.3 2 - Point Sub-Extract *XYZ*

The following data defines the coordinates for the 3D points requested from within each client. The number of nodes in the structure is KXYZ.

XYZ: R(3,KXYZ) (x, y, z)-coordinates for the points.

4.5.4 3 - Point Sub-Extract *Local Block Index*

The following data contains the clients local block index (bias 1) for this point. An index of zero is an indication that the point is not in a structured block. The number of entries in the structure is KLNIN and this is the same as KXYZ.

LBIN: I(KLBIN) the structured block index.

4.5.5 4 - Point Sub-Extract *IJK Indices*

The following data defines the IJK (with fraction) for the points requested from within each client. This contains valid data if the point is contained within a structured block. The indices are local to the clients numbering. The number of entries in the structure is KIJK and this is the same as KXYZ.

IJK: R(3,KIJK) (I, J, K)-indices for the points.

4.5.6 5 - Point Sub-Extract *Scalar*

The following data defines the current scalar for the point extract. The number of points in the structure is KS and this number is the same as KXYZ.

S: R(KS)	scalar functional values for the requested points.

4.5.7 6 - Point Sub-Extract *Vector*

The following data defines the current vector for each node in the client. The number of elements in the structure is KV and this number is the same as KXYZ.

V: R(3,KV)	vector values (Vx, Vy, Vz) for the points.

4.5.8 7 - Point Sub-Extract *Threshold*

The following data defines the current threshold values for the requested points. The number of nodes in the structure is KT and is the same as KXYZ.

T: R(KT)	threshold functional values for the points.

5 Server Suite Supplied Routines and Calls

Consistant with the **pV3** naming convension, the routines that are part of **pV3**'s server suite are prefixed with 'pV_', those that are supplied by the programmer start with 'pV'.

5.1 pVEvents

PVEVENTS(IWIN,TYPE,XE,YE,STATE)

This subroutine is called by the **pV3** server or viewer immediately after an **X** event has been pulled of the queue. This routine may perform certain functions based on the event, change the event status, have the server (or viewer) ignore the event or just do nothing. Not called in the batch server – *pV3Batch*.

IWIN:i/o: I — The **X** window ID of the event or:

- **IWIN=-1** No more events to process, therefore pass the control of the server to the data collection phase
- **IWIN=0** Output only. Ignore this event and pull the next one off the queue

NOTE: this must be INTEGER*8 (or *long*) on 64 bit machines.

TYPE:i/o: I — The event type. The event types that **pV3** windows can generate are:

- **TYPE=2/3** KeyPress/KeyRelease
- **TYPE=4/5** ButtonPress/ButtonRelease
- **TYPE=12** Expose
- **TYPE=14** NoExpose
- **TYPE=33** **X** ClientEvent

XE:i/o: I — The X pixel location in the window of the event (0 is the left)

YE:i/o: I — The Y pixel location in the window of the event (0 is the top of the window)

STATE:i/o: I — The event state. For Key events, the state is the **X** KeySym number (usually the ASCII code except for the special keys). For mouse button events, the state is the button number.

5.2 pV_Cursor

PV_CURSOR(FLAG)

This subroutine can be used by certain programmer-supplied routines when the programmer wants to add a function which needs user input from the text window. Should not be called from the batch server.

FLAG:i: L — **FLAG=.TRUE.** Move cursor to text window

FLAG=.FALSE. Move cursor back to previous window

5.3 pV_GetPointer

PV_GETPOINTER(OPT,PTR)

Returns the internal **pV3** server structure. This routine can be called from any programmer-supplied code in the interactive server or post-processing viewer.

OPT:i: I	Option set to specify what data to get.
PTR:o: I	Integer (in FORTRAN) used as a pointer to the structure. FORTRAN Note: this must be INTEGER*8 on 64-bit machines.

- OPT = 0 - **X** Event Structure:
  PTR = *XEvent - Useful for building Graphical User Interfaces (GUIs) on the **pV3** server or viewer. Allows passing the event information to tool-kits such as Motif. This need only be called once. The same structure is used for all event handling.
- OPT = 1 - **X** window structure for the 1D Window
- OPT = 2 - **X** window structure for the 2D Window
- OPT = 3 - **X** window structure for the 3D Window
- OPT = 4 - **X** window structure for the Dails Window
- OPT = 5 - **X** window structure for the Key Window
- OPT = 6 - **X** window structure for the Text Window
- OPT = 7 - **X** pointer for the Comparison Window (0 means the window is not open)
- OPT = 8 - **X** window structure for the Root Window
- OPT = 9 - **X** pointer for the Servers Window (0 means the window is not open)
  PTR = *Window - for the appropriate Window

5.4 pV_GetState

PV_GETSTATE(OPT,IVEC,RVEC,STRING)
Returns the internal **pV3** state. This **pV3** server suite routine can be called from any programmer-supplied code linked with the server, batch server or viewer.

OPT:i: I	Option set to specify what data to get
IVEC:i/o: I()	Integer data returned based on OPT (length also determined by OPT). Some OPTs control actions on input.
RVEC:o: R()	Real data returned based on OPT (length also determined by OPT)
STRING:o: C	Character data returned based on OPT

NOTE: The symbol † indicates global data (not discipline specific) in multi-disciplinary runs. All non-marked options are related to the current discipline.

- OPT = -8 - Get client discretization:

  IVEC(1) = Client index - Input (1 to total number of clients)

  IVEC(2) = NNODE - Number of nodes

  IVEC(3) = KCEL1 - Number of tets

  IVEC(4) = KCEL2 - Number of pyramids

  IVEC(5) = KCEL3 - Number of prisms

  IVEC(6) = KCEL4 - Number of hexas

  IVEC(7) = KNPTET - Number of polytet strips

  IVEC(8) = KPTET - Number of polytet cells

  IVEC(9) = KNBLK - Number of structured blocks

  IVEC(10) = KPHEDRA - Number of complex polyhedra

- OPT = -7 - Get client block sizes:

  IVEC(1) = Client index - Input (1 to total number of clients)

  IVEC(2) = Block index - Input (1 to KNBLK from OPT = -8)

  IVEC(3) = Ni

  IVEC(4) = Nj

  IVEC(5) = Nk

- OPT = -6 - Get programmer-defined cut data:

  IVEC(1) = Cut index - Input (1 to total number of cuts, 0 returns only IVEC(2))

  IVEC(2) = NPGCUT - total number of programmer-defined cuts

  STRING = TPGCUT - title for cut indexed by IVEC(1)

- OPT = -5 - Get field variable data:

  IVEC(1) = Field Variable Index - Input (1 to total number of field variables, 0 returns only IVEC(4))

  IVEC(2) = FKEYS - function type for field variable indexed by IVEC(1)

  IVEC(3) = IKEYS - X-keypress for activation

  IVEC(4) = NKEYS - total number of field variables

  RVEC(1-2) = FLIMS - limits for variable indexed by IVEC(1)

  STRING = TKEYS - title for the field variable indexed by IVEC(1)

- † OPT = -4 - Get 3D cursor data:

  IVEC(1) = Discipline ID

  IVEC(2) = Client index - 1 to total number of clients

  IVEC(3) = Cell index - the cell index in the client

  IVEC(4) = Mask - -1 for no data, or:

  0 - No Scalar or Vector

  1 - Scalar

  2 - Vector

  3 - Scalar and Vector

  RVEC(1-3) = XYZ position

  RVEC(4) = Time - simulation time for this data

  RVEC(5) = Scalar value

  RVEC(6-8) = Vector values

- OPT = -3 - Get client data:

  IVEC(1) = Client index - input (1 to total number of clients)

  IVEC(2) = PVM tid - the **PVM** task id of the client index

  IVEC(3) = IOPT - unsteady option for the client index

  IVEC(4) = Number of clients - the total number of clients

  IVEC(5) = State-Vector rank - 0 for no state vector

  STRING = Partition name - maximum of 20 characters used.

- † OPT = -2 - Get unsteady info:

  IVEC(1) = IOPT - maximum for all clients

  IVEC(2) = Pause State - 0 not paused, 1 paused, 2 single-step

  RVEC(1) = Time - simulation time currently displayed. This has no meaning for steady-state or *Non-Time Accurate* cases.

  STRING = SubTitle - the subtitle returned from the clients.

- † OPT = -1 - Get special key bindings. IVEC is filled with 32 integers that are the **X** KeySyms for the non-ASCII keys used by **pV3**. A file 'KeyBoard' contains the default **pV3** server KeySyms and the associated labels. These values may be modified and used by the server if the environment variable 'Visual_KB' is set to point to the file containing the new bindings.

- † OPT = 0 - Get Rev number:

  IVEC(1) = pV3 flag always -1

  IVEC(2) = pV3 type :

  0 - Interactive Server

  1 - Batch Server

  2 - Post-Processing Viewer

  3 - Programmed Server

  RVEC(1) = Revision Number

- † OPT = 1 - Get the current transformation matrix for the 3D view. 12 words of RVEC are used.

- † OPT = 2 - Get the current planar cut coordinates. 9 words of RVEC are used for 3 of the 4 corners of the cut plane in 3 space.

- OPT = 3 - Get the current scalar tool status.

  IVEC(1) = Current Tool :

  0 - No tool

  1 - Structured block plane

  2 - Planar cut

  3 - Planar cut positioning on

  4 - Programmed cut

  5 - Mapped surface

  6 - Mapped surface with surface functions

  7 - Iso-surface

  IVEC(2-8) = Block Indces - for 1 above

  IVEC(2) = Cut Index - for 4 above

  RVEC(1) = Current ZPRIME - for 2, 4 and 7 above

  RVEC(2) = XPC - for 1, 4, 5 and 6

  RVEC(3) = YPC - for 1, 4, 5 and 6

  RVEC(4) = HALFW - for 1, 4, 5 and 6

  RVEC(5) = 2D Rotation Angle in degrees - for 1, 4, 5 and 6

  RVEC(6) = Zmin - for 2, 3, 4, and 7 above

  RVEC(7) = Zmax - for 2, 3, 4, and 7 above

  RVEC(8) = Time - Time when RVEC(6) and RVEC(7) were last updated.

- † OPT = 4 - Lighting and Mirroring state:

  IVEC(1) = Mirror status :

  0 - No Mirroring

  1 - Mirror about X = 0.0

  2 - Mirror about Y = 0.0

  3 - Mirror about Z = 0.0

  IVEC(2) = Number of Lights maximum is 8

  RVEC(1-3) = Ambient light color (r,g,b) - values between 0.0 and 1.0

- OPT = 5 - Scalar state:

  IVEC(1) = Binding index

  RVEC(1) = Minimum for display - fmin shown in the key window

  RVEC(2) = Maximum for display - fmax shown in the key window

  RVEC(3) = Minimum scalar function for the entire node space

  RVEC(4) = Maximum scalar function for the entire node space

  RVEC(5) = Time RVEC(3) and RVEC(4) last updated

  STRING = Scalar title - maximum of 32 characters used

- OPT = 6 - Vector state:

  IVEC(1) = Binding index

  RVEC(1) = Vector scale - the scale factor for tufts and arrows

  RVEC(2) = Maximum Vector Magnitude for the entire node space

  RVEC(3) = Time RVEC(2) last updated

  STRING = Vector title - maximum of 32 characters used.

- OPT = 7 - Threshold state:

  IVEC(1) = Binding index - negative value indicates the appropriate scalar index

  IVEC(2) = Vector Cloud status - 0 = off, 1 = on

  IVEC(3) = Dynamic Threshold status - 0 = off, 1 = on

  RVEC(1) = Minimum for display - tmin shown in the key window

  RVEC(2) = Maximum for display - tmax shown in the key window

  RVEC(3) = Minimum scalar function for the entire node space

  RVEC(4) = Maximum scalar function for the entire node space

  RVEC(5) = Time RVEC(3) and RVEC(4) last updated

  STRING = Threshold title - maximum of 32 characters used.

- † OPT = 8 - User response status:

  IVEC(1) = First mouse button action - 0 = no button press

  IVEC(2) = X pixel location for action

  IVEC(3) = Y pixel location for action

  IVEC(4) = Second mouse button action

  IVEC(5) = X pixel location for action

  IVEC(6) = Y pixel location for action

  STRING = Text answer

  The data in IVEC is valid for those operations that require snapping a line/circle or a 'rubber-band' box.

- † OPT = 9 - Light Status:

  IVEC(1) = Light number - input (1 to the number of lights)

  RVEC(1-3) = Light color (r,g,b) - values between 0.0 and 1.0

  RVEC(4-6) = Light normal

- OPT = 10 - Current domain/static surface state:

  IVEC(1) = Surface index

  IVEC(2) = Current drawing state - see the Appendix

  IVEC(3) = Surface type :

  2 - Planar cut
  RVEC is filled with the 9 words (for the corners) at the time of the cut. This is like OPT = 2.

  4 - Programmer-defined cut
  RVEC(1) contains the ZPrime value.

  5 - Domain surface

  7 - Iso-surface
  RVEC(1) contains the iso-surface value.

  IVEC(4) = Maximum number of surfaces

  STRING = Surface title - maximum of 20 characters used.

- OPT = 11 - Current streamline state:

  IVEC(1) = Current streamline group number - 0 no streamlines active

  IVEC(2) = Current drawing state - See Appendix

  IVEC(3) = Stream group number

  IVEC(4) = Maximum number of streamline groups

  IVEC(5) = Global Surface number - 0 indicates a volume streamline

  RVEC(1) = Ribbon rotation in degrees

  RVEC(2) = Ribbon/Tube size

  RVEC(3) = Tube limitter

  RVEC(4) = Time limitter - 0.0 indicates no limitting

  STRING = Streamline title - maximum of 20 characters used.

- OPT = 12 - Client visibility state:

  IVEC(1) = Client index - input

  IVEC(2) = Visibility state - 0 is off / 1 is on.

  STRING = Partition name - maximum of 20 characters used.

- OPT = 13 - Current probe state:

  IVEC(1) = 1D plot type :

  0 - no plot
  1 - StreamLine Probe
  2 - Strip Chart
  3 - Line Probe
  4 - Edge Plot
  5 - Surface Layer Scan
  6 - Programmer-defined probe

  IVEC(2) = Point Probe status - 0 (off) or 1 (on)

  IVEC(3) = Reference line status - 0 (off) or 1 (on)

  RVEC(1) = X Minimum for plot

  RVEC(2) = X Maximum for plot

  RVEC(3) = Y Minimum for plot

  RVEC(4) = Y Maximum for plot

- OPT = 14 - Global drawing state:

  IVEC(1) = Edge Outline status - 0 (off), 1 (on - default), 2 (no internal boundary edges)

  IVEC(2) = Handedness † -1 (left - default), 1 (right handed)

  IVEC(3) = Transparent Plane † 0 (no plane for the planar cut), 1 (display the plane - default)

  IVEC(4) = Outline 3D Rendering † 0 (normal 3D drawing - default), 1 (only the outline)

  RVEC(1) = Object Radius † The radius of the sphere that encloses the object(s).

  RVEC(2) = Line Adjustment † The amount that lines are displaced (towards the viewer) so that the surfaces do not obscure them.

  RVEC(3) = Screen Z Clip value † data added to screen Z to move object(s) into the front or back clipping plane.

  RVEC(4) = Feature Line Dot - The value of the dot-product between Domain Surface facet normals below which the shared edge is displayed when the Feature Line attribute is set. Default: 0.0 - no Feature Lines, maximum value is 1.0 (same as turing the enitire on).

- OPT = 15 - Dynamic surface and Arrow state:

  IVEC(1) = Current drawing state - see the Appendix

  IVEC(2) = Number of tufts - the number of tufts on a side

  IVEC(3) = 3D Arrow representation :

  0 - Line segment

  1 - Line segment with colored head

  2 - Open head

  3 - Closed head

  RVEC(1) = Vector scale - the scale factor for tufts and arrows

  RVEC(2) = Vector scale for surface functions - the scale factor used with tufts and arrows for special surface functions

  RVEC(3) = 3D Arrow head length ratio

  RVEC(4) = 3D Arrow head spread ratio

- OPT = 16 - StreamLine/Bubble drawing state:

  IVEC(1) = StreamLine representation (when seeded):

  0 - StreamLine
  1 - Ribbon
  2 - StreamTube
  3 - Tubes with twist

  IVEC(2) = Bubble rendering :

  0 - off
  1 - Rendered in foreground color
  2 - Rendered using scalar color
  3 - Rendered with surface function (for Surface particles only)

  IVEC(3) = Current Surface/Volume setting :

  0 - Volume Scalar/Volume Vector
  1 - Surface Scalar/Volume Vector
  2 - Volume Scalar/Surface Vector
  3 - Surface Scalar/Surface Vector

- OPT = 17 - Key and Colormap state:

  IVEC(1) = Key status :

  0 - Scalar
  1 - Special Surface Scalar
  2 - † Time

  IVEC(2) = Number of entries in colormap

  IVEC(3) = Number of entries in background colormap - between 1 and 4

- OPT = 18 - Get colormap entry in current Key colormap:

  IVEC(1) = index - set on input:

  -2 - Foreground: used for outlines and pseudo-cursors, usually white
  -1 - Midground: used for 1D cursor and solid surfaces, usually grey
  0 - Background: used for Dial and 1D window background, usually black
  1-on - Colormap/background colormap entry. The background colormap indices follow the colormap entries.

  RVEC(1-3) = The color (r,g,b) - values between 0.0 and 1.0

- OPT = 19 - Contouring state:

  IVEC(1) = Dynamic contour status - 0 = off, 1 = on

  IVEC(2) = Number of contour levels

- OPT = 20 - Surface Scalar state:

  IVEC(1) = Binding index

  RVEC(1) = Minimum for display - fsmin shown in the key window

  RVEC(2) = Maximum for display - fsmax shown in the key window

  STRING = Scalar title - maximum of 32 characters used.

- OPT = 21 - Surface Vector state:

  IVEC(1) = Binding index

  RVEC(1) = Vector scale - the scale factor for tufts and arrows

  STRING = Vector title - maximum of 32 characters used.

- † OPT = 22 - Discipline state:

  IVEC(1) = current discipline index - 0 to the number of disciplines -1

  IVEC(2) = number of disciplines

  IVEC(3) = discipline index for rendering - -1 indicates that no rendering is in progress

  RVEC(1-3) = offset - default at $(0.0, 0.0, 0.0)$

  RVEC(4) = scale - default at 1.0

  STRING = discipline title - maximum of 20 characters used.

- OPT = 23 - Replication state:

  IVEC(1) = Client index - input (1 to the number of clients)

  IVEC(2) = Instance state - for plotting:

  n - plot n additional objects
  0 - allow the use of global mirroring
  -1 - Mirror about X = 0.0 for this client
  -2 - Mirror about Y = 0.0 for this client
  -3 - Mirror about Z = 0.0 for this client
  -4 - no mirroring or instancing

  IVEC(3) = Total number of replicants - 0 indicates that there can be no replication

  RVEC(1-16) = Replication Matrix - Any rotation and translation of objects can be defined via this 4x4 transformation matrix. It allows pure rotation like in axial turbomachinery or any combination of translation and/or rotation. This matrix is defined at the client.

- OPT = 24 - Replication mask:

  IVEC(1) = Client index - input (1 to the number of clients)

  IVEC(2) = Instance index - input (1 to n) from OPT = 23.

  IVEC(3) = Mask :

  0 - Do not plot this instance

  1 - Plot

5.5 pV_SetState

PV_SETSTATE(OPT,IVEC,RVEC,STRING)
Sets the internal **pV3** server module state. This routine can only be called from **pVEvents** or **pVSafe**.

OPT:i: I	Option set to specify what internal state data to change
IVEC:i: I()	Integer data set based on OPT (length also determined by OPT)
RVEC:i: R()	Real data set based on OPT (length also determined by OPT)
STRING:i: C	Character data set based on OPT

NOTE: The symbol † indicates global data (not discipline specific) in multi-disciplinary runs. All non-marked options relate to the current discipline or the discipline index is explicitly defined.

- OPT = -3 - Send the string in STRING to all clients.
- † OPT = -2 - Force the simulation time to the value found in RVEC(1). Post-Processing Viewer only.
- † OPT = -1 - Force a 3D and 2D window update. This tells **pV3** to do a complete redraw including the static objects.
- OPT = 0 - Obsolete. See **pV_GetState** and **pV_SetState** with OPT = 23.
- † OPT = 1 - Set the current transformation matrix for the 3D view. 12 words of RVEC are used.
- † OPT = 2 - Set the current planar cut coordinates. 9 words of RVEC are used for 3 of the 4 corners of the cut plane in 3 space.
- OPT = 3 - Set the current scalar tool values:

  IVEC(1) = Tool Flag - If this value is positive and no other scalar tool is on, this indicates that a structured block plane is desired. This number is the global block number. If this number is negative (and the negated value of the current global block number) then the tool is turned off.

  IVEC(2-7) = Block Indices - These are the values for I_{min}, I_{max}, J_{min}, J_{max}, K_{min} and K_{max} respectively. One pair must contain equal indices – this is the plane that will be spanned by the other indices. If a $_{min}$ value is set to zero it will be internally reset to one. Any $_{max}$ that is zero indicates the maximum extent of that block index range.

  RVEC(1) = ZPRIME

  RVEC(2) = XPC

  RVEC(3) = YPC

  RVEC(4) = HALFW

  RVEC(5) = 2D Rotation Angle in degrees

NOTES:
1) If you don't wish to change all of the above call pV_GetState(3,IVEC,RVEC,STRING) first!
2) Set IVEC(1) = 0 for any non-structured block interactions.

- † OPT = 4 - Set Lighting and Mirroring state:

  IVEC(1) = Mirror status :

  0 - No Mirroring

  1 - Mirror about X = 0.0

  2 - Mirror about Y = 0.0

  3 - Mirror about Z = 0.0

  IVEC(2) = Number of Lights maximum is 8

  RVEC(1-3) = Ambient light color (r,g,b) - values between 0.0 and 1.0

- OPT = 5 - Scalar state:

  IVEC(1) = Binding index - allows the changing of the current scalar function.
  A negative value indicates that the current scalar is not changed but the (absolute value of) the index should be reloaded at the clients and the min and max recalculated. Can also be used with threshold fields.

  RVEC(1) = Minimum for display - fmin shown in the key window

  RVEC(2) = Maximum for display - fmax shown in the key window

  STRING = Scalar title - up to 32 characters used (blank indicates no change)

- OPT = 6 - Vector state:

  IVEC(1) = Binding index - allows the setting of the current vector field.
  A negative value indicates that the current vector is not changed but the (absolute value of) the index should be reloaded and the range recalculated.

  RVEC(1) = Vector scale - the scale factor for tufts and arrows

  STRING = Vector title - up to 32 characters used (blank indicates no change)

- OPT = 7 - Threshold state:

  IVEC(1) = Binding index - negative value indicates the appropriate scalar index

  IVEC(2) = Vector Cloud status - 0 = off, 1 = on

  IVEC(3) = Dynamic Threshold status - 0 = off, 1 = on

  RVEC(1) = Minimum for display - tmin shown in the key window

  RVEC(2) = Maximum for display - tmax shown in the key window

  STRING = Threshold title - up to 32 characters used (blank string indicates no change).
  You cannot change the title of a scalar threshold with this function.

- † OPT = 8 - Enter user response:

  IVEC(1) = First mouse button action - 0 = no button press

  IVEC(2) = X pixel location for action

  IVEC(3) = Y pixel location for action

  IVEC(4) = Second mouse button action

  IVEC(5) = X pixel location for action

  IVEC(6) = Y pixel location for action

  STRING = Text answer

  The data in IVEC is required for those operations that snap a line/circle or a 'rubber-band' box.

- † OPT = 9 - Set Lights:

  IVEC(1) = Light number - 1 to the number of lights

  RVEC(1-3) = Light color (r,g,b) - values between 0.0 and 1.0

  RVEC(4-6) = Light normal

- OPT = 10 - Domain/static surface state:

  IVEC(1) = Surface index (-) value indicates that the numbered surface should be deleted without effecting what is the current surface

  IVEC(2) = Current drawing state - value < 0 doesn't change the state. See the Appendix.

  STRING = Surface title - maximum of 20 characters used (blank string indicates no change). Only for Domain surfaces.

- OPT = 11 - Streamline state:

  IVEC(1) = Streamline group index (-) value indicates that the numbered streamline group should be deleted without effecting what is the current streamline index

  IVEC(2) = Current drawing state - value < 0 doesn't change the state. See the Appendix.

  RVEC(1) = Ribbon rotation in degrees

  RVEC(2) = Ribbon/Tube size

  RVEC(3) = Tube limitter

  RVEC(4) = Time limitter - 0.0 indicates no limitting

  STRING = Streamline title - maximum of 20 characters used (blank string indicates no change). Warning: changing the name of a streamline may cause problems!

- OPT = 12 - Set client visibility state:

  IVEC(1) = Client index - input

  IVEC(2) = Visibility state - 0 is off / 1 is on.

  STRING = Partition name - maximum of 20 characters used.

- OPT = 13 - Set probe state:

  IVEC(1) = 1D plot type :

  0 - turn off plot

  1 - Streamline Probe

  6 - start Programmer-defined probe

  Note: use the proper 'events' for initiating the other probes.

- OPT = 14 - Set global drawing state:

  IVEC(1) = Edge Outline - 0 (off), 1 (on), 2 (no internal boundary edges)

  IVEC(2) = Handedness † -1 (left handed), 1 (right handed)

  IVEC(3) = Transparent Plane † 0 (no plane for the planar cut), 1 (display the plane)

  IVEC(4) = Outline 3D Rendering † 0 (normal 3D drawing), 1 (only the outline)

  RVEC(1) = Object Radius † The radius of the sphere that encloses the object(s).

  RVEC(2) = Line Adjustment † The amount that lines are displaced (towards the viewer) so that the surfaces do not obscure them.

  RVEC(3) = Screen Z Clip value † data added to screen Z to move object(s) into the front or back clipping plane.

  RVEC(4) = Feature Line Dot - The value of the dot-product between Domain Surface facet normals below which the shared edge is displayed when the Feature Line attribute is set. 0.0 - no Feature Lines, maximum value is 1.0.

- OPT = 15 - Set Dynamic surface and Arrow state:

  IVEC(1) = Current drawing state - see the Appendix

  IVEC(2) = Number of tufts - the number of tufts on a side

  IVEC(3) = 3D Arrow representation :

  0 - Line segment

  1 - Line segment with colored head

  2 - Open head

  3 - Closed head

  RVEC(1) = Vector scale - the scale factor for tufts and arrows

  RVEC(2) = Vector scale for surface functions - the scale factor used with tufts and arrows for special surface functions

  RVEC(3) = 3D Arrow head length ratio

  RVEC(4) = 3D Arrow head spread ratio

- OPT = 16 - Set StreamLine/Bubble drawing state:

  IVEC(1) = StreamLine seed representation :

  0 - StreamLine

  1 - Ribbon

  2 - StreamTube

  3 - Tubes with twist

  IVEC(2) = Bubble rendering :

  0 - off

  1 - Rendered in foreground color

  2 - Rendered using scalar color

  3 - Rendered with surface function (for Surface particles only)

  IVEC(3) = Current Surface/Volume setting :

  0 - Volume Scalar/Volume Vector

  1 - Surface Scalar/Volume Vector

  2 - Volume Scalar/Surface Vector

  3 - Surface Scalar/Surface Vector

- OPT = 17 - Set Key and Colormap state:

  IVEC(1) = Key status :

  0 - Scalar

  1 - Special Surface Scalar - if any Surface Scalars

  2 - † Time

  IVEC(2) = Number of entries in colormap

  IVEC(3) = Number of entries in background colormap - between 1 and 4

  This forces a Key Window redraw. When changing the colors, first set the Key state (Scalar, Surface Scalar or Time) if not already correct, then set the new colormap entries (calls with OPT = 18) and finally (again) make this call.

- OPT = 18 - Set colormap entry in current Key colormap:

  IVEC(1) = index :

  -2 - Foreground: used for outlines and pseudo-cursors - Key independent

  -1 - Midground: used for 1D cursor and solid surfaces - Key independent

  0 - Background: used for Dial and 1D window background - Key independent

  1-on - Colormap/background colormap entry. The background colormap indices follow the colormap entries.

  RVEC(1-3) = The color (r,g,b) - values between 0.0 and 1.0

- OPT = 19 - Set Contouring state:

  IVEC(1) = Dynamic contour status - 0 = off, 1 = on

  IVEC(2) = Number of contour levels - must be greater than 1

- OPT = 20 - Set Surface Scalar state:

  IVEC(1) = Binding index

  RVEC(1) = Minimum for display - fsmin shown in the key window

  RVEC(2) = Maximum for display - fsmax shown in the key window

  STRING = Scalar title - maximum of 32 characters used (blank string indicates no change)

- OPT = 21 - Set Surface Vector state:

  IVEC(1) = Binding index

  RVEC(1) = Vector scale - the scale factor for tufts and arrows

  STRING = Vector title - maximum of 32 characters used (blank string indicates no change)

- † OPT = 22 - Set Discipline state:

  IVEC(1) = Set discipline index - 0 to the number of disciplines -1. **pV3** must not be rendering for this to take effect.

  IVEC(2) = Redraw key and dials windows - 0 no redraw / 1 redraw (not to be called from PVSAFE).

  RVEC(1-3) = offset - default at $(0.0, 0.0, 0.0)$

  RVEC(4) = scale - default at 1.0

  NOTE: This should not be called while rendering is active.

- OPT = 23 - Set Replication state:

  IVEC(1) = Client index - 1 to the number of clients

  IVEC(2) = Instance state - for plotting:

  n - plot n additional objects (max is the number of replicants)
  0 - allow the use of global mirroring
  -1 - Mirror about X = 0.0 for this client
  -2 - Mirror about Y = 0.0 for this client
  -3 - Mirror about Z = 0.0 for this client
  -4 - no mirroring or instancing

- OPT = 24 - Set Replication mask:

  IVEC(1) = Client index - 1 to the number of clients

  IVEC(2) = Instance index - 1 to n (from OPT = 23).

  IVEC(3) = Mask :

  0 - Do not plot this instance

  1 - Plot

5.6 pVSafe

PVSAFE()

This programmer-supplied routine is called when the graphics thread of the server is stalled. This is the time where calls can be made that require neither thread to be active. The buffers have not been swapped, so that queries of extracts will look at the last state.

No Arguments

NOTES:

1) The first call to PVSafe is done when there is only one thread. This should be used to register all extracts (at least for the first allocation). This allows the 'batch' server to know what programmer-supplied data is going to be generated and therefore can enter this information in the output file header.

2) Calls to pV_SetState that cause any drawing (OPTs = 5, 7, 10, 11, 12, 17, 18, and 20) must be avoided.

5.7 pV_Register

PV_REGISTER(INDEX, NAME, SUBTYPE, SUBSIZE, SUBOPT, SUBLOC, ROUTINE, EXNUM)

Registers a programmer-defined extract with the **pV3** server. This routine should only be called when the threads are sync'ed, therefore the only valid place to execute this routine is within **pVSafe**. For multi-disciplinary cases, the discipline index must be set so that the extract is regestered within the appropriate discipline.

INDEX:i: I — The extract index. This number must be greater that 100 and defines an extract. Different disciplines must use unique extract indices!

NAME:o: C*20 — Extract name.

SUBTYPE:i: I(12) — The subextract types. Each extract is composed of up to 12 subextracts for each client. This vector defines whether the subextract is an integer (0) or a real (1).

SUBSIZE:i: I(12) — The subextract size per length. For example, if the subextract is for the 3D coordinates (X,Y,Z) that support the extract, the size would be 3.

SUBOPT:i: I(12) — The level of unsteadyness that requires the data at every time-step. Valid entries are 0 to 3 (where 3 is a special flag for a server-side only subextract). The following table specifies what action is taken with an existing subextract:

Client's OPT − >	0	1	2	3
SUBOPT = 0	leave	refill	refill	refill
SUBOPT = 1	leave	leave	refill	refill
SUBOPT = 2	leave	leave	leave	refill

SUBLOC:i: I(12) The subextract's locality. If this subextract comes from the clients then the value is -1. If this subextract is local to the server and it's length is set by another subextract, then SUBLOC must contain the index (0 biased) to that subextract. SUBOPTs for local subextracts must match that of the keyed subextract.

ROUTINE:i This is the programmer-supplied routine that will be called to render the data associated with this extract.
FORTRAN programmers must declare the subroutine 'EXTERNAL' in the calling module.

EXNUM:o: I This is a status return. If the value is zero or greater, that indicates success. The value is the number used for multiple allocations of extracts with the same INDEX (that use the same rendering routine). If the number is negative it is an indication of an error:

-1 - Invalid INDEX number

-2 - Invalid SUBTYPE in one of the entries

-3 - Invalid SUBSIZE in one of the entries

-4 - Invalid SUBOPT in one of the entries

-5 - Invalid SUBLOC in one of the entries

-6 - SUBTYPE mismatch for subsequent calls using INDEX

-7 - SUBSIZE mismatch for subsequent calls

-8 - SUBOPT mismatch for subsequent calls

-9 - SUBLOC mismatch for subsequent calls

-10 - ROUTINE mismatch for subsequent calls

-11 - Allocation error

-12 - Routine not called from **pVSafe**

-13 - SUBOPTs mismatch for local subextract

NOTES:
1) Having an extract that contains only subextracts that are SUBOPT=3 or SUBLOC>=0 will not request any data from the clients.
2) The length of a server-side subextract must be set by a call to **pV_SetSub**.

ROUTINE must have the form:

ROUTINE(ISTAT, EXNUM, PLOTMASK, CID, CNT,

LEN0,SUB0, ... LEN11,SUB11)

The programmer-supplied render routine for a registered extract. This routine is called for each active, visible **pV3** client. And, this routine will be called during the 3 stages of the 3D window's rendering. The drawing should be done by calls to **pV_Object3D**. See **pVDraw3D**.

ISTAT:i: I — Rendering stage:

0 - rendering for static object(s)

1 - rendering for dynamic object(s)

2 - rendering for translucent object(s)

EXNUM:i: I — The extract number.

PLOTMASK:i: I — The value that describes the drawing state of the extract.

CID:i: I — Client-id for this set of subextracts.

CNT:i: I — The client count (always starts at zero). Note: clients with visibility turned off are not included in the count.

LENn:i/o: I — The length of the subextracts. If the value is zero for a non-local extract then there is no data for this subextract. If the value is zero for a local subextract (and the keyed length is non-zero) then this routine can fill the subextract and set the length to the size of the keyed subextract. If the value is -1 for a local subextract, then there was some problem in allocation. Do NOT fill the subextract in this case!

SUBn:i/o — The subextracts. The size and type are determined by the call to register the extract.

5.8 pVSetExtract

PVSETEXTRACT(INDEX,EXNUM,PLOTMASK,REQMASK,IVAL,RVEC)
This routine gets called for each registered extract. The programmer must supply data associated with the status for the next set of requests from the clients. The post-processing viewer only needs PLOTMASK returned.

INDEX:i: I — The extract index. This number must be greater that 100 and defines an extract (and discipline).

EXNUM:i: I — The extract number associated with INDEX.

PLOTMASK:o: I — Programmer defined integer that specifies the plot attributes for the extract. This is passed to the render routine.

REQMASK:o: I — The request mask. Each bit specifies which subextracts are required to statisfy the plotting attributes. For example, 5 requests subextract 0 and subextract 2. If the most-significant bit is set all subextracts are requested, even if based on SUBOPT, the data exists (i.e. some state has changed).

IVAL:o: I — An integer sent to the clients associated with this extract.

RVEC:o: R(9) — A real vector of data sent to the clients with the request for this extract.

5.9 pV_GetExtract

PV_GETEXTRACT(EX,TYPE,NUM,IVEC,RVEC,NAME,NEXTEX)

Returns the internal **pV3** extract structure info. This routine can be called from any server suite programmer-supplied code. The extracts form a linked list. There is a unique list for each discipline. This routine allows the scanning of all active extracts by continual calls until the desired extract is found.

EX:i/o: I — Extract pointer (integer in FORTRAN). On input, this is the desired extract. The special case of the first extract is indicated by a 0 or NULL and is updated with the actual extract pointer for the current discipline. FORTRAN Note: this must be INTEGER*8 on 64 bit machines.

TYPE:o: I — The extract type or index. Zero indicates an extract pointer error.

NUM:o: I — The extract number. This is a unique number for the type.

IVEC:o: I(11) — Integer data set based on TYPE (length also determined by TYPE)

RVEC:o: R(12) — Real data set based on TYPE (length also determined by TYPE)

NAME:o: C*20 — Extract name.

NEXTEX:o: I — Extract pointer (integer in FORTRAN) to the next extract. 0 or NULL indicates that this is the last extract in the discipline. This can be used in the next call to **pV_GetExtract** (argument EX) to continue scanning the list. FORTRAN Note: this must be INTEGER*8 on 64 bit machines.

- Planar Cut - TYPE = 2

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

  IVEC(5) = Instancing mask

  RVEC(1-9) = Cut corners - Three of the 4 corners that denote the plane

  RVEC(10-12) = Plane normal

- Structured Block Plane - TYPE = 3

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

  IVEC(5) = Global Block number

IVEC(6) = Instancing mask

RVEC(1-6) = Block Indices - I_{min}, I_{max}, J_{min}, J_{max}, K_{min} and K_{max}.

RVEC(7) = Rotation of the data in the 2D Window

RVEC(8) = Center of 2D Window in X'

RVEC(9) = Center of 2D Window in Y'

RVEC(10) = 1/2 Window size

- Geometric Cut - TYPE = 4

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

  IVEC(5) = Cut index - May be the index negated

  IVEC(6) = Instancing mask

  RVEC(1) = Z prime

  RVEC(2) = Center of 2D Window in X'

  RVEC(3) = Center of 2D Window in Y'

  RVEC(4) = 1/2 Window size

  RVEC(5) = Rotation of the data in the 2D Window

  REVC(6) = DeltaTime - Zero indicates a normal extract. Any other value performs time averaging for the specified time segment (*Data Unsteady* only).

- Domain Surface - TYPE = 5

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

  IVEC(5) = Mapping flag

  IVEC(6) = Special surface scalar index

  IVEC(7) = Special surface vector index

  RVEC(1) = Center of 2D Window in X'

  RVEC(2) = Center of 2D Window in Y'

  RVEC(3) = 1/2 Window size

  RVEC(4) = Rotation of the data in the 2D Window

- Iso-Surface - TYPE = 7

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

  IVEC(5) = Scalar index for Iso-Surface

  RVEC(1) = Z prime

- StreamLine - TYPE = 18

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

  IVEC(5) = StreamLine Group number

  IVEC(6) = Client-id for client with seed location

  IVEC(7) = Cell index in client to start StreamLine

  IVEC(8) = Minimum StreamLine number for group

  IVEC(9) = Maximum StreamLine number for group

  IVEC(10) = Surface Index (0 - volume StreamLine)

  IVEC(11) = Number of StreamLine segments

  RVEC(1-3) = Seed location (XYZ)

- Particles - TYPE = 19

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

- Vector Cloud - TYPE = 20

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

  RVEC(1) = Threshold minimum

  RVEC(2) = Threshold maximum

- Points - TYPE = 21

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

  IVEC(5) = List Length - # of requested points

  IVEC(6) = Request Type - 0 XYZ, 1 IJK

- Programmer-defined - TYPE > 100

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

  IVEC(5) = IVAL

  RVEC(1-9) - Real values assoctated with the extract

5.10 pV_SetSub

ISTAT = PV_SETSUB(EX,SUBEX,NUMCS,LEN)
This sets the memory for the server-side subextract based on LEN. If there is a currently allocated block, then its size is adjusted to LEN. If LEN is zero any allocated block is free'd. This routine can be called from any server programmer-supplied code.

EX:i: I	Extract pointer (integer in FORTRAN) as returned by **pV_GetExtract** or **pV_Register**. FORTRAN Note: this must be INTEGER*8 on 64 bit machines.
SUBEX:i: I	Sub-extract number (0-11).
NUMCS:i: I	Client index (0 biased).
LEN:o: I	Length of structure, the total length of the memory block in words is LEN*SUBSIZE (when registered).
ISTAT:o: I	Return code: 0 - OK -1 - Invalid EX -2 - Invalid SUBEX -3 - Invalid NUMCS -4 - Allocation error

NOTES:
1) There are still subextracts for each client indexed by NUMCS – they need not be used.
2) After this call you may invoke **pV_GetSub** to expose the pointer to the block of memory. This block can now be filled.

5.11 pV_GetSub

ISTAT = PV_GETSUB(EX,SUBEX,NUMCS,PTR,LEN,CID)

Returns the internal **pV3** sub-extracts. This routine can be called from any server programmer-supplied code.

EX:i: I — Extract pointer (integer in FORTRAN) as returned by **pV_GetExtract**.
FORTRAN Note: this must be INTEGER*8 on 64 bit machines.

SUBEX:i: I — Sub-extract number (0-12 based on TYPE). 12 is a flag to indicate the State-Vector (only available within the Interactive Server if specifically requested – see **pV_SetExMask**).

NUMCS:i: I — Client index or StreamLine segment number (0 biased).

PTR:o: I — Integer (in FORTRAN) used as a pointer to the structure. In C this returns the pointer to the block of memory. A 0 (zero) indicates that the memory block is not allocated.

LEN:o: I — Length of structure. A 0 (zero) indicates that the structure is not currently filled.

CID:o: I — Client-id for the client that produced the segment (StreamLines Only).

ISTAT:o: I — Return code:

-1 - ERROR

0 - Not Updated since last access

1 - Data has been updated since last call to this routine

5.12 pV_ClearSub

PV_CLEARSUB(EX,SUBEX,NUMCS)

This clears the memory for the server-side subextract. It has the effect of causing the subsextract to be reloaded from the client(s). This routine should only be called from **pVSafe**.

EX:i: I — Extract pointer (integer in FORTRAN) as returned by **pV_GetExtract** or **pV_Register**.
FORTRAN Note: this must be INTEGER*8 on 64 bit machines.

SUBEX:i: I — Sub-extract number (0-11).

NUMCS:i: I — Client index (0 biased).

5.13 pV_SetExMask

PV_SETEXMASK(EX,MASK)

Sets the mask for the specified Extract. This routine should be called from **pVSafe** to insure that the change to the attributes is effective for the next iteration.

EX:i: I — Extract pointer (integer in FORTRAN) as returned by **pV_GetExtract**.
FORTRAN Note: this must be INTEGER*8 on 64 bit machines.

MASK:i: I — Plot mask to be set for the Extract.
Note: If the mask is set to -1, this informs **pV3** to get the State-Vector (for the Interactive Server only). -1 turns off the State-Vector flag once set.

5.14 pV_SetInsMask

PV_SETINSMASK(EX,MASK)

Sets the instancing flag for the specified Extract. This routine should be called from **pVSafe** to insure that the change to the attributes is effective for the next iteration. This is valid for programmed cuts, block planes and planar cuts only.

EX:i: I — Extract pointer (integer in FORTRAN) as returned by **pV_GetExtract**.
FORTRAN Note: this must be INTEGER*8 on 64 bit machines.

MASK:i: I — The Instancing flag to be set for the Extract; 0 – replicate (default for programmed cuts and block planes), 1 – do not instance (default for planar cuts).

5.15 pV_SetExColor

PV_SETEXCOLOR(EX,COLOR)

The default foreground color for an extract (if surface type) is grey. This routine sets the color.

EX:i: I — Extract pointer (integer in FORTRAN) as returned by **pV_GetExtract**.
FORTRAN Note: this must be INTEGER*8 on 64 bit machines.

COLOR:i: R(3) — 0.0 – 1.0 for red, green and blue. A -1 in the first entry of COLOR is a flag that indicates that the color should be set back to the default.

5.16 pVSend

PVSEND(IDISC)

This programmer-supplied routine is called by the IO thread of the server for each discipline. This invocation is after the beginning of frame messages have been sent to the clients and before the end of frame message. This is the call where any **pV3** server to client data may be sent. This should only be done through calls to **pV_SendData**.

IDISC:i: I	Discipline ID

5.17 pV_SendData

PV_SENDDATA(IDISC, ICID, IMID, NINT, INTS, NFLOAT, FLOATS, NCHAR, CHARS)

This routine should only be called from within **pVSEND**. It allows general messages to be sent from the server to the client(s). Each message is tagged by a message ID so the clients can determine what to do with the data. See **pVNotify** and **pV_GetServerData** in the client-side section for documentation on the routines required to recieve this data.

IDISC:i: I	Discipline ID
ICID:i: I	Client ID – If this is set to −1 then a broadcast of this data is sent to all clients in the discipline.
IMID:i: I	This is the message ID that may be freely used to indicate the type of message/data
NINT:i: I	The number of integer values in INTS to send in this message
INTS:i: I(*)	The integer values
NFLOAT:i: I	The number of real values in FLOATS to send
FLOATS:i: R(*)	The real values
NCHAR:i: I	The number of characters in CHRS to send in this message
CHARS:i: C*(*)	The characters

The following routine can be used to keep another GUI up-to-date when pV3 is rubber-banding or rendering:

5.18 pVRubber

PVRUBBER(IDWIN, ITYPE)

This programmer defined call-back is invoked during operations that would lock out an ancillary GUI. Also can inform the GUI about the status of the Network and Event Queue.

IDWIN:i: I

3 or 2 for the window where the operation is occuring. 0 indicates that **pV3** is rendering. 5 reports the state of the 'Traffic' lights.

ITYPE:i: I

Operation or State:

0 - set the first point (type unknown) or IDWIN = 0

1 - set first point - line

2 - set first point - circle

3 - set first point - box

4 - set second point - line

5 - set second point - circle

6 - set second point - box

0 - Red local, Red network light

1 - Yellow local, Red network light

2 - Green local, Red network light

3 - Red local, Yellow network light

4 - Yellow local, Yellow network light

5 - Green local, Yellow network light

6 - Red local, Green network light

7 - Yellow local, Green network light

8 - Green local, Green network light

The following routines are used for drawing programmer defined objects in the 3D window:

5.19 pVDraw3D

PVDRAW3D(ISTAT)

This programmer supplied subroutine is called by the **pV3** server or viewer at different times during the rendering phase. It is the responsibility of this routine to specify the object(s) to be plotted based on the rendering status. The object(s) are defined by calls to **pV_Object3D** and **pV_Annotate3D**. Calls to **pV_GetState**, **pV_GetExtract**, **pV_GetSub** and **pV_GetPointer** are valid but invocations of **pV_SetState** should be avoided.

ISTAT:i: I	Rendering stage: **0** - rendering for static object(s) **1** - rendering for dynamic object(s) **2** - rendering for translucent object(s)

The options listed above reflect the manner that the server suite draws the 3D scene. If the case is steady-state (or unsteady and *pause* is in effect) and the viewing transformation matrix is NOT changing, static objects only get rendered once (until the transformation matrix is changed again). *Snap-Shot Rendering* is used to improve drawing performance so the result of the rendering of the static objects is saved in a secondary *pixmap* and *ZBuffer*. These items are used to start the scene generation for subsequent animation frames.

Dynamic objects are those that will change or move from frame to frame even if the underlying (static) object(s) are not moving. An example of this is any dynamic surface (cut plane or iso-surface) or StreamLine animation using ‘blobs’ to display the integration pseudo-time.

Translucent objects, even if static, must be rendered last in order to get the scene to look correct. The ZBuffer must be updated with the translucent surfaces last.

If the application is unsteady (and not *paused*), the static objects as well as the dynamic and translucent objects are rendered each time step.

5.20 pV_Object3D

PV_OBJECT3D(ITYPE,ICOLOR,XYZP,RADII,COL,NP)

This routine must only be used from within **pVDraw3D** or the registered routine for programmer defined extracts. **pV_Object3D** defines additional plotting for the 3D window. Multiple objects may be drawn by multiple calls to this routine from within **pVDraw3D** or the extract rendering routine.

ITYPE:i: I

Type of plotting primitive:

- **0** - disjoint quadrilaterals - NP must be 4 times the number of quads
- **1** - disjoint triangles - NP must be 3 times the number of tris
- **2** - polytriangle strip - NP must be the length of the strip
- **3** - disjoint line segments - NP must be 2 times the number of segments
- **4** - polyline - NP must be the length of the line
- **5** - spheroids - NP must be the number of shpere-like objects to plot

ICOLOR:i: I

The method used for coloring the object:

- **-1** - draw the object white
- **0** - draw the object in grey
- **1** - color the object from the data in COL(NP)
- **2** - color the object from the color triads in COL(3,NP)
- **3** - color the object from the data in COL(NP) – No lighting
- **4** - color the object from the color triads in COL(3,NP) – No lighting

XYZP:i: R(3,NP)

The (x, y, z)-coordinates of the points that support the primitive.

RADII:i: R(NP)

The spheroid radius (in user coordinates). For ITYPE = 5 only. Points are plotted (and are MUCH faster) if RADII(1) is 0.0

COL:i: R(NP) or R(3,NP)

Color scaling for the data points (used for ICOLOR = 1). The values must be in the range 0.0 to 1.0 and the scalar field color map is applied to that range to compute the color for the point. COL contains the actual color values for ICOLOR = 2.

NP:i: I

Object length (the number of points). NOTE: if NP is negative XYZP is taken to be screen coordinates and not object coordinates.

5.21 pV_Annotate3D

PV_ANNOTATE3D(FID,COLOR,SIZE,WIDTH,XYZ,NORX,NORY,STR)
This routine must only be used from within **pVDraw3D** or the registered routine for programmer defined extracts. **pV_Annotate3D** draws annotation for the 3D window. Multiple text strings may be drawn by multiple calls to this routine from within **pVDraw3D** or the extract rendering routine.

FID:i: I	Font ID: **0** - Normal ASCII encoding **1** - Symbol font
COLOR:i: R(3)	0.0 – 1.0 for red, green and blue. Sets the text color.
SIZE:i: R	character size – object coordinates.
WIDTH:i: R	pixel width of the stroke-lines.
XYZ:i/o: R(3)	The (x, y, z)-coordinates of the start of the text. This is returned with the position for next character.
NORX:i: R(3)	Unit normal direction for the text.
NORY:i: R(3)	Unit normal direction for height of the text.
STR:i: C*(*)	FORTRAN Character string

NOTE: If NORY == 3*0.0 then it is assumed that the annotation will be based in screen coordinates and not in object space. SIZE is based on the window size (2.0 is the basis). XYZ must be normalized to that basis (-1.0 to 1.0). NORX(1) holds the angle like pV_Annotate2D.

The following routines are used for drawing programmer defined objects in the 2D window:

5.22 pVDraw2D

PVDRAW2D(ISTAT)

This programmer supplied subroutine is called by the server or viewer at different times during the 2D window rendering phase. It is the responsibility of this routine to specify the object(s) to be plotted based on the rendering status. The object(s) are defined by calls to **pV_Object2D** and **pV_Annotate2D**. Calls to **pV_GetState**, **pV_GetExtract**, **pV_GetSub** and **pV_GetPointer** are valid but invocations of **pV_SetState** should be avoided.

ISTAT:i: I	Rendering stage: **0** - rendering for static object(s) **1** - rendering for dynamic object(s)

Like rendering for the 3D window, the 2D window scene generation has multiple phases. Because translucency is not supported (nor does it make much sense) in the 2D window and there is no ZBuffer, the translucent phase is not implemented.

The only 2D dynamic object that **pV3** currently uses is the mapped cross-hair cursor.

For a program controlled 2D window, the programmer must perform a forced update (**pV_SetState** OPT = -1) for a static draw or should render everything during the dynamic phase.

5.23 pV_Object2D

PV_OBJECT2D(ITYPE,ICOLOR,XYW,COL,NP)

This routine must only be used from within **pVDraw2D**. It defines additional plotting for the 2D window. Multiple objects may be drawn by multiple calls to this routine from within **pVDraw2D**.

ITYPE:i: I — Type of plotting primitive:

0 - disjoint quadrilaterals - NP must be 4 times the number of quads

1 - disjoint triangles - NP must be 3 times the number of tris

2 - polytriangle strip - NP must be the length of the strip

3 - disjoint line segments - NP must be 2 times the number of segments

4 - polyline - NP must be the length of the line

5 - points - NP must be the number of dots

ICOLOR:i: I — The method used for coloring the object:

-1 - draw the object white

0 - draw the object in grey

1 - color the object from the data in COL(NP)

2 - color the object from the color triads in COL(3,NP)

XYW:i: R(2,NP) — X and Y window values for the points that make up the drawing primitive. The plotting in the 2D window for X ranges from -1.0 (left side) to 1.0 (the right side of the window). The Y range is from -1.0 (bottom) to 1.0 (the top of the window).

COL:i: R(NP) or R(3,NP) — Color scaling for the data points (used for ICOLOR = 1). The values must be in the range 0.0 to 1.0 and the scalar field color map is applied to that range to compute the color for the point. COL contains the actual color values for ICOLOR = 2.

NP:i: I — Object length in points

5.24 pV_Annotate2D

PV_ANNOTATE2D(FID,COLOR,SIZE,WIDTH,XYW,ANGLE,STRING)

This routine must only be used from within **pVDraw2D** or the registered routine for programmer defined extracts. **pV_Annotate2D** draws annotation for the 2D window. Multiple text strings may be drawn by multiple calls to this routine from within **pVDraw2D** or the extract rendering routine.

FID:i: I	Font ID: 0 Normal ASCII encoding, 1 - Symbol font.
COLOR:i: R(3)	0.0 – 1.0 for red, green and blue. Sets the text color.
SIZE:i: R	character size – window coordinates.
WIDTH:i: R	pixel width of the stroke-lines.
XYW:i/o: R(2)	The (x', y')-coordinates of the start of the text. These coordinates are in the range -1 to 1 for the viewable window. XYW is returned with the position for next character.
ANGLE:i: R	degrees, positive is righthanded rotation.
STRING:i: C*(*)	FORTRAN Character string

FORTRAN example that shows the entire normal font:

```
subroutine pVDraw2D(istat)
integer istat
real    xy(2), CH, color(3), width
data color/1.0, 1.0, 1.0/, width/1.0/, CH/0.0625/
if(istat .eq. 0) return
xy(1) = -0.9
xy(2) =  0.8
CALL pV_Annotate2D(0, color, CH, width, xy, 0.0, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ')
xy(1) = -0.9
xy(2) =  0.7
CALL pV_Annotate2D(0, color, CH, width, xy, 0.0, 'abcdefghijklmnopqrstuvwxyz')
xy(1) = -0.9
xy(2) =  0.6
CALL pV_Annotate2D(0, color, CH, width, xy, 0.0, '0123456789,.;:`"!?@#$%&|()')
xy(1) = -0.9
xy(2) =  0.5
CALL pV_Annotate2D(0, color, CH, width, xy, 0.0, '[]{}<>_+-*=/^~')
return
end
```

The following routines are used for programming 1D window drawing:

5.25 pVProbe

PVPROBE(FLAG,XAXIS,YAXIS)
This programmer supplied subroutine is called by the server or viewer at the end of the rendering phase if the programmer-defined probe is activated. It is the responsibility of this routine to specify the line(s) to be plotted and return the axis annotation. The line or lines are defined by calls to **pV_Line**. Calls to **pV_GetState**, **pV_GetExtract**, **pV_GetSub** and **pV_GetPointer** are valid but invocations of **pV_SetState** should be avoided.

FLAG:i: L	logical flag
	.TRUE. first call after probe has been activated
	.FALSE. subsequent calls
XAXIS:o: C*32	X-Axis label for the plot
YAXIS:o: C*32	Y-Axis label for the plot

Note: Currently the only way to initiate the programmer-defined probe is by a call to **pV_SetState** with OPT = 13 from **pVEvents**.

5.26 pV_Line

PV_LINE(X,Y,NP)
This routine must only be used from within **pVProbe** or **pVDraw1D**. It defines a line to be plotted in the 1D window. Multiple lines may be drawn by multiple calls to this routine. Auto-scaling is performed on the first line for the first call to **pVProbe** or **pVDraw1D** after the probe has been started. Afterwards user or programmed events can control the scaling.

X:i: R(NP)	X values for the line
Y:i: R(NP)	Y values for the line
NP:i: I	Line length

5.27 pVDraw1D

PVDRAW1D(ISTAT)
This programmer supplied subroutine is called by the server or viewer at the end of the 1D window drawing. It is the responsibility of this routine to specify other lines to be plotted. The line(s) are defined by calls to **pV_Line** and **pV_LineColor**. Calls to **pV_GetState**, **pV_GetExtract**, **pV_GetSub** and **pV_GetPointer** are valid but invocations of **pV_SetState** should be avoided.

ISTAT:i: I	Probe type, see **pV_GetState** $OPT = 13$, IVEC(1).

5.28 pV_LineColor

PV_LINECOLOR(COLOR)

This routine sets the line color for drawing to be done by **pV_Line**. After the drawing is complete, the line color is reset to white. Therefore if one needs to set the color before plotting any lines except if white is to be uesd.

COLOR:i: R(3)	The color; 0.0 – 1.0 for red, green and blue.

5.29 pVInit

PVINIT(NPIX1,NSX1,NSY1,NPIX2,NSX2,NSY2,NPIX3,NSX3,NSY3)

This routine is called at server suite initialization by the graphics thread before the windows are open. It allows the customized selection of window sizes and placement as well as allowing any advanced programming that should only be performed at initialization (such as memory allocation). Note: this is called by the batch server even though no windows are opened.

NPIX1:i/o: I — The size of the 1D window. Enters with **pV3**s default size.

NSX1:i/o: I — The suggested X pixel location (in the root window) for the start position of the 1D window. Enters with **pV3**s default location.

NSY1:i/o: I — The suggested Y pixel location (in the root window) for the start position of the 1D window. Enters with **pV3**s default location.

NPIX2:i/o: I — The size of the 2D window. Enters with **pV3**s default size.

NSX2:i/o: I — The suggested X pixel location (in the root window) for the start position of the 2D window. Enters with **pV3**s default location.

NSY2:i/o: I — The suggested Y pixel location (in the root window) for the start position of the 2D window. Enters with **pV3**s default location.

NPIX3:i/o: I — The size of the 3D window. Enters with **pV3**s default size.

NSX3:i/o: I — The suggested X pixel location (in the root window) for the start position of the 3D window. Enters with **pV3**s default location.

NSY3:i/o: I — The suggested Y pixel location (in the root window) for the start position of the 3D window. Enters with **pV3**s default location.

The following routines are used for picking objects in the 3D window:

5.30 pV_Pick

PV_PICK(ALL)
This routine sets the picking mode.

ALL:i: I	The mode: **-1** - Turns picking off, if on. **0** - Turns picking on for all extracts (ignore the individual extract mask). **1** - Turns picking on and use the extract pick mask to determine what is pickable.

5.31 pV_Select

PV_SELECT(BIT)
This should be called from within the render routine specified in the call to **pV_Register**. This sets the object bit for masking to be used with the programmer supplied extracts. This call must be placed between calls to **pV_Object3D** so the system can differentiate the objects and return meaningful indices.

BIT:i: I	Must be between 0 and 30. It must be set for picking to be properly masked.

5.32 pV_GetPicMask

PV_GETPICMASK(EX,MASK,ALL)
This returns the state of picking for the extract and the system.

EX:i: I	Extract pointer (integer in FORTRAN) as returned by **pV_GetExtract**. If NULL (0 in FORTRAN) just the pV3 server pick state is returned. FORTRAN Note: this must be INTEGER*8 on 64 bit machines.
MASK:o: I	The current pick mask for the extract.
ALL:o: I	The current pV3 server pick mode.

5.33 pV_SetPicMask

PV_SETPICMASK(EX,MASK)

This sets the picking mask for the extract. For those extracts that can have multiple objects drawn, a bit on in the mask refers to what objects are pickable.

EX:i: I — Extract pointer (integer in FORTRAN) as returned by **pV_GetExtract**.
FORTRAN Note: this must be INTEGER*8 on 64 bit machines.

MASK:i: I — -1 indicates all are pickable, 0 means that this extract can be ignored during picking. If 1 (bit 0) then the first object is selectable. For surface extracts, a mask of 3 specifies the surface entities. Objects must also be rendered as well as having their mask set on to be picked.

5.34 pVPicked

PVPICKED(EX,XYZ,REP,BIT,ISUB,INDEX,STATE)

This call-back is invoked once per render loop (at the end) when picking is active or when something has been picked.

EX:i: I — Extract pointer as returned by **pV_GetExtract**, **pV_Register** or **pV_CrExtract**. If NULL (0 in FORTRAN) this is an indication that the cursor was not pointing at any pickable object in the 3D window.
FORTRAN Note: this must be INTEGER*8 on 64 bit machines.

XYZ:i: R(3) — The 3D coordinates at the highlighted/picked point.

REP:i: I — The replicant index for objects that can be instanced. If negative this is an indication that the XYZ is from the mirrored reflection.

BIT:i: I — The selected bit index as specified by the call to **pV_Select** or from table below for built-in extracts.

INDEX:i: I — The index into the subextract or the call to **pV_Object3D** (1 baised).

STATE:i: I — 0 just pointed to, not yet picked,
1 picked – internally picking is now turned off.

BIT table for Domain Surfaces, Planar, Programmed Cuts and IsoSurfaces:

BIT	Mask	Sub-Extract	Description
0	1	0	Disjoint triangles
1	2	1	Disjoint quads
2	4	3	Mesh lines
3	8	4	Surface Outline
4	16	6	Vectors

BIT table for StreamLines, Particles and Vector Clouds:

BIT	Mask	Sub-Extract	Description
0	1	2	Nodes

6 Client Side Programming

6.1 The Use of Pointers in FORTRAN for the Clients

Some of the data that gets returned from **pV_GetStruc** or **pV_GetServerData** is in the form of pointers to blocks of memory. Either of these modes may be used in dealing with this data:

- %val

  A mechanism exists on all major workstation's FORTRAN (but NOT on CRAYs) to allow the pointer to be passed to a SUBROUTINE or FUNCTION and then have the memory treated as a normally declared vector or array. This is done by the VAX extension '%val(pointer)' used in the CALL or function invocation. When the pointer is passed to the sub-program by 'value', it is equivalent to passing a variable by 'reference' (the FORTRAN method). That is, in both cases, the address of the memory of interest is placed in the stack!

  In this case, the pointers are treated as INTEGER variables. This causes a problem in portability. The pointers must be treated as INTEGER*4 variables on all 32 bit machines. On 64 bit machines (ALPHAs and SGIs with R8000, R10000 or R12000 chips, in native mode) the pointers are INTEGER*8.

- POINTER statment

  The POINTER statement supported by most FORTRANs removes the portability problem. This syntax allows the coupling of a pointer with an array.

Using either mechanism, sophisticated **pV3** enhancements may be performed using FORTRAN. To complete this picture, three additional entries exist in to the client libraries so that FORTRAN programmer can allocate and free up blocks of memory.

6.1.1 MALLOCPV

PTR = MALLOCPV(NBYTES)

This function is equivalent to the C routine 'malloc'. It allocates a block of memory and returns the pointer to the block. On 64 bit machines, this function may need to be declared INTEGER*8.

NBYTES:i: I	The number of bytes to allocate.
PTR:o: I	The address of the block (0 is an error indicator).

6.1.2 FREEPV

FREEPV(PTR)

This function is equivalent to the C routine 'free'. It deallocates a block of memory. NOTE: Only free up blocks of memory that YOU allocate!

PTR:i: I	The address of the block.

6.1.3 REALLOCPV

PTRN = REALLOCPV(PTRO, NBYTES)

This function is equivalent to the C routine ‘realloc’. It re-allocates a block of memory and returns the pointer to the new block. On 64 bit machines, this function may need to be declared INTEGER*8.

PTRO:i: I	The address of the block.
NBYTES:i: I	The number of bytes to allocate.
PTRN:o: I	The address of the new block (0 is an error indicator).

6.2 Client Structures

In general, this data is the information returned from the programmer-supplied routines, though the connectivity tables are internally generated (with the proper IOPT). The structures are exposed so that multiple copies of the data need not be kept.

6.2.1 Node based

The following data is based on the 3D node space:

XYZ: R(3,NNODE) (x, y, z)-coordinates of grid nodes.

S: R(NNODE) Scalar function values.

T: R(NNODE) Threshold function values.

V: R(3,NNODE) Vector function values (Vx, Vy, Vz).

Z: R(NNODE) Current z' values for planar or programmed cuts.

IBLANK: I(*) IBLANK values for structured blocks. The length is the number of nodes in the structured blocks. For a case that is only structured blocks the length is NNODE. If the pointer to this array is 0, then no IBLANKing is active.

TBCON: I(*) Client-id number for block connectivity outside current domain.

6.2.2 Cell based

The following data is based on the 3D cell space:

CEL1: I(4,KCEL1) Node pointers for tetrahedral cells.

CEL2: I(5,KCEL2) Node pointers for pyramid cells.

CEL3: I(6,KCEL3) Node pointers for prism cells.

CEL4: I(8,KCEL4) Node pointers for hexahedral cells.

NPTET: I(8,KNPTET) Poly-Tetrahedra strip header:

NPTET(1,n) = the pointer to the end of the strip n, i.e. it points to the last entry in PTET for the poly-tetrahedral strip

NPTET(2,n) = the first node in the poly-tetrahedra

NPTET(3,n) = the second node in the poly-tetrahedra

NPTET(4,n) = the third node in the poly-tetrahedra

NPTET(5,n) = the fourth node in the poly-tetrahedra

NPTET(6,n) = the last node in the poly-tetrahedra

NPTET(7,n) = first connection (face 1 of the first tetra)

NPTET(8,n) = last connection (face 2 of the last tetra)

PTET: I(KPTET)

The rest of each poly-tetrahedra, 1 node per cell. The first and last node numbers in PTET are replaced by a flag to indicate the start and end of the strip. The flag is the negative of the strip number so that if you are in a strip, you can find which strip it is.

BLOCKS: I(6,KNBLCK)

Structured block definitions:

BLOCKS(1,m) $= Ni$

BLOCKS(2,m) $= Nj$

BLOCKS(3,m) $= Nk$

BLOCKS(4,m) = cell number that terminates the block

BLOCKS(5,m) = node number that terminates the block

BLOCKS(6,m) = index into CBLOCK that starts the connectivity table for the block.

6.2.3 Domain Surfaces

The following data is based on the surface space:

NSURF: I(3,KNSURF)

NSURF(1,n) is the pointer to the end of domain surface group n, i.e. it points to the last entry in both SCON and SCEL for that group.
NSURF(2,n) is the startup drawing state.
NSURF(3,n) is the global surface number.

SCON: I(KSURF)

connecting cell number for internal boundaries.

SCEL: I(4,KSURF)

node numbers for surface faces. For quadrilateral faces SCEL must be ordered clockwise or counter-clockwise; for triangular faces, SCEL(4,n) must be set to zero.

TSURF: C*20(KNSURF)

titles for domain surfaces.

6.2.4 Domain Surface Edges

The following data is based on the surface edge space:

NSED: I(2,KNSED)

NSED(1,n) is the pointer to the end of the surface edge n, i.e. it points to the last entry in SED for that group.
NSED(2,n) is the surface group number associated with this edge. NOTE: a domain surface may have as few a zero edges (the surface closes totally upon itself) and can have more than one edge!

SED: I(*)

pointers to nodes that make the surface edge. Each node connects to the last (except for the first entry) in a polyline structure.

6.2.5 Connectivity

The connectivity tables are used by the particle path algorithms and other tools that require neighborhood data. Any connection (cell number) that is negative indicates that there is no neighboring cell and the value is the index into SURF, SCON and SCEL for the surface face. If the pointers to all of these structures are 0, then this information has been deallocated.

CCEL1: I(4,KCEL1) — cell indices for the neighbors touching each of the four faces of the tetrahedron.

CCEL2: I(5,KCEL2) — cell indices for the neighbors touching each of the five faces of the pyramid.

CCEL3: I(5,KCEL3) — cell indices for the neighbors touching each of the five faces of the prism.

CCEL4: I(6,KCEL4) — cell indices for the neighbors touching each of the six faces of the hexahedon.

CPTET: I(2,KPTET) — The neighbor for face 3 and face 4 for each tetrahedra in the strip. Face 1 and face 2 are implicit (the last and next cells) except for the beginning and end of the strip where the connectivity can be found in the header NPTET.

CBLOCK: I(*) — A vector that contains the cell indices for the husk of the block. The start index for a particular block is in BLOCKS(6,m).

face 1 - $k = 1$
first $(Ni - 1) * (Nj - 1)$ entries indexed in a nested set of loops with j as the outer and i as the inner.

face 2 - $i = Ni - 1$
next $(Nj - 1) * (Nk - 1)$ entries indexed in a nested set of loops with k as the outer and j as the inner.

face 3 - $j = Nj - 1$
next $(Ni - 1) * (Nk - 1)$ entries indexed in a nested set of loops with k as the outer and i as the inner.

face 4 - $i = 1$
next $(Nj - 1) * (Nk - 1)$ entries indexed in a nested set of loops with k as the outer and j as the inner.

face 5 - $k = Nk - 1$
next $(Ni - 1) * (Nj - 1)$ entries indexed in a nested set of loops with j as the outer and i as the inner.

face 6 - $j = 1$
next $(Ni - 1) * (Nk - 1)$ entries indexed in a nested set of loops with k as the outer and i as the inner.

SURF: I(KSURF) — cell number that contains the face. Zero indicates a face without a cell.

CCEL: I(4,KSURF) — surface indices for the neighbors touching each of the three or four faces of a surface face.

6.3 pV_GetStruc

PV_GETSTRUC(OPT,PTR,LEN)

Returns the internal **pV3** client side structure. This **pV3** routine can be called from any programmer-supplied code.

OPT:i: I	Option set to specify what data to get.
PTR:o: I	Integer (in FORTRAN) used as a pointer to the structure. In C this returns the pointer to the block of memory. A 0 (zero) indicates that the memory block is not allocated. FORTRAN Note: this is an INTEGER*8 value on machines with 64 bit pointers.
LEN:o: I	Length of structure.

- Node Structures:

OPT	PTR - Pointer to	length of structure	Field index
301	XYZ	NNODE	
302	S	NNODE	KSCL
303	T	NNODE	KTHR
304	V	NNODE	KVCT
305	Z	NNODE	LASTZ
306	IBLANK	number of nodes in blocks	
307	TBCON	number of nodes in blocks	

- Cell Structures:

OPT	PTR - Pointer to	length of structure
311	CEL1	KCEL1
312	CEL2	KCEL2
313	CEL3	KCEL3
314	CEL4	KCEL4
315	NPTET	KNPTET
316	PTET	KPTET
317	BLOCKS	KNBLCK
318		KPHEDRA

- Domain Surface Structures:

OPT	PTR - Pointer to	length of structure
321	NSURF	KNSURF
322	TSURF	KNSURF
323	SURF	KSURF
324	SCEL	KSURF
327	SCON	KSURF

- Domain Surface Edge Structures:

OPT	PTR - Pointer to	length of structure
325	NSED	KNSED
326	SED	Number of ployline edge points - NSED(1,KNSED)

- Connectivity Structures:

OPT	PTR - Pointer to	length of structure
331	CCEL1	KCEL1
332	CCEL2	KCEL2
333	CCEL3	KCEL3
334	CCEL4	KCEL4
336	CPTET	KPTET
337	CBLOCKS	len of husks
338	CCEL	KSURF

6.4 pVSetStruc

FLAG = PVSETSTRUC(OPT,LEN,PTR)

Allows the programmer to control certain internal **pV3** client side structures. This call-back is invoked during pV_Init and pV_Update (for structure unsteady cases). If the programmer is responsible for the structure, then **pV3** will not free the memory block.

OPT:i: I — Option set to specify what memory block to set.

LEN:i: I — Length of structure.

PTR:i/o: I — Integer (in FORTRAN) to be used as a pointer to the appropriate structure. May require reallocation when input value is not NULL (or 0).
FORTRAN Note: this is an INTEGER*8 value on machines with 64 bit pointers.

FLAG:o I — Status flag: 0 – Let **pV3** deal with the memory block, 1 – PTR should be used for this block of memory.

- Node Structures:

OPT	length of structure (words)	PTR - Pointer to
301	3∗LEN	XYZ

- Cell Structures:

OPT	length of structure (words)	PTR - Pointer to
311	4∗LEN	CEL1
312	5∗LEN	CEL2
313	6∗LEN	CEL3
314	8∗LEN	CEL4
315	8∗LEN	NPTET
316	8∗LEN	PTET
317	6∗LEN	BLOCKS

6.5 pV_Field

PV_FIELD(KSCL, KVCT, KTHR, LASTZ)

Returns the indices for the last filled internal **pV3** client side node structures. This **pV3** routine can be called from any programmer-supplied code.

KSCL:o: I	Scalar field index
KVCT:o: I	Vector field index
KTHR:o: I	Threshold field index. Negative indicates T is filled with the scalar field associated with -KTHR.
LASTZ:o: I	Flag indicating what is in Z: **(-)** - Iso-surface with scalar field index equal to -LASTZ **(0)** - Result of equation of the plane for a planar cut **(+)** - Programmed cut with index LASTZ

6.6 pVExtract

PVEXTRACT(INDEX,EXNUM,REQMASK,IVAL,RVEC)

It is the responsibility of this routine to calculate and send back to the server any requested subextracts. The sending of messages to the server must be done by calls to either **pV_SendXi** or **pV_SendXr** based on the type of the data (specified during server registration - SUBTYPE).

INDEX:i: I	The extract index. Always greater than 100.
EXNUM:i: I	The extract number associated with INDEX.
REQMASK:i: I	The request mask. Each bit specifies which subextracts are required to statisfy the plotting attributes. For example, 5 requests subextract 0 and subextract 2.
IVAL:i: I	An integer associated with this extract.
RVEC:i: R(9)	A real vector of data associated with this extract request.

6.7 pV_SendXi

PV_SENDXI(INDEX,EXNUM,SUBINDEX,SUBSIZE,LEN,IVEC)
Sends an integer subextract back to the server.

INDEX:i: I The extract index. Always greater than 100.

EXNUM:i: I The extract number associated with INDEX.

SUBINDEX:i: I The subextract index (0-11).

SUBSIZE:i: I The subextract size per length. For example, if the subextract is to define a set of disjoint tris that support the extract, the size would be 3. This must match the size specified in the server code during registration.

LEN:i: I Length of structure.

IVEC:i The array of data to be passed to the server. The actual number of integers transferred is LEN ∗ SUBSIZE.

6.8 pV_SendXr

PV_SENDXR(INDEX,EXNUM,SUBINDEX,SUBSIZE,LEN,RVEC)
Sends a real subextract back to the server.

INDEX:i: I The extract index. Always greater than 100.

EXNUM:i: I The extract number associated with INDEX.

SUBINDEX:i: I The subextract index (0-11).

SUBSIZE:i: I The subextract size per length. For example, if the subextract is for the 3D coordinates (X,Y,Z) that support the extract, the size would be 3. This must match the size specified in the server code during registration.

LEN:i: I Length of structure.

RVEC:i The array of data to be passed to the server. The actual number of floating point values transferred is LEN ∗ SUBSIZE.

6.9 pVSSInit

PVSSINIT()
This call-back is invoked at the end of **pV3** initialization for steady-state cases. This allows the programmer to know when this phase is complete. The routine in the library prints out this message; ‘pV3 Initialization Phase Complete!’.

No Arguments

6.10 pVLocal

PVLOCAL(NB,IRANGE,JRANGE,KRANGE)

This call-back is invoked by the structured block extractor for each request of a plane (and therefore only required for modified servers that ask for these extracts). The servers only know the local storage so this routine must translate the input global indices to local indices for the client. All arguments are both input and output.

NB:i/o: I — The block number. If the block is not in this client then this value should be set to 0 upon return.

IRANGE:i/o: I(2) — The I range (I_{min} and I_{max}) for a structured block plane. When an I plane is specified I_{min} is equal to I_{max}.

JRANGE:i/o: I(2) — The J range (J_{min} and J_{max}) for the structured block plane. When an J plane is specified both J_{min} and J_{max} are equal.

KRANGE:i/o: I(2) — The K range (K_{min} and K_{max}) for the structured block plane. When an K plane is specified K_{min} is equal to K_{max}.

NOTE:
Any minimum range value set to 0 is internally reset to 1. An I_{max} of 0 indicates Ni, J_{max} of 0 is reset to Nj and K_{max} of 0 specifies Nk.

6.11 pVUpdate

PVUPDATE(ISTAT)

This call-back is invoked for unsteady cases from within the call to pV_Update. pVUpdate is called when any structure data and/or new node coordinates have been adjusted and before any server requests are processed.

ISTAT:i: I — The current client status:

-104 - **pV3** has not been properly initialized.

-1 - No servers – the structures may not be valid.

0 - At least one server is active and there are no errors.

others - documented client side errors – See the **pV3** Programmers Guide.

6.12 pV_Int

PV_INT(KC, XYZ, NNODES, NODES, WEIGHTS, NSUFG)

This routine calculates and returns the cell index containing the target position and the weights applied to the nodes of the cell to interpolate to that position. This **pV3** routine can be called from any programmer-supplied code.

KC:i/o: I

On input either 0 indicating that a search is to be one to find the cell or a 3D cell index to start the search. It is always best to start from a known cell that is close to the target. On output:

KC = 0 - Error position not in the volume

KC > 0 - 3D cell index containing the position

KC < 0 - (-)index of surface facet index of exiting face

XYZ:i: R(3)

Target position

NNODES:o: I

Number of nodes that support cell/facet

NODES:o: I(8)

3D Node indices, filled to NNODES for KC > 0.
For KC < 0; 3D Node indices for the facet and:

NODES(5) = face index in the cell

NODES(6) = 3D cell index

NODES(7) = Client ID for internal surfaces

WEIGHTS:o: R(4,8)

only filled for KC > 0.

WEIGHTS(1,n) - weight applied to the value associated with the 3D cell index in NODES to interpolate to position XYZ.

WEIGHTS(2,n) - weight applied to the value associated with the 3D cell index in NODES to compute the X derivative at XYZ

WEIGHTS(3,n) - weight applied to the value associated with the 3D cell index in NODES to compute the Y derivative at XYZ

WEIGHTS(4,n) - weight applied to the value associated with the 3D cell index in NODES to compute the Z derivative at XYZ

NSUFG:o: I

The surface group conatining the facet, only filled for
KC < 0, zero otherwise.

6.13 pVBounce

PVBOUNCE(KC, X0, U0, DT, XBOUNCE)

This routine gets called when the streamline or particle integrator hits a domain surface face that contains no information about continuing the integration. pVBounce will also get called when a face is hit with all nodes having an IBLANK of 2. This call-back can be used to attempt to keep the path within the domain (with obvious reduction in numerical accuracy). If bouncing is desired, it is the programmers resposibility to reposition the path to a valid location. Calls to pV_Int can be used to determine which facet within the domain surface has been hit.

KC:i/o: I

On input the cell number that conatins the last valid position found in X0.

On output this is the cell index containing, or close to, the location returned in XBOUNCE. It is usually OK to leave KC unmodified if the target position is visible to X0. The flag 0 (zero) indicates no bounce – the integration should terminate.

X0:i: R(3)

The last valid position

U0:i: R(3)

The vector field value at X0

DT:i: R

The time-step to complete this integration phase

XBOUNCE:i/o: R(3)

On input this is the target position. This location may not be the requested final position (at DT) but may be some intermediate guess used by the integrator (a middle stage in the RK4 scheme or an updated point in the Newton-Raphson iteration for BD4). XBOUNCE could be used as the argument XYZ in a call to pV_Int to get the detailed information about the path intersection.

On output XBOUNCE should be set to the bounced position (if KC has not been set to 0).

6.14 pVNotify

PVNOTIFY(HANDLE, MESSID)

This routine gets invoked when a general server to client message is generated by a modified running **pV3** server application. The routine **pV_GetServerData** must be called in order to revieve the data associated with the message.

HANDLE:i: I	An INTEGER handle that identifies the messsge
MESSID:i: I	The message index specified at the server

6.15 pV_GetServerData

PV_GETSERVERDATA(HANDLE, MESSID, NINT, PINT, NFLOAT, PFLOAT, NCHAR, PCHAR)

This routine gets the pointers that contain the data associated with the HANDLE that was specified by the call to **pVNotify**. This call can be executed in any client-side code after (or from within) the **pVNotify** call. It is the responsibility of the programmer to free up the memory exposed via these pointers after the data has been used. Once this call is made the message attached to HANDLE is removed from the internal message stack.

HANDLE:i: I	The INTEGER handle that identifies the messsge
MESSID:o: I	This is the message ID set by the server
NINT:o: I	The number of integer values
PINT:o: pI	The pointer to the integer values
NFLOAT:o: I	The number of real values
PFLOAT:o: pR	The pointer to the real values
NCHAR:o: I	The number of characters
PCHAR:o: pC	The pointer to the characters

FORTRAN Note:
PINT, PFLOAT and PCHAR are INTEGER*8 values on machines with 64 bit pointers.

A Plotting Masks

The following are additive (or-able) so that the proper attibutes can be specified.

A.1 Cut Surfaces

This mask controls the **pV3** scalar tools (planar cuts, programmed cut surfaces, iso-surfaces, structured block planes and domain surfaces) attributes.

1 - Render - Surface rendering on

2 - Grid - Mesh display on

4 - Grey - Surface colored with grey

8 - Threshold - Surface is thresholded according to the threshold function and limits

16 - Contour - Contour lines are plotted on the surface

32 - Translucent - Plot surface using the translucent attribute

64 - Arrows - Arrow drawing on

128 - Tufts - Grid of tufts on (dynamic only)

256 - Mapping - A 2D mapping exists for this surface (domain only)

512 - Probing - 2D probing is active

1024 - Outline - Outline drawing is requested (with the mask equal to only this flag)

4096 - Feature Lines - Feature Lines are drawn for this surface. See **pV_[S/G]etState** OPT = 14, for Domain Surfaces only.

8192 - Whole Arrows - With Arrows ON this bit disables the plotting of the vector in the 3D window.

16384 - Normal Arrows - With Arrows ON this bit enables the plotting of the vector component normal to the surface in the 3D window.

32768 - Tangent Arrows - With Arrows ON this bit enables the plotting of the vector component tangent to the surface in the 3D window.

A.2 StreamLines

This mask controls the **pV3** streamline plotting attributes and therefore the requested sub-extracts.

1 - Render - StreamLine rendering on

2 - Tube - Tube rendering on

4 - Grey - StreamLine drawn with default color

8 - Threshold - StreamLine is thresholded according to the threshold function and limits (not currently implemented)

16 - Back - StreamLine is backward going (can not be active with 32)

32 - Fore - StreamLine goes down stream (can not be active with 16)

64 - Ribbon - Ribbon rendering on (with 2 makes tubes with twist)

512 - Particles - Seeding on

2048 - Probing - StreamLine probe currently active for this StreamLine (Read-only).

A.3 Particles

This mask controls the **pV3** bubble rendering attributes and therefore the requested sub-extracts.

1 - Render - Bubble rendering on

2 - Size - Bubble size based on divergence like tubes - currently not used

4 - Grey - Bubble colored with default color

16 - Time - Bubbles are colored with the time of spawning

32 - Time Lines - Plot lines between particles in the same group

A.4 Vector Clouds

1 - Render - Vector cloud rendering on

4 - Grey - Vector cloud colored with default color