Server Builder's Guide

for

pV3 Rev. 2.05

Bob Haimes

December 12, 2001

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

This manual is a guide for those individuals wishing to use **pV3**'s client-side API and network based data movement, but do not want to view the data via **pV3**'s interactive server. This may be necessary when either the data presentation is not appropriate or some other workstation-enhanced technique (such as a Cave or VR) is used.

1.1 Unsteady Classification

Transient applications are classified in the following manner:

- OPT = 0; *Steady-State*
  This is the simplest case. Nothing changes in time.
- OPT = 1; *Data Unsteady*
  In this type of application the grid structure and position are fixed in time. The data defined at the nodes (both scalar and vector) changes with each time step. An example is when a boundary condition in the domain is changing.
- OPT = 2; *Grid Unsteady*
  These cases are 'Data Unsteady' plus the grid coordinates associated with each node are also allowed to move with each snapshot. An example of this is stator/rotor interaction in turbomachinery. The stator and rotor grids are separate, with the rotor grid sliding past the stator grid. In this case the stator portion is actually 'Data Unsteady' and the rotor grid moves radially.
- OPT = 3; *Structure Unsteady*
  If the number of nodes, number of cells or cell connectivity changes from iteration to iteration the case is 'Structure Unsteady'. An example of this mode is store separation.

1.2 Extracts

pV3 has been designed to minimize network traffic. The client-side library extracts lower dimensional data required by the requested visualization tool from the volume of data in-place. This distilled data is transferred to the graphics workstation. To further reduce the communication burden posed by the visualization session, the transient problem classification described above is used. Only the extracted data that has changed from the last iteration is sent over the network. An extract is therefore the results of a visualization tool (geometric cut, iso-surface, streamline and etc.) collected in a manner that provides flexibility and minimizes the volume of data so that a network based visualization system can provide good frame-rates.

There are 2 type of extracts allowed within the **pV3** system. The first are pre-defined and include those listed in Section 2. At startup the following built-in extracts will have been created:

- Domain Surface. One extract for each global surface with the plotting attributes set by the client side.
- Particle Data. For unsteady flows with vector fields.
- Vector Cloud. For clients with vector fields. The attributes set for this tool are in the off state at initialization.
- Dynamic. One extract is used by *pV3Server* for all dynamic tools.

The second type of extract is programmer defined. In this case code must be supplied at both the client and server sides.

Each extract is divided into as many as 12 sub-extract components. This is done for the following reasons:

- Unsteady updates
  Extract information should be segregated based upon reducing the amount of data. For example, a geometric-based cut is generated in a *Data Unsteady* client. For the second and subsequent iterations, all geometric data does not change, and therefore need not be retransmitted. The only new data required is the scalar field values associated with each vertex of the object. With only moving this data, the extract can be properly rendered, coloring the surface based on the current field. In this case, having the scalar values as a sub-extract produces a great benefit.
- Network usage
  Segregating data based on type is important. The underlying message passing (i.e. PVM) may deal with hetrogenous machine environments. Being able to do the bit or byte *twiddling* – based on type is required.

See Section 2 for an example of **pV3**'s pre-defined extracts and their sub-extract types.

For programmer defined extracts that use derived types and/or complex structures, the data must be decomposed at the client side so that it can pass through the network interface. On the server side the data can be reassembled from the sub-extracts.

1.3 Multi-Threading

pV3's servers (*pV3Server, pV3Batch* and *pV3Viewer*) are all multi-threaded. In fact, they have 2 threads. The application that gets built using this guide also uses 2 threads, like the interactive server. See Figure 1.

All extracts are double-buffered. This allows concurrent execution of the threads without data contention. The I/O thread collects extracts in the client's current iteration as the Graphics thread is rendering from the previous set of data. Once the thread handshaking is complete, the buffers are swapped, the last rendered data is thrown away (where appropriate) and the process continues.

The application built from this guide will have the same architecture as seen in Figure 1. The difference is that the **pV3** code used to 'Render scene', 'Get Xevents' and 'Update state' is replaced by a single programmer supplied routine; **pVRender**. See Section 5.1.

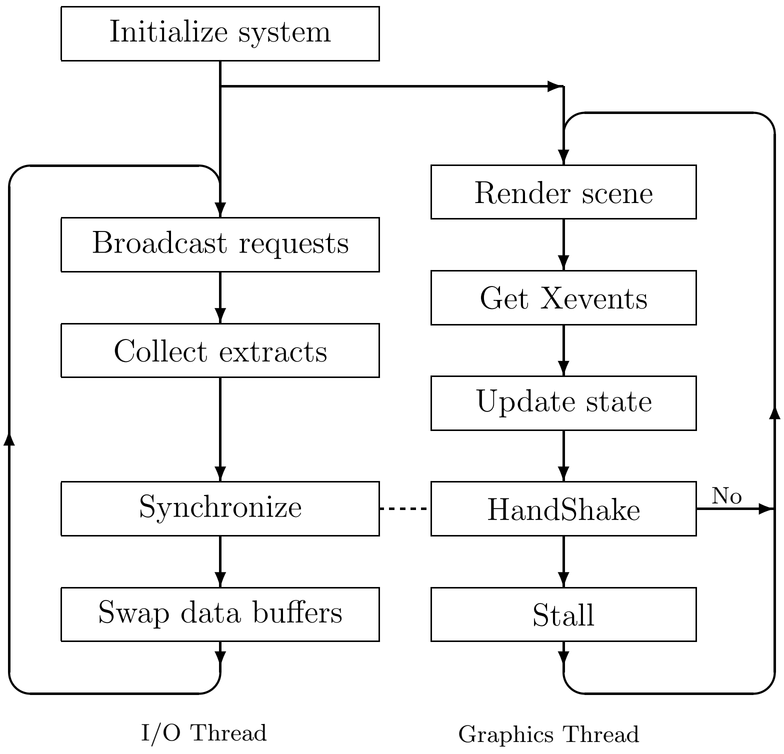

Figure 1: The **pV3** Server's Threading Control

1.4 Programming Notation

pV3 was designed to be accessible from both FORTRAN and C. FORTRAN is more restrictive in argument passing and naming, therefore it has shaped the programming interface. The routine descriptions in this guide are from the C programmer's point of view. But because FORTRAN is supported with the same API all routine arguments are pass by reference. It is assumed that a routine's argument is not modified unless documented as such.

For IBM and HP ports, all **pV3** entry points are the FORTRAN names in lower-case. On all other platforms except the CRAY, external entries are lower-case with an underscore ('_') appended to the end. CRAY entry points are upper-case with no appended underscores. See the file 'pV3ser.h' or 'wsdepend.h' in the servers subdirectory of the distribution for a method to avoid these problems.

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'.

2 Built-in Server Side Extracts

The following section describes the internal data stored in the **pV3** server structures for the built-in extracts. This data can be 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. 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.

2.1 Surfaces

This data is generated by the **pV3** scalar tools (planar cuts, 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
3	Block Planes	1 2 3 4 5 6 7 8
4	Geometric Cut	0 1 2 3 4 5 6 7 8
5	Domain Surface	0 1 2 3 4 5 6 7 8
7	Iso-Surface	0 1 2 3 4 5 6 7

0 - Surface Sub-Extract *Tris*

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

int TRIS[KTRI][4]	disjoint triangle definitions.
	TRIS[][0] = first node index for the triangle.
	TRIS[][1] = second node index for the triangle.
	TRIS[][2] = third node index for the triangle.
	TRIS[][3] = the parent 3D cell number (in the client).

1 - Surface Sub-Extract *Quads*

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

int QUADS[KQUAD][5]	disjoint quadrilateral definitions.
	QUADS[][0] = first node index for the quadrilateral.
	QUADS[][1] = second node index for the quadrilateral.
	QUADS[][2] = third node index for the quadrilateral.
	QUADS[][3] = fourth node index for the quadrilateral.
	QUADS[][4] = the parent 3D cell number (in the client).

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.

float XYZ[KXYZ][3]	(x, y, z)-coordinates for the nodes.

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.

int FACE[KFACE][2]	disjoint line definitions.
	FACE[][0] = first node index for the line.
	FACE[][1] = second node index for the line.

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.

int EDGE[KEDGE][3]	disjoint line definitions.
	EDGE[][0] = first node index for the line.
	EDGE[][1] = second node index for the line.
	EDGE[][2] = the parent surface face number (in the client).

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.

float S[KS]	scalar functional values for the nodes.

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.

float V[KV][3]	vector values (Vx, Vy, Vz) for the nodes.

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.

float T[KT]	threshold functional values for the nodes.

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.

float XY[KXY][2]	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.

2.2 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.

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.

int CELL[KCELL]	the parent 3D cell number (in the client).

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.

float TIME[KTIME]	integration time (from the seed position).

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.

float XYZ[KXYZ][3]	(x, y, z)-coordinates for the points.

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.

float DIV[KDIV]	used for streamtube rendering, where the size of the tube is based on a starting size mutiplied by **e** to this power.

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.

float ANG[KANG]	angle of the twist for ribbons in degrees.

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.

float S[KS]	scalar functional values for the points.

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.

float V[KV][3]	vector values (Vx, Vy, Vz) for the points.

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.

float T[KT]	threshold functional values for the points.

2.3 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.

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.

int NUM[KNUM]	the global particle number.

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.

float TIME[KTIME]	bubble simulation time when the particle was seeded.

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.

float XYZ[KXYZ][3]	(x, y, z)-coordinates for the bubbles.

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.

float DIV[KDIV]	optionally used for bubble rendering, where the size of the particle is based on a starting size mutiplied by **e** to this power.

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.

float S[KS]	scalar functional values for the bubbles.

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.

float V[KV][3]	vector values (Vx, Vy, Vz) for the bubbles.

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).

float T[KT]	threshold functional values for the bubbles.

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).

int GI[KGI]	group index for the bubbles (used for *time lines*).

2.4 Vector Clouds

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.

float XYZ[KXYZ][3]	(x, y, z)-coordinates for the vector cloud.

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.

float S[KS]	scalar functional values for the 3D nodes.

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.

float V[KV][3]	vector values (Vx, Vy, Vz) for the vector cloud.

2.5 Points

This data is updated by the **pV3** clients after a Point extract is created. This can only be done via a call to **pV_CrExtract**.

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.

int INDX[KINDX]	the point index.

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.

int ICEL[KICEL]	the cell index.

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.

float XYZ[KXYZ][3]	(x, y, z)-coordinates for the points.

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.

int LBIN[KLBIN]	the structured block index.

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.

float IJK[KIJK][3]	(I, J, K)-indices for the points.

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.

float S[KS]	scalar functional values for the requested points.

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.

float V[KV][3]	vector values (Vx, Vy, Vz) for the points.

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.

float T[KT]	threshold functional values for the points.

3 Calls that can be Invoked by Either Thread

3.1 pV_DiscipStat

PV_DISCIPSTAT(DID,NID,NCL,DNAME)
This routine returns the status for the disciplines known to the system.

int *DID	The discipline ID. The first discipline id always 0, therefore it is always safe to make this call with this argument zero.
int *NID	The total number of disciplines in the simulation. Returned.
int *NCL	The number of clients in the discipline DID. Returned.
char DNAME[20]	The discipline's name. Returned.

NID must be atleast 1 for any valid **pV3** application therefore it is always valid to make this call with DID = 0.

3.2 pV_ClientStat

PV_CLIENTSTAT(DID,CID,OPT,NCL,ON,CNAME)
This routine returns the status for a client within a discipline.

int *DID	The discipline ID. Must be a value from 0 to NID-1.
int *CID	The client ID. Must be a value from 1 to NCL.
int *OPT	The client's unsteady index (0-3). Returned.
int *NCL	The number of clients in the discipline DID. Returned.
int *ON	Visibility flag – not used. Returned.
char CNAME[20]	The clients's name. Returned.

NCL must be atleast 1 for any valid **pV3** discipline therefore it is always valid to make this call with CID = 1.

3.3 pV_CutStat

PV_CUTSTAT(DID,CIN,NCT,CTITLE)
This routine returns the status for a programmed cut within a discipline.

int *DID	The discipline ID. Must be a value from 0 to NID-1.
int *CIN	The cut index. Must be a value from 1 to NCT.
int *NCT	The number of geometric cuts in the discipline DID. Returned.
char CTITLE[32]	The geometric cut's title. Returned.

There may be no cuts (i.e. NCT = 0). To determine the number of cuts, call this routine with CIN = 1, then check NCT.

3.4 pV_FieldStat

PV_FIELDSTAT(DID,FID,NFL,FTY,FLIMS,FNAME)
This routine returns the status for a field variable within a discipline.

int *DID	The discipline ID. Must be a value from 0 to NID-1.
int *FID	The field ID. Must be a value from 1 to NFL.
int *NFL	The number of field variables in the discipline DID. Returned. An error is indicated by the value -1 which indicates that the discipline index or field ID is out of range.
int *FTY	The field's type (1-5). Returned. **1** Scalar **2** Vector **3** Surface scalar **4** Surface vector **5** Threshold
float FLIMS[2]	Field limits from the clients. Returned.
char FNAME[32]	The field's name. Returned.

NFL must be atleast 1 for any valid **pV3** application therefore it is always valid to make this call with FID = 1.

3.5 pV_SetDiscip

PV_SETDISCIP(DID)
This routine sets the current discipline. Not needed for a single discipline case.

int *DID	The discipline ID. Must be a value from 0 to NID-1.

3.6 pV_GetExtract

PV_GETEXTRACT(EX,TYPE,EXNUM,IVEC,RVEC,NAME,NEXTEX)
Returns the internal **pV3** extract structure info for the current discipline. 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.

void **EX	Extract pointer. On input, this is the desired extract. The special case of the first extract is indicated by a NULL and is updated with the actual extract pointer.
int *TYPE	The extract type. Returned.
int *EXNUM	The extract number. Returned.
int IVEC[]	Integer data set based on TYPE (length also determined by TYPE). Returned.
int RVEC[]	Float data set based on TYPE (length also determined by TYPE). Returned.
char NAME[20]	Extract name. Returned.
void **NEXTEX	Returned pointer to the next extract. NULL indicates that this is the last extract. NEXTEX can be used in the next call to **pV_GetExtract** (argument EX) to continue scanning the list.

The following data is related to data in the Graphics buffer:

- Planar Cut - TYPE = 2
  IVEC[0] = Plot Mask

  IVEC[1] = Scalar field index

  IVEC[2] = Vector field index

  IVEC[3] = Threshold index

  IVEC[4] = Instancing mask

  RVEC[0-8] = Cut corners - Three of the 4 corners that denote the plane

  RVEC[9-11] = Plane normal

- Structured Block Plane - TYPE = 3
  IVEC[0] = Plot Mask

  IVEC[1] = Scalar field index

  IVEC[2] = Vector field index

  IVEC[3] = Threshold index

  IVEC[4] = Global Block number

  IVEC[5] = Instancing mask

  RVEC[0-5] = Block Indices - I_{min}, I_{max}, J_{min}, J_{max}, K_{min} and K_{max}.

  RVEC[6] = Rotation of the data in the 2D Window

RVEC[7] = Center of 2D Window in X'

RVEC[8] = Center of 2D Window in Y'

RVEC[9] = 1/2 Window size

- Geometric Cut - TYPE = 4

  IVEC[0] = Plot Mask

  IVEC[1] = Scalar field index

  IVEC[2] = Vector field index

  IVEC[3] = Threshold index

  IVEC[4] = Cut index - May be the index negated

  IVEC[5] = Instancing mask

  RVEC[0] = Z prime

  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

  REVC[5] = 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[0] = Plot Mask

  IVEC[1] = Scalar field index

  IVEC[2] = Vector field index

  IVEC[3] = Threshold index

  IVEC[4] = Mapping flag

  IVEC[5] = Special surface scalar index

  IVEC[6] = Special surface vector index

  RVEC[0] = Center of 2D Window in X'

  RVEC[1] = Center of 2D Window in Y'

  RVEC[2] = 1/2 Window size

  RVEC[3] = Rotation of the data in the 2D Window

- Iso-Surface - TYPE = 7

  IVEC[0] = Plot Mask

  IVEC[1] = Scalar field index

  IVEC[2] = Vector field index

  IVEC[3] = Threshold index

  IVEC[4] = Scalar index for Iso-Surface

  RVEC[0] = Z prime

- StreamLine - TYPE = 18

  IVEC[0] = Plot Mask

  IVEC[1] = Scalar field index

  IVEC[2] = Vector field index

  IVEC[3] = Threshold index

  IVEC[4] = StreamLine Group number

  IVEC[5] = Client-id for client with seed location

  IVEC[6] = Cell index in client to start StreamLine

  IVEC[7] = Minimum StreamLine number for group

  IVEC[8] = Maximum StreamLine number for group

  IVEC[9] = Surface Index (0 - volume StreamLine)

  IVEC[10] = Number of StreamLine segments

  RVEC[0-2] = Seed location (XYZ)

- Particles - TYPE = 19

  IVEC[0] = Plot Mask

  IVEC[1] = Scalar field index

  IVEC[2] = Vector field index

  IVEC[3] = Threshold index

- Vector Cloud - TYPE = 20

  IVEC[0] = Plot Mask

  IVEC[1] = Scalar field index

  IVEC[2] = Vector field index

  IVEC[3] = Threshold index

  RVEC[0] = Threshold minimum

  RVEC[1] = Threshold maximum

- Points - TYPE = 21

  IVEC[0] = Plot Mask

  IVEC[1] = Scalar field index

  IVEC[2] = Vector field index

  IVEC[3] = Threshold index

  IVEC[4] = List Length - # of requested points

  IVEC[5] = Request Type - 0 XYZ, 1 IJK

- Programmer-defined - TYPE > 100

  IVEC[0] = Plot Mask - Not used

  IVEC[1] = Scalar field index

  IVEC[2] = Vector field index

  IVEC[3] = Threshold index

  IVEC[4] = IVAL

  RVEC[0-8] - Float values assoctated with the extract

3.7 pV_Pause

PV_PAUSE(STATE)
Handles the *Pause* and *Single-Step* functions as initiated by key strokes in the interactive server. NOTE: The process of pasuing is pipelined and gets set after another call to **pVSafe**.

int *STATE	Pause state (input and returned) On input: -2 - Release form Single-step -1 - Release from pause or step in Single-Step mode 0 - Do nothing – just get status 1 - Pause 2 - Set Single-Step Output (status): -4 - mode already invoked -3 - Steady State case -2 - More than 1 server -1 - Change of pause state already queued 0 - Not Paused or in Single-Step mode 1 - Paused 2 - In Single-Step mode

4 I/O Thread Routines and Calls

4.1 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

NOTE:

The first calls to **pVSafe** are done when there is only one thread. The first should be used to register all extracts (at least for the first allocation).

4.2 pV_Register

EX = 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 registered within the appropriate discipline. The client-side extraction code must be linked with the client application. See the Advanced Programmer's Guide.

void *EX	The extract pointer if EXNUM does not indicate an error.
int *INDEX	The extract index. This number must be greater that 100 and defines an extract. Different disciplines must use unique extract indices!
char NAME[20]	Extract name.
int SUBTYPE[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 float (1).
int SUBSIZE[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.
int SUBOPT[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

int SUBLOC[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.
void (*ROUTINE)()	Not used – for compatibility with the Advanced Programmer's interface.
int *EXNUM	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. 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

4.3 pVSetExtract

PVSETEXTRACT(INDEX,EXNUM,PLOTMASK,REQMASK,IVAL,RVEC)
This routine gets called for each registered extract during the request collection phase.

int *INDEX	The extract index. This number must be greater that 100 and defines an extract (and discipline).
int *EXNUM	The extract number associated with INDEX.
int *PLOTMASK	Not used, for compatibility with the Advanced Programmer's interface.
int *REQMASK	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). Must be filled on return.
int *IVAL	An integer sent to the clients associated with this extract. Must be set upon return.
float RVEC[9]	A float vector of data sent to the clients with the request for this extract. Must be set upon return.

4.4 pV_SetExState

PV_SETEXSTATE(EX,PLTMASK,SCALAR,VECTOR,THRES)
Sets the plot mask and field variables 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 overrides the default field variable specifications for this extract. The default values can be set via a call to **pV_SetState** with OPT = 5, 6, and/or 7.

void **EX	Extract pointer as returned by **pV_GetExtract**, **pV_Register** or **pV_CrExtract**. NOTE: these pointers do not change during the life of the server application.
int *PLTMASK	Plot mask to be set for the extract – built-ins only. See the Appendix for the mask values.
int *SCALAR	Scalar field index to be used. Zero indicates the default scalar.
int *VECTOR	Vector field index to be used. Zero indicates the default vector index.
int *THRESH	Threshold index, (-) indicates a scalar field index. Zero indicates the use of the default threshold field variable.

4.5 pV_CrExtract

EX = PV_CREXTRACT(TYPE,NAME,IVEC,RVEC,EXNUM)
Creates a built-in extract for the current discipline.

void *EX	The extract pointer if EXNUM does not indicate an error.
int *TYPE	The extract type. Valid entries are 2, 4, 7, 18 and 21. Type of 21 invokes the call-back **pVFillPoints**.
char NAME[20]	Extract name.
int IVEC[7]	Integer data set based on TYPE.
float RVEC[9]	Float data set based on TYPE.
int *EXNUM	The returned extract instance. If the number is negative it is an indication of an error: -1 - Length error -2 - Invalid TYPE -11 - Allocation error -12 - Routine not called from **pVSafe**

- Planar Cut - TYPE = 2

  IVEC[0] = Instancing Mask - 0 allow replication, 1 no instancing

  RVEC[0-8] = Cut corners - Three of the 4 corners that denote the plane

- Structured Block Plane - TYPE = 3

  IVEC[0] = Global Block number

  IVEC[1] = Instancing Mask - 0 allow replication, 1 no instancing

  RVEC[0-5] = Block Indices - I_{min}, I_{max}, J_{min}, J_{max}, K_{min} and K_{max}.

- Geometric Cut - TYPE = 4

  IVEC[0] = Cut index - 1 to NCT

  IVEC[1] = Instancing Mask - 0 allow replication, 1 no instancing

  RVEC[0] = Z prime

  REVC[5] = DeltaTime - Zero indicates a normal extract. Any other value performs time averaging for the specified time segment (*Data Unsteady* only).

- Iso-Surface - TYPE = 7

  IVEC[0] = Scalar index for Iso-Surface

  RVEC[0] = Z prime

- StreamLine - TYPE = 18

  IVEC[0] = StreamLine Group number

IVEC[1] = Client-id for client with seed location

IVEC[2] = Cell index in client to start StreamLine

IVEC[3] = Minimum StreamLine number for group

IVEC[4] = Maximum StreamLine number for group

IVEC[5] = Surface Index (0 - volume StreamLine)

IVEC[6] = StreamLine number

RVEC[0-2] = Seed location (XYZ)

- Point - TYPE = 21

  IVEC[0] = Length of Point Request list

  IVEC[1] = Request Type - 0 XYZ, 1 IJK

4.6 pVFillPoints

PVFILLPOINTS(EX,EXNUM,PTYPE,LEN,CC,POS,NAME,NLen)
This routine gets called when a Point extract is created by **pV_CrExtract**. It specifies the points in space to be located and returned with data.

void **EX	Extract pointer.
int *EXNUM	The extract number associated with EX.
int *PTYPE	The specified request type – 0 XYZ, 1 IJK
int *LEN	The specified number of points.
int CC[][2]	The client ID/cell index pair (filled): If the type is 1 then CC[][1] refers to the global block number and not the cell index. A client id = -1 tries all clients.
float POS[][3]	The specified position (filled): If PTYPE is 0 this is just the XYZ coordinates, otherwise it refers to the IJK in the global block specified in CC. The IJKs can have fractional components which cause interpolation in the appropriate cell.
char NAME[NLen]	Extract name.
int NLen	NAMEs length.

4.7 pV_DeExtract

PV_DEEXTRACT(EX)
Deletes the specified extract. This routine should only be called from **pVSafe**.

void **EX	Extract pointer as returned by **pV_GetExtract**, **pV_Register** or **pV_CrExtract**.

4.8 pV_DynTool

PV_DYNTOOL(DID, IFN, PTMSK, IVEC, RVEC, ISTAT)

Handles the Dynamic Surface functions as as in the interactive server. Should be be called from pVSafe to ensure the changes get made after pVSafe returns.

int *DID	The discipline ID
int *IFN	The function index: 0 - No tool 2 - Planar cut 3 - Structured block tool 4 - Geometric cut 5 - Domain surface mapping 6 - Domain surface mapping with special functions 7 - Iso-surface
int *PTMSK	The plot mask
int IVEC[]	Integer data set based on IFN
int RVEC[]	Float data set based on IFN
int *ISTAT	Return status

- Planar Cut - IFN = 2

  RVEC[0-8] = Cut corners - Three of the 4 corners that denote the plane

- Structured Block Plane - IFN = 3

  IVEC[0] = Global Block number

  IVEC[1-6] = Block Indices - I_{min}, I_{max}, J_{min}, J_{max}, K_{min} and K_{max}.

- Geometric Cut - IFN = 4

  IVEC[0] = Cut index

  RVEC[0] = Z prime

  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

- Domain Surface - IFN = 5 & 6

  IVEC[0] = Surface Index

  IVEC[1] = Special surface scalar index

  IVEC[2] = 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 - IFN = 7

  IVEC[0] = Scalar index for Iso-Surface

  RVEC[0] = Z prime

4.9 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**.

void **EX	Extract pointer as returned by **pV_GetExtract** or **pV_Register**.
int *SUBEX	Sub-extract number (0-11).
int *NUMCS	Client index (0 biased).

5 Graphics Thread Routines and Calls

5.1 pVRender

PVRENDER(TIME)

This is the routine that gets called to render the data. It might get called more than once for each set of data depending on the length of time to render and the client-side update frequency. In this case, the value of TIME does not change.

float *TIME	The simulation time for the data.

5.2 pV_GetSub

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

Returns the internal **pV3** sub-extracts. This routine returns the Graphics thread pointers (from the two buffers).

void **EX	Extract pointer as returned by **pV_GetExtract**, **pV_Register** or **pV_CrExtract**.
int *SUBEX	Sub-extract number (0-11 based on TYPE).
int *NUMCS	Client index or StreamLine segment number (0 biased).
void **PTR	Returned pointer to the structure. NULL indicates that the memory block is not allocated.
int *LEN	Length of structure. A 0 (zero) indicates that the structure is not currently filled. Returned.
int *CID	Client-id for the client that produced the segment Returned (StreamLines Only).
int ISTAT	Return code: -1 - ERROR 0 - Not Updated since last access 1 - Data has been updated since last call to this routine

5.3 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.

void **EX	Extract pointer as returned by **pV_GetExtract** or **pV_Register**.
int *SUBEX	Sub-extract number (0-11).
int *NUMCS	Client index (0 biased).
int *LEN	Length of structure, the total length of the memory block in words is LEN*SUBSIZE (when registered).
int ISTAT	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.

6 Running Your Server

The **PVM** daemon(s) and with co-processing, the solver, must be executing. Without a **pV3** server running, every time the solution is updated, a check is made for the number of members in the **PVM** group *pV3Server* (Note: this name can be changed for multiple jobs running under the same user ID – see the Section 6.1 for the environment variable 'pV3_Group'). If no servers are found, no action is taken. When a **pV3** server starts, it enrolls in the specified group. The next time the solution is updated, an initialization message is processed and the visualization session begins. Each subsequent time in the solver completes a time step, visualization state messages and *extract* requests are gathered, the appropriate data calculated, collected and sent to the active server(s).

When the user is finished with the visualization, the server sends a termination message and exits. The clients receive the message, and if no other servers are running, cleans up any memory allocations used for the visualization. Then the scheme reverts to looking for server initialization, if termination was not specified at **pV3** client initialization.

6.1 Environment Variables

A **pV3** server built using this guide automatically looks at three Unix environment variables:

'pV3_TO' should be used to change the internal Time-Out constant. If the variable is set, it must be an integer string which is the number of seconds to use for the Time-Out constant (the server's default is 60). This may be required if the time between solution updates is long. See the section in the **pV3** Server User's Reference Manual on *Time-Outs and Error Recovery.*

'pV3_Group' is usefull for differentiating multiple **PVM** jobs running under the same user ID. If this variable is set for the solver (client-side) before execution, it overrides the default client side group name *pV3Client.* The name used is the string assigned to this variable with *Client* appended. By setting this variable before server execution, it will set the server group to the variable's string with *Server* appended instead of using *pV3Server.* Only clients with the appropraite matching group name will be connected to this session.

'pV3_Threading' is used to specify how handshaking is handled between the active threads. There are two methods; (1) 'Hard' where the thread sits in a hard loop (with a thread yeild) looking for a change of state, or (2) 'Flag' where the threads use Semaphores for waiting until the state changes. The advantage of 'Hard' is interactivity, the advantage for 'Flag' is less processor time consumed. By default, this variable is set to 'Hard' for most situations with single processor workstations and 'Flag' for multi-processors.
Exceptions: ALPHA's default is 'Hard', SUN's default is 'Flag', reguardless of number of processors.

6.2 Special File - The Lock File

If the server is running on a multi-processor SGI workstation a file is used for the coordination of the 2 threads generated during execution. This file has the name '.pV3.locks' and is open in the current directory. It should be noted that running two invocations of the **pV3** server from the same directory will NOT work. Both will use the same file for the lock and semaphore arena!

7 An FEL Example

The following is a very simple coding example of both making a programmed defined extract (at the server and client-side) as well as skeleton code for the customized server. It is assumed that the visualization is steady-state and there is only one discipline.

7.1 Server code

7.1.1 I/O Thread code

```
#include <stdio.h>
#include <stdlib.h>
#include "pV3ser.h"

/* required to deal with visualization control and state */

extern void get_field(int *scalar, int *vector, int *thresh);
extern int  get_pltmask(int type, int exnum);

void
PVSAFE()
{
  static int EXNUM = -14;
  static char *name = "FEL StreamLine      ";
  static int subtype[12] = { 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
  static int subsize[12] = { 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
  static int subopt[12]  = { 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
  static int subloc[12]  = {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};

  int   ivec[11], type, exnum, pltmask, scalar, vector, thresh, opt;
  char  exname[20];
  float rvec[12];
  void  *ex, *nextex;

  /* register the extract */
  opt = 101;
  if (EXNUM != -14) PV_REGISTER(&opt, name, subtype, subsize,
                                subopt, subloc, NULL, &EXNUM);

  /* get the current field variables */
  get_field(&scalar, &vector, &thresh);

  /* loop through all defined extracts */
  ex = NULL;
  PV_GETEXTRACT(&ex, &type, &exnum, ivec, rvec, exname, &nextex);
  while (ex != NULL) {
    PV_GETEXTRACT(&ex, &type, &exnum, ivec, rvec, exname, &nextex);

    /* set extract's attributes for the next set of data */
    pltmask = 0;
    if (type < 100) pltmask = get_pltmask(type, exnum);
    PV_SETEXSTATE(&ex, &pltmask, &scalar, &vector, &thresh);

    ex = nextex;
  }
}

void
PVSETEXTRACT(int *index, int *exnum, int *pltmsk, int *mask,
             int *ival, float *rvec)
{
  /* called only for programmed extracts */
  if (*index != 101) return;

  /* get the one sub-extract */
  *mask = 1;

  /* set the start location */
  rvec[0] = 0.0;  /* X */
  rvec[1] = 0.0;  /* Y */
  rvec[2] = 0.0;  /* Z */
}
```

7.1.2 Graphics Thread code

```
#include <stdio.h>
#include <stdlib.h>
#include "pV3ser.h"

  typedef struct {
    float X; float Y; float Z;} Triad;

void
PVRENDER(float *time)
{
  int   ivec[11], type, exnum, len, cid;
  char  exname[20];
  float rvec[12];
  void  *ex, *nextex;
  Triad *streamline;

  /* loop through all defined extracts */
  ex = NULL;
  PV_GETEXTRACT(&ex, &type, &exnum, ivec, rvec, exname, &nextex);
  while (ex != NULL) {
    PV_GETEXTRACT(&ex, &type, &exnum, ivec, rvec, exname, &nextex);

    /* our extract */
    if (type == 101) {
      PV_GETSUB(&ex, 0, 0, (void **) &streamline, &len, &cid);
      /* plot the streamline */
    }

    /* domain surfaces */
    if (type == 5) {
      /* plot domain surfaces */
    }

    ex = nextex;
  }
}
```

7.2 Client Side

It is assumed that most of the **pV3** client-side code has already been constructed. See the **pV3** Programmer's Guide for a complete description of this coupling. The code listed below calculates a streamline using FEL as a programmed extract. It assumes that only one extract has been defined and that extract has only one sub-extract.

It may be necessary to consult the Advanced Programmer's Guide and FEL's documentation to understand this C++ code listing.

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>

#include <FEL.h>
#include "pV3.h"

#define MAX_LENGTH 2000
#define TIMESTEP 0.02

void
pVEXTRACT(int *index, int *exnum, int *reqmask, int *ival, float *rvec)
{
    static int first = 0;
    static FEL_grid *grid;
    static FEL_vector_field *velocity;
    float v[3], *pVel, *buffer, *iblank;
    int i, i3, j, opt, nnodes, nblank;
    int length;
    float streamline[MAX_LENGTH][3];
    vertex_data *pVgrid;
    FEL_bary_pos current_bary;
    FEL_bary_pos last_bary;
    FEL_phys_pos current_position;

    if (first == 0) {
      // instantiate grid object
      grid = new FEL_structured_grid("grid", 1);

      // set the grid type
      grid->set_grid_type(FEL_GRID_P3D_NO_IBLANK);
      grid->set_grid_type(FEL_GRID_P3D_SINGLE_ZONE);

      // load the grid data
      opt = 301;
      pV_GETSTRUC(&opt, (void **)&buffer, &nnodes);
      pVgrid = (vertex_data *) malloc(nnodes * sizeof(vertex_data));
      for (i=0,i3=0; i<nnodes; i++,i3+=3) {
        (pVgrid + i)->x      = buffer[i3  ];
        (pVgrid + i)->y      = buffer[i3+1];
        (pVgrid + i)->z      = buffer[i3+2];
        (pVgrid + i)->iblank = 1;
      }
      opt = 306;
      pV_GETSTRUC(&opt, (void **)&iblank, &nblank);
      if (nblank != 0)
        for (i=0; i<nnodes; i++) (pVgrid + i)->iblank=iblank[i];
      grid->geom->new_timestep(pVgrid);

      // instantiate a vector field
      velocity = FEL_make_new_vector_field("velocity", grid, 2);
      // select the file type
      velocity->set_file_type(FEL_PLOT3D_SOLUTION_FILE);
      // load the vector data
      opt = 304;
      pV_GETSTRUC(&opt, (void **)&pVel, &nnodes);
      velocity->new_timestep(pVel);
    }
    first++;

    // initialize the current physical position
    length = 0;
    current_position.x = rvec[0];
    current_position.y = rvec[1];
    current_position.z = rvec[2];
    current_position.time = 0.0;

    // initialize the streamline vertices
    streamline[length][0] = current_position.x;
    streamline[length][1] = current_position.y;
    streamline[length][2] = current_position.z;

    // initialize the barycentric coordinates
    grid->phys_to_bary(current_position, current_bary);
    last_bary = current_bary;

    // Euler streamline computation loop
    while (length < MAX_LENGTH-1)
    {
        // get the velocity value using last_bary to optimize
        // point location algorithm.  Stop if get_value
        // returns 0 as that means we fell off the grid
        if (!velocity->get_value(current_position, last_bary, v)) break;

        // add the velocity times the timestep to the current position
        current_position.x += TIMESTEP * v[0];
```

```
        current_position.y += TIMESTEP * v[1];
        current_position.z += TIMESTEP * v[2];
        length++;

        // save the current position as a streamline vertex
        streamline[length][0] = current_position.x;
        streamline[length][1] = current_position.y;
        streamline[length][2] = current_position.z;
    }

    // send the streamline data to the server
    i = 3; j = 0;
    length++;
    pV_SENDXR(index, exnum, &j, &i, &length, (float *)streamline);
}
```

A Plotting Masks for Built-in Extracts

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, block planes, iso-surfaces 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.

8092 - 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