

Parallel Compositing Specification

Version 1.0
Sep 13, 2006

Table of Contents

Revision History	4
1 Overview.....	4
2 Definitions / Nomenclature	5
2.1 Application.....	5
2.2 System	5
2.3 Session	5
2.4 Host	5
2.5 Context.....	5
2.6 Frame.....	5
2.7 Framelet.....	6
2.8 Output	6
2.9 Result.....	6
2.10 Relationships Between PC Terms.....	7
2.11 Order of Function Calls.....	7
3 PC Functions	8
3.1 PC Function Parameter Types.....	8
3.1.1 Parameter Types.....	8
3.1.2 PCchannel Type.....	8
3.2 System & Session Functions	9
3.2.1 pcSystemInitialize – Initialize PC for a Program	9
3.2.2 pcSystemFinalize – Uninitialize PC for a Program	10
3.2.3 pcSessionCreate – Create a PC session	11
3.2.4 pcSessionDestroy – Destroy a PC Session	11
3.2.5 PC System Properties	11
3.2.6 pcSystemGetInteger / pcSystemGetString – retrieve a system property.....	13
3.3 Context Functions.....	13
3.3.1 PC Context Properties.....	14
3.3.2 pcContextCreateMaster – create a context as the master	17
3.3.3 pcContextCreate – create a context as a slave	18
3.3.4 pcContextDestroy – destroy an existing context.....	19
3.3.5 pcContextGetInteger / pcContextGetString – retrieve a context property.....	19
3.3.6 pcContextSetInteger / pcContextSetString – set a context property.....	20
3.3.7 pcContextSync – synchronize context changes for all hosts.....	21
3.4 Frame Functions.....	21
3.4.1 Data Structures used by Frames	22
A.1.1 Sequencing of Frame Functions	22
3.4.2 pcFrameBegin – begin a new frame	23
3.4.3 pcFrameAddFramelet – add a framelet to the frame	24
3.4.4 pcFrameEnd – finished adding framelets to the frame	25
3.4.5 pcFrameResultChannel – retrieve resultant pixels for the frame.....	26
3.4.6 pcFrameResultQuery – test if a frame result if available.....	27
3.5 Miscellaneous Functions.....	27
3.5.1 pcGetProcAddress – get the address of an extension function.....	27
3.5.2 pcGetErrorString – get a text description of a PCerr.....	28
3.5.3 pcQueryExtension – query if an extension is present	28
4 Versions of the Library	29
5 Extensions	29
Currently Defined Extensions.....	30
5.1 PC_EXT_io_count – measure amount of network traffic.....	30

5.1.1	Additional Properties	30
5.2	PC_EXT_cur_gfx_ctx	31
5.2.1	Additional Properties	31
5.2.2	pcFrameAddGLFrameletEXT – add current pixels on graphics card	31
5.3	PC_HP_frame_output.....	31
5.3.1	Additional Properties	32
5.3.2	pcFrameWaitOutputHP – retrieve available output.....	32
5.3.3	pcResultGetChannelHP – get a channel of a result.....	33
5.3.4	pcResultDestroyHP – destroy a PCresult.....	33

Revision History

Revision	Date	Editor	Comment
0.0.01	4/18/2006	SRG	Initial version
0.0.02	4/20/2006	SRG	Added graphics card extensions
0.0.03	4/27/2006	SRG	Separated system & session; added pcFrameWait/QueryOutput
0.0.04	4/28/2006	SRG	editing changes
0.0.05	6/01/2006	SRG	changes from CEI review
0.0.06	6/01/2006	SRG	removed PC_EXT_LIB_GFX_CTX extension (CEI request)
0.0.07	6/20/2006	SRG	add PCresult
0.0.08	7/12/2006	SRG	add PCchannel
1.0.00	7/16/2006	SRG	Version 1.0
1.0.01	7/31/2006	SRG	a few typos and initial values for ctx properties
1.0.02	9/13/2006	SRG	spelling of extensions; args to pcFrameAddGLFrameLetEXT; pcSystemFinalize, PC_LIBRARY_PATH, versioning properties

1 Overview

The Parallel Compositing library (PC) is a set of functions that enable a set of graphics cards to work together to produce images. The benefits of using a set of graphics cards over a single card include:

- scaling up the graphic card resources that can be used over what is typically available on a single graphics card. These include:
 - scaling the texture memory available
 - scaling the overall size of the image produced
 - scaling the amount of geometry that can be processed
- scaling of the host system resources available over what is typically available in a single computer. These include:
 - amount of host memory
 - bandwidth between host memory and the graphics card

The PC library exploits a scaling technique called Sort-last¹ rendering. Using this technique, a number of graphics cards produce images. These images are then combined using various compositing operators to produce the final image.

While the PC library uses a particular scaling technique, this technique is, in fact, quite general. It can:

- create very large images
- do volume rendering
- support processing very large datasets
- work with either distributed or shared memory systems
- work on a variety of platforms and graphics cards
- be extended to support new networking technologies and compositing operators

¹ Sort-last rendering differs from another common technique called Sort-first rendering. In the case of sort-first rendering, the objects being drawn are culled or sorted before they are given to each of the graphics cards.

2 Definitions / Nomenclature

PC has a set of terms that will be used throughout the document. This section defines these terms and how they relate to one another.

2.1 Application

An Application is one or more programs that run on one or more computers that use the PC library to create images. In some cases all the programs may run on a single computer; in other cases the programs may be distributed across multiple computers. From the point of view PC library, all these programs create a session and then manipulate this session to do useful work.

2.2 System

A System is a computer and operating system environment that is capable of running one or more programs.

2.3 Session

A Session is the PC entity that connects all the programs in an Application. A session is identified by a property known as a *sessionId*. All programs that make up an Application create a session by specifying the same *sessionId*. A program can be a part of only one session at a time.

2.4 Host

It is best to think of a host as a thread of execution. A host differs from a program in that a program can have more than one thread. To contribute images to frames a program must have at least one host.

A host has a unique identifier that consists of 2 parts:

- A *hostName*: this is an ASCII name that is usually associated with the system where the host executes. Often the *hostName* is the TCP name for the system, but it need not be. In general, all other hosts running on different systems need to be able to figure out how to create a network connection to this host by using its hostname.
- A *hostId*: if an application wants more than one thread of execution running on a given system, it must distinguish those hosts using the *hostId*. *HostIds* are integers. Typically the threads of execution are numbered 0..n, however, the only requirement of the PC library is that multiple hosts that are part of the same context and running on the same system have unique *hostIds*.

2.5 Context

A *context* is a PC entity that connects all the hosts that will work together to produce a sequence of images. Once a program has created a *session*, it can create as many hosts as it needs. The hosts that will produce or consume a sequence of *frames* create a *context*. The *context* helps the hosts coordinate the creation and consumption of the sequence of *frames*.

The *context* holds the shared state among the hosts. This includes the size of the *frame* being produced and whether the *frame* is ready to be consumed or not.

A *session* can have one or more *contexts* based on the needs of the application. A specific host can be member of one or more *contexts*. See section 3.3 for more details.

2.6 Frame

A *frame* is a rectangular area of pixels being produced and consumed by the hosts in a *context*. The size of the *frame* and the data maintained for each pixel is documented in the *context*. The

size of the *frame* can change; however, it needs to change in a coordinated way that is understood by all the hosts. The PC library provides functions to coordinate these changes (see 3.3.7).

The data maintained for each pixel in a frame varies based on the type of pixels being used and the operator used to combine the pixels. PC exposes this data through the *pixel format* property of the context. An application can describe the format of the pixels it wishes to composite. The format describes the pixels that a program will give to and receive from the library. RGBA (byte of red color, byte of green color, byte of blue, byte of alpha) is an example for a pixel format.

2.7 Framelet

A *framelet* is a sub rectangle of a *frame*. Since PC supports a Sort-last rendering model, typically a *frame* is produced by compositing a set of pieces. Each of these pieces is called a *framelet*. A host can contribute 0 or more *framelets* to a *frame*. The host declares the dimensions, location on the *frame*, and number of the *framelets* it will contribute at the start of the production of a *frame*.

2.8 Output

An *output* is a sub-rectangle of a *frame* that a host wishes to retrieve once the *frame* has been produced. A host declares at most one *output* that it wants to retrieve. The dimension and location of this *output* on the *frame* are stored in the *context*. Two different hosts can declare the same or different *outputs*.

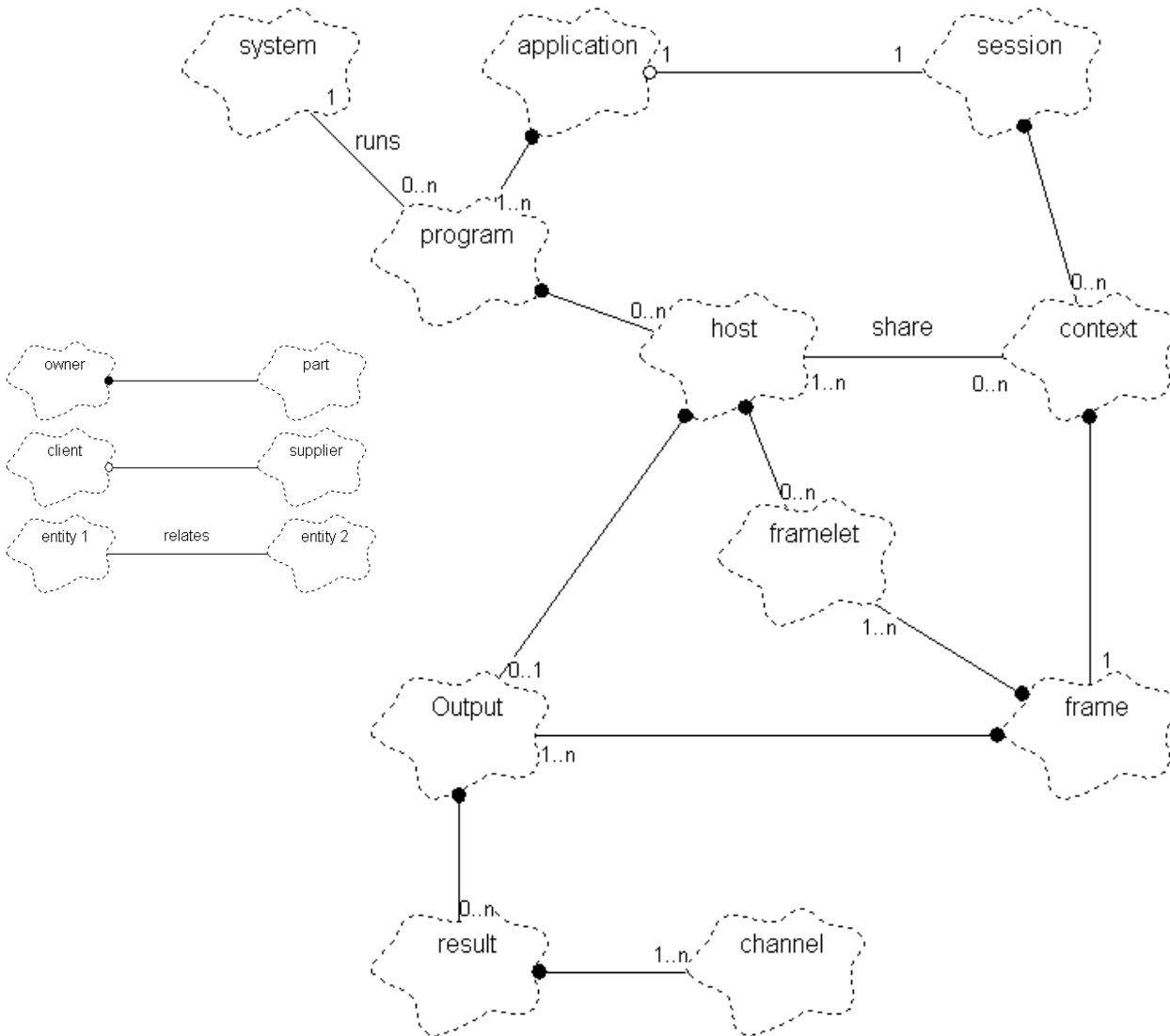
2.9 Result

A *result* is a sub-rectangle of an *output*. The host can retrieve its *output* either as a whole or in pieces. Each rectangle of pixels retrieved from the *output* is called a *result*. Based on the specific operator being used, a result will have one or more *channels* that can be retrieved. For example, a result associated with the depth operator has 2 channels:

- Color channel: composited pixels
- Depth channel: corresponding depth value of each pixel

A host can retrieve zero or more *results* from the *output* it has declared in the *context*.

2.10 Relationships Between PC Terms



The diagram above illustrates the relationships among the entities defined by PC. The clouds name the entities. The lines state relationships between objects. As illustrated in the key, the relationships are of 3 types:

1. ownership (black dot): the entity with the dot owns the entity on the other end
2. use (white dot): the entity with the dot uses the entity on the other end
3. relates (no dot): the 2 entities are related by the verb that labels the relationship – such a relationship is read from left to right (e.g. one or more hosts share a context)

2.11 Order of Function Calls

Applications that use the PC library want to create a sequence of images. PC calls each image a frame. As discussed above, the generation of a frame is typically done by a set of hosts that each produce a part of each frame. As a consequence, calls to the PC functions typically follow a

certain sequence. Sometimes PC dictates the sequence of calls; sometimes what is outlined below is simply a common sequence.

1. create a session
2. for each host
 - a. create a context
 - b. set/get context properties
 - c. for each frame
 - i. [update context properties]
 - ii. begin the frame
 - iii. [for each framelet]
 1. render the pixels for the framelet
 2. add the framelet to frame
 - iv. end the frame
 - v. [for each result]
 1. retrieve the channels of the result
 - d. destroy the context
3. destroy the session

3 PC Functions

This section describes the functions in the PC library.

3.1 PC Function Parameter Types

3.1.1 Parameter Types

The following parameter types are used by functions in the PC library:

PC type	properties of the type
PCerr	unsigned integer representing the status of a library call Call <code>pcGetErrorString</code> for a text string describing a status.
PCid	unsigned integer representing an instance of a PC object such as a frame
PCint	signed integer
PCuint	unsigned integer
PCenum	fixed set of integer values that typically have predefined meanings. The list of properties for a context is an example of a PCenum.
PCstring	null terminated sequence of characters
PCvoidp	an unspecified memory area
PCcontext	an opaque value representing an instance of a PC context. A PCcontext is need on PC calls where it is necessary to identify the context to be manipulated by the library.
PCchannel	a structure that describes a result. (see 3.1.2)

3.1.2 PCchannel Type

As described in 2.9, PC uses arrays to describe pixels that are to be composited by the library and then returned to a program through a result. Since these arrays are managed by the library, the program needs various information to read information in the array. This information is distilled into a PCchannel structure whose fields are described below.

PCchannel Field Table

Result Property	Meaning
channelType	the type of channel described by the channel structure
xOffset	the x offset of the origin of the result within the output
yOffset	the y offset of the origin of the result within the output
width	the width of the result in pixels
height	the height of the result in pixels
pixelFormat	format of the pixels in the result
frameId	the frame id associated with the result
size	the number of bytes between the start of adjacent array elements
alignment	the memory alignment of the first array element in a row. <ul style="list-style-type: none"> • 1 = byte aligned • 2 = even byte aligned (least significant bit of address is 0) • 4 = quad byte aligned (2 least significant bits are 0) • 8 = octa byte aligned (3 least significant bits are 0)
rowLength	the number of element between the start of row n and row n+1 of the array
address	the address of the origin of the array

Channel Types Tables

Channel Type	Meaning
PC_CHANNEL_COLOR	the channel holding the color values
PC_CHANNEL_DEPTH	the channel holding the depth values

3.2 System & Session Functions

Function	Purpose
pcSystemInitialize	Initialize PC library for a program
pcSystemFinalize	Uninitialize PC library for a program
pcSystemGetInteger	Retrieve a system property as an integer
pcSystemGetString	Retrieve a system property as a string
pcSessionCreate	Create a PC session
pcSessionDestroy	Destroy a PC session

3.2.1 pcSystemInitialize – Initialize PC for a Program

Synopsis

```
PCerr pcSystemInitialize(const PCstring searchPath)
```

Parameters

searchPath	a pointer to a string that specifies a series of directories to search for the PC shared libraries before looking in the standard system directories.
------------	---

Return Values

This function returns PC_NO_ERROR on successful completion.

Description

This function is used to initialize a program's usage of the PC library. This function is called to change the series of directories searched for shared libraries supplying PC functions and extensions. If called, it must be called prior to calling any other PC function.

PC looks for the following shared libraries:

- one supporting the network transport layer
- one supporting the remaining PC functions

PC searches for the network transport layer shared library in the following directories in the following order:

1. if defined, the list of directories specified by the environment variable `PC_TRANSPORT_PATH`
2. if `pcSystemInitialize` is called, the list of directories specified by the parameter `searchPath`
3. the operating system defined series of directories

PC searches for the PC functions shared library in the following directories in the following order:

1. if defined, the list of directories specified by the environment variable `PC_LIBRARY_PATH`
2. if `pcSystemInitialize` is called, the list of directories specified by the parameter `searchPath`
3. the operating system defined series of directories

The names of the PC specific shared libraries are implementation defined, but will be different. The exact format of the `searchPath` parameter, environment variable values, directory delimiters, and operating system mechanism is not defined by this spec. A NULL or empty `searchPath` parameter indicates `pcSystemInitialize` is adding no additional directories to search.

When called, this function will attempt to bind the shared libraries used by the program. If this function is not called, a call to any other system or session function will implicitly attempt to bind the shared libraries used by the program.

An implementation may limit the `searchPath` to a single directory.
An implementation may require a program call `pcSystemInitialize`.

3.2.2 `pcSystemFinalize` – Uninitialize PC for a Program

Synopsis

```
PCerr pcSystemFinalize()
```

Parameters

none	
------	--

Return Values

This function returns `PC_NO_ERROR` on successful completion.

Description

This function unloads the PC library so it can no longer be used by a program. Calling this function is optional. If it is not called, the PC Library remains loaded until the program exits. The typical reason for calling this function is to make it possible to reinitialize the PC Library with an alternate search path. This allows an alternate version of the PC Library to be loaded.

3.2.3 pcSessionCreate – Create a PC session

Synopsis

```
PCerr pcSessionCreate(PCid sessionId)
```

Parameters

sessionId	an unsigned integer that acts as a unique identifier of the session. All programs that are a part of an application must call this function with the same identifier. Just how unique must this identifier be? If there are multiple sessions that are sharing a network at a given time, each session must have a different sessionId.
-----------	--

Return Values

This function returns PC_NO_ERROR on successful completion.

Description

This function creates a PC session.

To use the PC library, a program must create a session. This function must be called before creating any contexts. A program may create only one session at a time. pcSessionDestroy may be called to delete a session.

In order to work together to produce a set of images, the programs of an application must create a session with the same sessionId.

3.2.4 pcSessionDestroy – Destroy a PC Session

Synopsis

```
PCerr pcSessionDestroy()
```

Parameters

none	
------	--

Return Values

This function returns PC_NO_ERROR on successful completion.

Description

This function ends the active PC session. This function destroys all the active contexts that are a part of the session before returning.

3.2.5 PC System Properties

A system used by a program has a set of properties. These properties tell the program about the environment where it is running. For example, the program can determine the extensions supported by the PC library on the system or what networks are available for the programs to use.

The only property that can be set by a program is PC_SESSION_PATH. This property is set by pcSystemInitialize.

Some properties have multiple values and you need to qualify which value you want. An example is PC_NETWORK_NAME which is qualified by the integer index assigned to a specific network by the PC library. Varying the qualifier between 0 and PC_NUM_NETWORKS - 1, a program can discover all the networks supported by the library on this system.

A number of the properties can be retrieved in both an integer and string form.

Two versions of the PC library that return different values for system properties may not be compatible.

Property	Qualifier	Type	Can be Set	Meaning
PC_LIB_VERSION	none	integer	no	PC_SPEC_MAJOR*100 + PC_SPEC_MINOR
PC_LIBRARY_PATH	none	string	yes	The full directory path of the loaded library. This includes the actual library file. This property is set by pcSystemInitialize.
PC_NUM_NETWORKS	none	integer	no	Number of network transport layers available for the program to use. The program can control the transport layer used with the PC_NETWORK_ID property when creating a context.
PC_NETWORK_ID	yes	integer	no	A library assigned value that represents a particular network transport layer. You qualify the network layer of interest by specifying the index of the network. The index will be a value between 0 and PC_NUM_NETWORKS-1.
PC_NETWORK_NAME	yes	string	no	A library assigned value that describes a particular network transport layer. You qualify the network layer of interest using the same technique as PC_NETWORK_ID.
PC_SPEC_MAJOR	none	integer	no	The major and minor version of this spec supported by the current implementation.
PC_SPEC_MINOR	none	integer	no	
PC_VENDOR	none	string	no	A string listing the vendor who created the core part of the library.
PC_VERSION	none	string	no	A string describing the current implementation of the Library as described in section 4
PC_SESSION_PATH	none	string	yes	The list of directories that are

				searched to find the library and any extensions. You can set the property using <code>pcSystemInitialize</code> .
PC_EXTENSIONS	none	string	no	The list of extension modules currently loaded.
PC_VOLATILE_FRAMELET_LIMIT	none	integer	no	The maximum value of PC_VOLATILE_FRAMELET supported by the implementation
PC_RETAIN_OUTPUT_LIMIT	none	integer	no	The maximum value of PC_RETAIN_OUTPUT supported by the implementation

3.2.6 `pcSystemGetInteger` / `pcSystemGetString` – retrieve a system property

Synopsis

```
PCerr pcSystemGetInteger
    (PCenum property, PCint qualifier, PCint* value)
PCerr pcSystemGetString
    (PCenum property, PCint qualifier, PCstring* value)
```

Parameters

property	the property to retrieve. The names of the supported properties are listed in the table in section 3.2.5.
qualifier	the qualifier of the property to retrieve. See the table in 3.2.5 for the properties that require qualifiers and the meaning of the qualifier. For those properties that do not require a property, this parameter is ignored.
value	the variable to hold the property on return from the function. For <code>pcSystemGetInteger</code> , the property's value is stored in the value parameter. For <code>pcSystemGetString</code> , the address of the property is stored in the value parameter. The program must <u>not</u> free the storage holding the string.

Return Values

These functions return `PC_NO_ERROR` on successful completion. *<tbl:return values>*

Description

These functions are called to return system properties.

3.3 Context Functions

Function	Purpose
<code>pcContextCreateMaster</code>	Create a context as the master
<code>pcContextCreate</code>	Create a context when host is not the master
<code>pcContextDestroy</code>	Destroy an existing context
<code>pcContextGetInteger</code>	Retrieve a context property as an integer
<code>pcContextGetString</code>	Retrieve a context property as a string
<code>pcContextSetInteger</code>	Set an integer context property

pcContextSetString	Set a string context property
pcContextSync	Synchronize context changes for all hosts

Parallel compositing is based on the concept of multiple processes/threads creating parts of an image that are then combined and displayed. PC calls each of these processes/threads a host. To coordinate their work the hosts share a context. Each host creates the context using a context create function. The hosts can share information and coordinate their execution through the context.

One of the hosts is the master. The remaining hosts are called slaves. The distinction between the master and the slaves from a PC library point of view is minimal. All hosts have access to the full API with one exception. The master creates the context by calling pcContextCreateMaster. This function specifies all the hosts that will be members of the context. The slaves create the context by calling pcContextCreate.

A host is identified by a tuple:

hostName:hostId

where hostName is the name of the system that will run the host

hostId is an integer used to distinguish 2 hosts running on the same system

this tuple is frequently represented as a string with a colon separating the 2 parts. For example:

red4:2 (system red4, hostId 2)

n1.cei.com:0 (system n1.cei.com, hostId 0)

Each host that is a member of a context must have a unique hostName:hostId tuple. Two hosts with the same hostName:hostId pair but in different contexts are unrelated from the point of view of the PC library. The PC library must be able to create a network connection to a machine using hostName. How this happens may vary based on the network used.

3.3.1 PC Context Properties

A PC session can contain one or more contexts. A context is a shared space among the hosts that are members of the context. Each of the hosts can set and retrieve properties of the context.

A specific property is either global or host specific. Global properties have a single value that is the same for all hosts in the context. A host specific property can have a different value for each host. The table below notes whether a property is global or host specific in the *Host Specific* column.

Some properties cannot be set when the context is created. The *On Create* column in the table below indicates whether a property can be set when the context is created.

Some properties cannot be changed once the context is created. The *Can be Changed* column in the table below indicates whether a property can be changed after the context is created.

Using On Create and Can be Changed we can make some observations about properties. Also note that only the master can specify properties when creating a context.

Observation	On Create	Can be Changed	Example
Read only	No	No	PC_HOSTINDEX
Must be set on context create	Yes	No	PC_PIXEL_FORMAT
Must be set after context create	No	Yes	PC_OUTPUT_X_OFFSET
Can be set anytime	Yes	Yes	PC_FRAME_WIDTH

3.3.1.1 Standard Context Properties

These properties exist in all implementations.

Standard Context Properties

Property	Host Specific	Type	On Create Can be Changed		Meaning
			On Create	Can be Changed	
PC_FRAME_WIDTH	no	integer	yes	yes	Overall height and width of the frame created by the context. The height and width are specified in pixels. Default value: 256
PC_FRAME_HEIGHT	no	integer	yes	yes	
PC_PIXEL_FORMAT	no	integer	yes	no	The type of the pixels in the frame. Each pixel has an encoding for the color of that pixel and any additional information such as depth or transparency that may be needed to composite the pixels. Both encodings are captured in this property. Specific values of this property are listed in the Pixel Format Table below. Default value: none
PC_COMPOSITE_TYPE	no	integer	yes	no	Specifies how the pixels will be composited. Specific values of this property are listed in the Composite Type Table below. Default value: none
PC_COMPRESSION_HINT	no	integer	yes	yes	Directs the library to attempt to compress pixels being transferred between hosts. The possible values of this property are: <ul style="list-style-type: none"> • PC_COMPRESSION_NONE • PC_COMPRESSION_LOW • PC_COMPRESSION_MED • PC_COMPRESSION_HIGH When set, this property is only a suggestion to the implementation. Default value: PC_COMPRESSION_NONE
PC_NETWORK_ID	no	integer	yes	no	Specifies the network to use to transfer images between the hosts when they run on different systems. The acceptable values for this property are defined by the system property PC_NETWORK_ID. Default value: 0 (the default network)
PC_OUTPUT_DEPTH	no	integer	yes	yes	Directs the library to be prepared to include pixel depth information when a program requests a result. Default value: 0 (no depth in output)
PC_OUTPUT_X_OFFSET	yes	integer	no	yes	Different hosts can request different outputs from the frame. These properties control the area of the frame that can be retrieved as a result on a particular host. The values of these properties must specify a rectangular area within the frame described by PC_FRAME_WIDTH and PC_FRAME_HEIGHT. PC_OUTPUT_WIDTH or PC_OUTPUT_HEIGHT set to 0 indicate no output is needed by the host.
PC_OUTPUT_Y_OFFSET	yes	integer	no	yes	
PC_OUTPUT_WIDTH	yes	integer	no	yes	
PC_OUTPUT_HEIGHT	yes	integer	no	yes	

					Default value: 0 for x and y offset Default value: 256 for width and height
PC_NUM_HOSTS	no	integer	no	no	The number of hosts in the context.
PC_HOSTINDEX	yes	integer	no	no	This property is the index for the current host. Each host has an index between 0 and PC_NUM_HOSTS-1. The index assigned to a host is its position in the array passed by the master to pcContextCreateMaster. When requesting a host specific property, you do so by providing the index that corresponds to the host.
PC_HOSTID	yes	integer	no	no	Each host has a hostId and hostName. These are all specified when the context is created on the master by calling pcContextCreateMaster. The hostname:hosted for each host are an element of this array.
PC_HOSTNAME	yes	string	no	no	
PC_SESSION_ID	no	Integer	no	no	The session id for the current session as set by the function pcSessionCreate.
PC_RETAIN_OUTPUT	yes	integer	yes	no	Normally the output of a frame is only available between pcFrameEnd and the following pcFrameBegin. This property relaxes this constraint and lets the caller access the output for the number of frames designated by this property. The value of this property cannot exceed the value of the system property PC_RETAIN_OUTPUT_LIMIT. See section 3.4.5 for the details Default value: 1
PC_VOLATILE_FRAMELET	yes	integer	yes	no	Normally framelets are copied by the PC library when passed as a parameter to a function call. This property relaxes this constraint potentially allowing the library to be more efficient. The property is an integer stating how many frames in the future the application will reuse the buffer. The value of this property cannot exceed the value of the system property PC_VOLATILE_FRAMELET_LIMIT See section 3.4.3 for the details. Default value: 0

The PC_PIXEL_FORMAT property describes the pixels in the frame. This property is logically 2 values:

- A color value that can be extracted using the mask PC_PF_COLOR_MASK
- An additional data value than can be extracted using the mask PC_PF_DEPTHMASK

Predefined constants are defined for each of these logical values that can be OR'ed together to construct a value for PC_PIXEL_FORMAT.

Pixel Format Table

Pixel Format	Color vs Depth	Meaning
PC_PF_BGRA8	color	4 byte pixel consisting of color in BGR order + 8bit alpha value
PC_PF_RGBA8	color	4 byte pixel consisting of color in RGB order + 8bit alpha value
PC_PF_BGR8	color	3 byte pixel consisting of color in BGR order – there is no 4 th byte
PC_PF_Z32I	depth	additional 4 bytes of depth data as 32bit integer
PC_PF_Z32F	depth	additional 4 bytes of depth data as 32bit float
PC_PF_Z24I	depth	additional 3 bytes of depth data as 24bit integer

PC_PF_Z8I	depth	additional 1 byte of depth data as 8bit integer
-----------	-------	---

The PC_COMPOSITE_TYPE property describes how the pixels from multiple hosts are to be composited.

Pixel Compositing Table

Compositing Algorithms	Meaning
PC_COMP_DEPTH	The color of the pixel closest to the eye masks a more distant pixel
PC_COMP_ALPHA_SORT	The color of the pixel is a blending of the colors of the 2 pixels. The blending is dictated by the transparency (alpha value) of the 2 pixels

3.3.1.2 Extended Context Properties

The following properties are in various PC extensions. If an implementation does not support one of the features, retrieving or setting the property should return the error PC_UNIMPLEMENTED.

Extended Context Properties

Property	Type				Meaning
	Host Specific		On Create	Can be Changed	
PC_RETAIN_BUFFERS	no	integer	yes	yes	to be supplied
PC_CONTEXT_ID	no	Integer	yes	no	An integer id uniquely assigned by the library to each context within a session. Any host that is a member of a context returns the same value. Any 2 hosts that are members of different contexts return different values.

3.3.2 pcContextCreateMaster – create a context as the master

Synopsis

```
PCerr pcContextCreateMaster
(const PCint *properties, PCstring *hosts, PCstring thisHost,
PCcontext *ctx)
```

Parameters

properties	an array of initial properties for the context. The array is a series of value pairs where: <ul style="list-style-type: none"> the first element of the pair is the name of a property listed in section 3.3.1. Terminate the array pairs by using the constant PC_PROPERTY_END as the first element. the second element is the initial value of the property Only properties marked "On Create" can be used in this array.
hosts	an array describing the hosts of this context. The array has one element per host. Each host is described using a string with the format:

	hostName:hostId The end of the array is indicated by an array element with a NULL value.
thisHost	the name of this host in the format: hostName:hostId This host must be one of the hosts in the hosts parameter
ctx	a variable to hold the context that is created

Return Values

This function returns PC_NO_ERROR on successful completion. <td:return values>

Description

This function is used by the master host to create a context. The master's name (thisHost) must be one of the hosts listed in hosts parameter. The properties parameter specifies the initial properties of the context. Certain parameters can only be specified when the context is created. These properties may only be specified using the properties parameter.

This function will not return until all hosts have created the context or the library encounters an error creating a context. On a successful return, the parameter ctx will be set to a PCcontext value that can be used in other functions requiring a context.

3.3.3 pcContextCreate – create a context as a slave

Synopsis

```
PCerr pcContextCreate
(PCid netId, PCstring thisHost, PCcontext* ctx)
```

Parameters

netId	the ID of the network transport layer for this context. It must match the value of the PC_NETWORK_ID property passed to pcContextCreateMaster(). To use the default network, specify PC_ID_DEFAULT (the first transport layer).
thisHost	the name of this host in the format: hostName:hosted The master via the hosts parameter specified the members of the context. thisHost must be on of these hosts
ctx	a variable to hold the context that is created

Return Values

This function returns PC_NO_ERROR on successful completion. <td:return values>

Description

This function is used by a slave host to create a context. The slave's name (thisHost) must be one of the hosts listed by the master in its hosts parameter.

This function will not return until all hosts have created the context or the library encounters an error creating a context. On a successful return, the parameter ctx will be set to a PCcontext value that can be used in other functions requiring a context.

3.3.4 pcContextDestroy – destroy an existing context

Synopsis

```
PCerr pcContextDestroy(PCcontext ctx)
```

Parameters

ctx	the context to destroy
-----	------------------------

Return Values

This function returns PC_NO_ERROR on successful completion. <td>return values</td>

Description

This function is used to destroy an existing context. It must be called by all hosts that created the context originally. The function will block until all hosts have done so. This function cannot be called between pcFrameBegin and pcFrameEnd calls on the same context.

Calling pcSessionDestroy destroys all contexts in a session. It is an alternate means of destroying a context.

3.3.5 pcContextGetInteger / pcContextGetString – retrieve a context property

Synopsis

```
PCerr pcContextGetInteger
(PCcontext ctx, PCenum property, PCint hostIndex, PCint* value)
PCerr pcContextGetString
(PCcontext ctx, PCenum property, PCint hostIndex, PCstring* value)
```

Parameters

ctx	the context whose properties are to be retrieved
property	the property to retrieve. The names of the supported properties are listed in the table in section 3.3.1
hostIndex	for host specific properties (marked as such in the context properties table), this parameter lets you specify the index of the host whose property you want. Each host is assigned a hostIndex between 0..PC_NUM_HOSTS-1. A few hostIndex constants are also defined by PC that can be used for this parameter: <ul style="list-style-type: none"> • PC_LOCALHOST_INDEX = hostIndex of host making the call • PC_MASTER_INDEX = hostIndex of the master host • PC_INDEX_DEFAULT = PC_LOCALHOST_INDEX for global properties, this parameter is ignored
value	the variable to hold the property on return from the function. For pcContextGetInteger, the property's value is stored in the value parameter. For pcContextGetString, the address of the property is stored in the value parameter. The program must <u>not</u> free the storage holding the string.

Return Values

These functions return PC_NO_ERROR on successful completion. <td:return values>

Description

These functions are called to retrieve the current value of a context property. Context properties fall in one of two categories:

- Global: meaning the properties is the same for all hosts
- Host Specific: meaning the property can be different for each host

The tables in section 3.3.1 list whether a property is global or host specific. The parameter hostIndex is ignored when you request the value of a global property. For a host specific property, the parameter hostIndex specifies which host's property you wish to retrieve. Several additional constants are also provided for common hostIndexes. These can be used in lieu of the specific index for these common hosts.

3.3.6 pcContextSetInteger / pcContextSetString – set a context property

Synopsis

```
PCerr pcContextSetInteger  
(PCcontext ctx, PCenum property, PCint hostIndex, PCint value)
```

```
PCerr pcContextSetInteger  
(PCcontext ctx, PCenum property, PCint hostIndex, PCstring value)
```

Parameters

property	the property to set. The names of the supported properties are listed in the table in section 3.3.1 (those marked as “can be changed”)
hostIndex	for host specific properties (marked as such in the context properties table), this parameter lets you specify the index of the host whose property you want. Each host is assigned a hostIndex between 0..PC_NUM_HOSTS-1. A few hostIndex constants are also defined by PC that can be used for this parameter: <ul style="list-style-type: none">• PC_LOCALHOST_INDEX = hostIndex of host making the call• PC_MASTER_INDEX = hostIndex of the master host• PC_INDEX_DEFAULT = PC_LOCALHOST_INDEX for global properties, this parameter is ignored
value	the value to assign to the property

Return Values

These functions return PC_NO_ERROR on successful completion. <td:return values>

Description

These functions are called to set the current value of a context property. Context properties fall in one of two categories:

- Global: meaning the properties is the same for all hosts
- Host Specific: meaning the property can be different for each host

The tables in section 3.3.1 list whether a property is global or host specific. The parameter hostIndex is ignored when you set the value of a global property. For a host specific property, the parameter hostIndex specifies which host's property you wish to set. Several additional

constants are also provided for common hostIndexes. These can be used in lieu of the specific index for these common hosts.

Not all context properties can be set by these functions. First, certain properties can only be set when the context is created. See section 3.3.1 for additional details on setting context properties. Second, context parameters cannot be set between pcFrameBegin and pcFrameEnd.

3.3.7 pcContextSync – synchronize context changes for all hosts

Synopsis

```
PCerr pcContextSync(PCcontext ctx)
```

Parameters

ctx	the context whose properties are to be synchronized
-----	---

Return Values

This function returns PC_NO_ERROR on successful completion. *<tdb:return values>*

Description

The PC library supports any of the hosts changing the values of context properties. Since each of the hosts can be running on a separate machine, a mechanism is needed to coordinate the any changes across all the hosts that are members of the context. This is the purpose of pcContextSync.

The function is designed to be used as follows. Hosts make changes to the context by calling pcContextSetInteger or pcContextSetString. Typically one host changes global properties such as PC_FRAME_HEIGHT and PC_FRAME_WIDTH. This host may also set the host specific PC_OUTPUT... properties for each host. Following these changes, the host calls pcContextSync. The remaining hosts that wish to receive these changed properties will call pcContextSync followed by a series of pcContextGetInteger or pcContextGetString calls.

This function effectively acts as synchronization barrier to coordinate these calls. If pcContextSync is called by one host, it must be called by all hosts. At the point the host setting the properties calls pcContextSync, the changes made by this host are flushed to all of the hosts. The hosts reading the context properties don't progress beyond the pcContextSync until the changed parameters have been received.

To be a bit more precise,

- hosts do not block on the pcContextSync call. Instead they will block (if necessary) on any attempt to use a context property until after the context has been updated on that host.
- there is an implicit pcContextSync executed immediately following a pcFrameBegin call, so there is no need to sync before starting a frame.
- if more than one host sets a specific context property, the results are undefined
- a host may not call pcContextSync between pcFrameBegin and pcFrameEnd

3.4 Frame Functions

Function	Purpose
pcFrameBegin	Begin a new frame
pcFrameAddFramelet	Add a framelet to the frame
pcFrameEnd	Finished adding framelets to frame

pcFrameResultChannel	Retrieve a channel of a result (e.g. color data) for a frame
pcFrameResultQuery	Query if a result within a frame is available

The purpose of the PC library is to create a sequence of frames. The production of frames revolves around a number of concepts:

- Frame (see 2.6 for definition)
- Framelets (see 2.7 for definition)
- Outputs (see 2.8 for definition)
- Results (see 2.9 for definition)
- Channels (see 2.6 for definition)

3.4.1 Data Structures used by Frames

The properties of a frame are shared among all hosts within a context. These include a width and height of the frame and the type of pixels generated to fill the frame. These properties are stored in the context.

Framelets are represented in PC by one or more a 2-dimensional array of pixel values. The actual number of arrays varies based on the operator used to composite the pixels (PC_COMPOSITE_TYPE). Each array is described in PC as a channel. The table below lists the channels needed for each operator.

Operators	Channels Needed by the Operator
PC_COMP_ALPHA_SORT	PC_CHANNEL_COLOR
PC_COMP_DEPTH	PC_CHANNEL_COLOR PC_CHANNEL_DEPTH

An output is also represented by the same 2-dimensional arrays. The PC_CHANNEL_COLOR is always created. PC_CHANNEL_DEPTH is only created if needed by the operator and the program requests the channel be supplied in the output. The context property PC_OUTPUT_DEPTH set to true (non-zero) requests the library to provide the PC_CHANNEL_DEPTH for all outputs.

A result is a sub-rectangle of an output, so it is also represent by the same 2-dimensional arrays.

Although 4 concepts are defined (frame, framelet, output, result), the frame functions only deal with 2 directly. Each host may present zero or more framelets to the PC library to be composited. Each host may request zero or more results.

The memory used for framelets is allocated and freed by the program calling the PC library. The memory used for results is allocated and freed by the PC library. Access to results is enabled by the library through a special PC type called a PCchannel.

The function pcFrameResultChannel has parameters that specify a result and a channel. It then returns the requested channel of the result using the structure PCchannel described in section 3.1.2. This structure includes the address of channel array, the size of elements and the number of array elements in a row. This information makes it possible to navigate the elements of the array or use the channel as input to a graphic function such as the OpenGL glDrawPixels function.

A.1.1 Sequencing of Frame Functions

Using the PC library typically involves each host performing the sequence of steps described in section 2.11.

This section describes the PC library functions in step 2.c of section 2.11.

All hosts must call `pcFrameBegin` for each frame. `pcFrameBegin` ends in an implicit call to `pcContextSync` to synchronize any changes to the frame's properties.

Those hosts producing framelets will call `pcFrameAddFramelet`.

All hosts must call `pcFrameEnd` for each frame to indicate that all framelets are ready to be composited.

Those retrieving output will call `pcFrameResult`. This function will not return until the pixels being requested are available.

The points of synchronization among the hosts are 2:

1. A need for a property in the context following `pcFrameBegin`
2. A need for a frame result

Other than these 2 points, the individual hosts can progress independently and in parallel.

Many of the frame functions take a `frameId` as a parameter. This parameter identifies which of the series of frames generated by a program the function call is referencing. `frameId`'s are generated by `pcFrameBegin` and then referenced by the remaining functions.

Each frame (and thus corresponding `frameId`) has a limited lifetime when it can be referenced. Without using any of the PC extensions, a frame's lifetime is between the `pcFrameBegin` call when it is generated and the next `pcFrameBegin` call when the next frame is generated. Trying to reference the frame outside this period will result in an error. The extension context parameters `PC_VOLATILE_FRAMELET` and `PC_RETAIN_OUTPUT` extend the lifetime of the frame for a requested number of frames see 3.4.3 and 3.4.5 for more details.

3.4.2 `pcFrameBegin` – begin a new frame

Synopsis

```
PCerr pcFrameBegin
(PCcontext ctx, PCid* frameId, PCuint numFramelets,
 PCuint* xOffset, PCuint* yOffset, PCuint* width, PCuint* height,
 PCuint* order)
```

Parameters

<code>ctx</code>	the context this frame belongs to
<code>frameId</code>	a unique id generated by the PC library for this frame. This id is used on subsequent calls to the library to identify the call as being for this frame.
<code>numFramelets</code>	the number of framelets that will be generated by this host for this frame
<code>xOffset</code>	is an array giving the horizontal offsets of each framelet within the frame.
<code>yOffset</code>	is an array giving the vertical offsets of each framelet within the frame.
<code>width</code>	is an array giving the width of each framelet within the frame
<code>height</code>	is an array giving the height of each framelet within the frame
<code>order</code>	is an array giving the order of each framelet within the frame

Return Values

This function returns PC_NO_ERROR on successful completion. *<td:return values>*

Description

This function is used to begin compositing for a frame. It corresponds to step 2.c.ii in section 2.11. It must be called by all host members of the context specified by the parameter ctx.

Each host may contribute 0 or more pixel contributions to the frame. Each of these contributions is called a framelet and has the following properties:

Framlet Property	Meaning
xOffset	the x offset of the origin of the framelet within the frame
yOffset	the y offset of the origin of the framelet within the frame
width	the width of the framelet in pixels
height	the height of the framelet in pixels
pixel format	implicitly given by the context property PC_PIXEL_FORMAT
order	some types of compositing (notably alpha blending) have an ordering among the framelets being composited – this property specifies the order with respect to other framelets

The parameter numFramelets states the number of framelets that will be added by this host before it calls pcFrameEnd. The remaining parameters describe the properties of each framelet that will be added. The first element in each array gives the properties of first framelet that will be added by pcFrameAddFramlet; the second element in each array gives the properties of second framelet to be added; etc.

The framelets described must all be within the frame described by the context properties PC_FRAME_WIDTH and PC_FRAME_HEIGHT. The parameter order is ignored by all operators that do not require order be specified. If all of the framelets will cover the entire frame (i.e. xOffset=0, yOffset=0, width=PC_FRAME_WIDTH, height=PC_FRAME_HEIGHT), then it is acceptable to pass NULL for all of these arrays.

On success, this function returns a frame ID number in the parameter frameId.

3.4.3 pcFrameAddFramelet – add a framelet to the frame

Synopsis

```
PCerr pcFrameAddFramlet  
(PCcontext ctx, PCid frameId, PCvoidp color, PCvoidp depth)
```

Parameters

ctx	the context this frame belongs to
frameId	the unique id generated for this frame by pcFrameBegin
color	a pointer to the buffer containing the color pixels for this framelet
depth	a pointer to the buffer containing the depth data for the pixels of this framelet

Return Values

This function returns PC_NO_ERROR on successful completion. *<td:return values>*

Description

This function adds a framelet to the frame and corresponds to step 2.c.iii.2 in section 2.11.

The number of times this function is called for this frame is given by the numFramelets parameter specified on this frame's pcFrameBegin call. This might be 0 if this host is not rendering any framelets. The i^{th} call to this function specifies the pixel data that corresponds to the i^{th} framelet. Note that the number of framelets per frame can change from frame to frame.

The color and depth parameters point to arrays of pixel data for this framelet. The format of this data is dictated by the context property PC_PIXEL_FORMAT. In general, you want to choose a format that is efficient for the application to generate. If the type of compositing specified by the context property PC_COMPOSITE_TYPE requires no depth data, depth should be NULL.

The buffers passed to pcFrameAddFramelet are normally not modified by the PC library and can be reused by the caller on return from this function.

The context property PC_VOLATILE_FRAMELET grants the PC library the right to relax the normal behavior. If the value of PC_VOLATILE_FRAMELET is greater than 0, the library is permitted to modify the framelet and the caller won't modify the framelet until after PC_VOLATILE_FRAMELET calls to pcFrameBegin. An implementation may restrict the value of property to a small positive integer. The system property PC_VOLATILE_FRAMELET_LIMIT specifies the maximum value of this property supported by the implementation.

The default value for this property is 0. This value requires the PC library to extract any information it needs from the framelet before returning to the caller. A value of 1 states that the library can continue to extract information and modify the framelet until the next pcFrameBegin call. A value of 2 supports double buffering compositing. In this case, the calling program could be supplying a framelet for frame $n+1$ while the library is still compositing frame n .

3.4.4 pcFrameEnd – finished adding framelets to the frame

Synopsis

```
PCerr pcFrameEnd(PCcontext ctx, PCid frameId)
```

Parameters

ctx	the context this frame belongs to
frameId	the unique id generated for this frame by pcFrameBegin

Return Values

This function returns PC_NO_ERROR on successful completion. *<td:return values>*

Description

This function marks the end of adding framelets to the frame and corresponds to step 2.c.iv in section 2.11.

This function must be called by all hosts, even if they did not provide any framelets to composite. After this call is made, a host is no longer allowed to make pcFrameAddFramelet() calls for the frame specified by the parameter frameId.

After this function call is made and before the next pcFrameBegin() call is made, the application may call pcFrameResult() to retrieve the composited pixels.

3.4.5 pcFrameResultChannel – retrieve resultant pixels for the frame

Synopsis

```
PCerr pcFrameResultChannel  
    (PCcontext ctx, PCid frameId,  
     PCuint xOffset, PCuint yOffset, PCuint width, PCuint height,  
     PCuint channelType, PCchannel channel)
```

Parameters

ctx	the context this frame belongs to
frameId	the unique id generated for this frame by pcFrameBegin
xOffset	the x offset in pixels from the origin of the output
yOffset	the y offset in pixels from the origin of the output
width	the width of the result in pixels
height	the height of the result in pixels
channelType	the channel to return
channel	a PCchannel that will be initialized for accessing the channel specified by this function

Return Values

This function returns PC_NO_ERROR on successful completion. <tbid:return values>

Description

This function retrieves output pixels of the frame and corresponds to step **2.c.v.1** in section **2.11**.

This function will block until the pixel information requested by the parameters is available. If a program needs to check on the status of a result without blocking, it can call the function pcFrameResultQuery().

The parameters of the function describe the pixel information to be returned. The first two parameters (ctx and frameId) identify the frame of interest. This call must be made following the host's call to pcFrameEnd for the frame specified by the parameter frameId.

The next four parameters (xOffset, yOffset, width and height) described the result's location on the output. The xOffset and yOffset are relative to the origin of the output. The result must be within the boundaries of the output described by the context properties for this host.

The final two parameters (channelType and channel) describe the type of pixel information to be delivered to the calling program. channelType state the channel (color or depth) to be deliver and on a successful return the actual channel information is provided to the calling program in the parameter channel.

The buffers produced by this function are 2-dimensional arrays. The elements of the array are not necessarily contiguous. Element (x,y) in any channel buffer can be retrieved as follows:

```
Given: channel is the structure passed as the final parameter  
rowSpan = channel.size * channel.rowLength +  
           modulo( channel.size * channel.rowLength, channel.alignment )  
address of (x,y) = channel.address + y * rowSpan + x * channel.size
```

A host may call pcFrameResultChannel one or more times provided the context properties:

- PC_OUTPUT_WIDTH
- PC_OUTPUT_HEIGHT

have been set to non-zero values. Each call may request the same or a different channel. Setting either PC_OUTPUT_WIDTH or PC_OUTPUT_HEIGHT to 0 indicates that the host will not request a result.

The arrays retrieved by pcFrameResultChannel are managed by the PC library. The calling program may not modify them and should extract any information from the arrays before the next call to pcFrameBegin.

The context property PC_RETAIN_OUTPUT requires the PC library not reuse the arrays for an additional number of frames specified by the property. This extends the period of time the calling program has to retrieve the information. For example, PC_RETAIN_OUTPUT set to 2 supports double buffering the output. The application can be compositing frame n+1 while it is still retrieving the pixels for frame n.

3.4.6 pcFrameResultQuery – test if a frame result is available

Synopsis

```
PCerr pcFrameResultQuery
(PCcontext ctx, PCid frameId,
 PCuint xOffset, PCuint yOffset, PCuint width, PCuint height)
```

Parameters

ctx	the context this frame belongs to
frameId	the unique id generated for this frame by pcFrameBegin
xOffset	the x offset in pixels from the origin of the output
yOffset	the y offset in pixels from the origin of the output
width	is the width of the result in pixels
height	is the height of the result in pixels

Return Values

This function returns PC_NO_ERROR on successful completion. *<td:return values>*

Description

The pcFrameResultChannel function will block until the selected result is complete. If an application needs to check if a result is available without blocking, it can call this function. If a subsequent call to pcFrameResultChannel will return without blocking, this function returns PC_RESULT_READY. If pcFrameResultChannel would block, this function returns PC_RESULT_BUSY.

3.5 Miscellaneous Functions

Function	Purpose
pcGetProcAddress	get the address of an extension function
pcGetErrorString	get a text description of a PCerr
pcQueryExtension	query if an extension is implemented

3.5.1 pcGetProcAddress – get the address of an extension function

Synopsis

```
PCvoidp pcGetProcAddress(PCstring funcName)
```

Parameters

funcName	the name of an extension entry point
----------	--------------------------------------

Return Values

NULL if the function is not implemented in this version of the library, otherwise the address of the function.

Description

This function is used to return a pointer to a function that is defined by an extension to the PC library. If the entry point does not exist in the implementation, this function returns NULL.

Applications must still make a runtime check for the existence of the extension before trying to use the returned function pointer. In the case where the entry point is provided by a library other than PC, `pcGetProcAddress` might return a non NULL value, however, the extension would not be defined.

3.5.2 `pcGetErrorString` – get a text description of a `PCerr`

Synopsis

```
PCstring pcGetErrorString(PCerr error)
```

Parameters

error	a PC error code
-------	-----------------

Return Values

A string describing the PC error code passed as the first parameter. If the parameter `error` is not a PC error code, "Unknown error" is returned. The program must not free the storage holding the string.

Description

This function is used to convert an error code returned by a PC library function into a string that describes the error code.

3.5.3 `pcQueryExtension` – query if an extension is present

Synopsis

```
PCint pcQueryExtension(PCstring extensionName)
```

Parameters

extensionName	the name of an extension entry point
---------------	--------------------------------------

Return Values

True (non-zero) if the extension named by the parameter `extensionName` is present in the implementation; otherwise false (0).

Description

This function is used to test if an extension is available in the current PC library implementation. As explained in 4, each extension to the PC library has a name. If the parameter passed is the name of an extension that is implemented by the library, the function will return true. If the extension is not supported or is not a valid extension name, the function will return false.

4 Versions of the Library

The definition of PC will change through time. The extensions section explains a means of introducing new features that may eventually be adopted as standard features. PC provides a number of properties that allow a caller to:

- Understand the version of this spec the implementation conforms to
- Any version attached by the vendor to the implementation to reflect different releases

The system property PC_VERSION captures this information as a string using the following format:

```
<pc-spec-version><space><vendor-version>
```

Where :

```
<pc-spec-version> : PC_SPEC_MAJOR.PC_SPEC_MINOR[.<edit>]
```

and is one of the version numbers cited in the revision history of this document

```
<vendor-version> : PC_VENDOR<space><vendor-specific-version>
```

For example:

1.1.005 Hewlett Packard 2.1.342

Implements version 1.1 of spec, edit 005

Implemented by Hewlett Packard as part of their 2.1.342 release

5 Extensions

PC allows for individual implementations of the library and the data transport layer to provide extensions beyond the standard interfaces. If the extensions require a new function, the pcGetProcAddress function can be called to get a pointer to the additional functions. Other extensions may only extend the list of PCenum values.

Before one can use an extension, two tests need to be made.

1. check for the existence of the extension in the header files at compile time
2. check for the existence of the extension using the function pcQueryExtension at runtime

Each extension has a name. This name will be defined as a C #define if the extension is defined in the header files.

```
#ifdef <extension-name>
```

If the #define exists, then the code can be compiled and checked for errors.

For the runtime check, call the function pcQueryExtension passing the name of the extension (the C #define) as a string parameter. The function will return true (non-zero) if the extension is implemented in the current implementation of the library.

To distinguish extensions from core PC features, the following naming conventions are used:

- A unique extension name of the form *PC_<type>_<name>* is associated with each extension. If the extension is supported by more than one vendor, <type> is EXT. If the extension is supported by a single vendor, <type> should designate that vendor. For example:
 - PC_EXT_io_count is an extension supported by more than one vendor
 - PC_HP_frame_output is an extension supported by the vendor HP

- All functions defined by an extension will have names of the form *<function><type>*. If the extension is supported by more than one vendor, *<type>* is EXT. If the extension is supported by a single vendor, *<type>* should designate that vendor. For example:
 - pcFrameGLAddFrameletEXT is an extension function supported by more than one vendor
 - pcFrameWaitOutputHP is an extension function supported by the vendor HP
- All PCenum values defined by the extension will have names of the form *<name>_<type>*. If the extension is supported by more than one vendor, *<type>* is EXT. If the extension is supported by a single vendor, *<type>* should designate that vendor. For example:
 - PC_WRITE_COUNT_EXT is an extension PCenum supported by more than one vendor
 - PC_WRITE_COUNT_HP is an extension PCenum supported by the vendor HP

EXT extensions can be promoted to the required core features in later versions of the PC library. When this occurs, the extension specifications are merged into the core part of this specification. The functions and PCenum values that are promoted will have the EXT or _EXT suffix removed.

PC implementations of these later versions should continue to export the extension name and function name and PCenum names with the EXT or _EXT suffix as a transition aid.

Currently Defined Extensions

5.1 PC_EXT_io_count – measure amount of network traffic

This extension returns statistics on the additional network traffic added by the PC library.

Extension Name	PC_EXT_io_count
Additional Properties	PC_WRITE_COUNT_EXT PC_READ_COUNT_EXT
Additional Functions	none

5.1.1 Additional Properties

The extension adds the following properties to each context.

Property	Type				Meaning
	Host Specific		On Create	Can be Changed	
PC_WRITE_COUNT_EXT	yes	integer	no	no	This property returns the number of bytes written by the local host to all other hosts since the last retrieval of this property. Retrieving the property resets the property to 0. Although the property is local and applies to a host, the property can only be retrieved for the local host.
PC_READ_COUNT_EXT	yes	integer	no	no	This property returns the number of bytes read by the local host from all other hosts since the last retrieval of this property. Retrieving the property resets the property to 0. Although the property is local and applies to a host, the property can only be retrieved for the local host.

5.2 PC_EXT_cur_gfx_ctx

This extension supports the library extracting the pixels for a framelet directly from a graphics card. Using this extension, the program calls `pcFrameAddGLFramelet` as a replacement for `pcFrameAddFramelet`. The extension function will read pixels from the current OpenGL context to fill the framelet.

Extension Name	PC_EXT_cur_gfx_ctx
Additional Properties	none
Additional Functions	pcFrameAddGLFrameletEXT

5.2.1 Additional Properties

none

5.2.2 pcFrameAddGLFrameletEXT – add current pixels on graphics card

Synopsis

```
PCerr pcFrameAddGLFrameletEXT
    (PCcontext ctx, PCid frameid, PCint xOffset, PCint yOffset)
```

Parameters

ctx	the context this frame belongs to
frameid	the unique id generated for this frame by <code>pcFrameBegin</code>
xOffset	horizontal offset in the extracted image on the graphics card to use as the origin
yOffset	vertical offset in the extracted image on the graphics card to use as the origin

Return Values

This function returns `PC_NO_ERROR` on successful completion. *<td:return values>*

Description

This function, like `pcFrameAddFramelet`, supplies the pixels for a framelet. The pixels are obtained by issuing a `glReadPixels(xOffset, yOffset, frameletWidth, frameletHeight,...)` from the current OpenGL graphic context.

The current OpenGL context must be compatible the `PIXEL_COMPOSITING_TYPE`. For example, if the `PIXEL_COMPOSITING_TYPE` is `PC_COMP_DEPTH`, the OpenGL context must include depth information for the pixels.

5.3 PC_HP_frame_output

This extension supports reducing the latency of extracting pixels and depth data from an output.

Extension Name	PC_HP_frame_output
Additional Properties	new type PCresult
Additional Functions	pcFrameWaitOutputHP

	pcResultGetChannelHP pcResultDestroyHP
--	---

5.3.1 Additional Properties

PC type	properties of the type
PCresult	an opaque value representing an instance of a PC result. A PCresult is needed in pcFrameWaitOutput to track a result returned by this function. The concept exists in the core part of the spec but there are no functions that specifically manipulate a result.

5.3.2 pcFrameWaitOutputHP – retrieve available output

Synopsis

```
PCerr pcFrameWaitOutputHP
(PCcontext ctx, PCid frameId,
 PCresult result);
```

Parameters

ctx	the context this frame belongs to
frameId	the unique id generated for this frame by pcFrameBegin
result	a PCresult that will be initialized by this function

Return Values

This function returns PC_NO_ERROR on successful completion. *<td>return values</td>*

Description

This function retrieves output pixels of the frame and corresponds to step **2.c.v.1** in section **2.11**.

This function uses a different approach to retrieving output than pcFrameResultChannel. It checks to see if any new pixels have arrived in the output since the last call to this function. If a rectangular area such as one or more complete rows has arrived, it creates a result describing this area and returns the rectangular area using the parameter result. If the library cannot construct a rectangular area, the function blocks until it can. If the entire output has already been returned, the function returns immediately with the return value PC_ALL_OUTPUT_RETURNED.

How an implementation might segment the output into rectangular areas is determined by the implementation. Although rows of the output (i.e. scanlines) were cited in the example above, the implementation is free to return results of any and varied sized.

Results returned by pcFrameResultChannel do not effect what pcFrameWaitOutputHP tracks as having been returned. pcFrameWaitOutputHP will not return the status PC_ALL_OUTPUT_RETURNED until a series of one or more pcFrameWaitOutputHP calls has returned results covering the entire output for the calling host.

The final parameter (result) is a value initialized by the library on a successful return from this function for accessing the information associated with the result. Using this value as an argument to the function pcResultGetChannel in section 5.3.3, a program can access the channels (color or depth) of the result.

This call must be made following the host's call to pcFrameEnd for the frame specified by the parameter frameId.

The extension context property PC_RETAIN_OUTPUT requires the PC library not reuse the arrays for an additional number of frames specified by the property. See section 3.4.5 for further details.

5.3.3 pcResultGetChannelHP – get a channel of a result

Synopsis

```
PCerr pcResultGetChannelHP  
(PCresult result, PCuint channelType, PCchannel* channel);
```

Parameters

result	a PCresult
channelType	the channel to return
channel	a PCchannel that will be initialized for accessing the channel specified by this function

Return Values

This function returns PC_NO_ERROR on successful completion. *<td:return values>*

Description

This function return a channel that describes one of the 2-dimensional arrays associated with the parameter result. The actual channel returned is specified by the parameter channelType.

The functions in this extension are structured differently than those in the core spec due to the way pcFrameWaitOutputHP works. Because two calls to pcFrameWaitOutputHP return different results, you cannot call it repetitively to get different channels of the same result as you can with pcFrameResultChannel.

5.3.4 pcResultDestroyHP – destroy a PCresult

Synopsis

```
PCerr pcResultDestroyHP(PCresult result)
```

Parameters

result	a PCresult
--------	------------

Return Values

This function returns PC_NO_ERROR on successful completion. *<td:return values>*

Description

As described in 3.4.5, a result will continue to return valid properties for the number of frames specified by the context property PC_RETAIN_OUTPUT. Calling this function resets a PCresult so it no longer references a result.

The first parameter specifies the result that you no longer wish to retain.