BPatch Interface Reference

Version 0.1

July 31, 1998

John Robb

email:johnrobb@us.ibm.com

IBM Corporation

RS/6000 Development

522 South Road, MS P-963

Poughkeepsie, New York 12601

Draft Document

Table of Contents

About This Document:

This subroutine reference contains a description of the BPatch interface which is required for the Dynamic Probe Classs Library (DPCL).

The functions and classes may have the word proposed next to them. This means that this is not yet part of the DyninstAPI.

Some of the proposed functions are similar to the existing BPatch ones. If the existing BPatch interface is not listed in this book, then it is not required by DPCL.

Restrictions:

Inserting probes in code segments loaded using the load system call is not supported at this time.

Comments:

1. BPatch_vector has been changed to STL vector.
2. Some functions will call the error callback routine. For each of these functions, the document should indicate that an error callback routine bay be triggered. Do the parms passed to the error callback need further definition? What if there is no callback registered?
3. In two places BPatch_sourceObj is exposed as a return value: getSourceBlock in BPatch_basicBlock and getObjParent in BPatch_sourceObj
4. If the BPatch_sourceObj class is not exposed in the interface, what header will the user include to get access to the member functions. Should be have two flavors of BPatch_sourceObj.h; one for the interface and one internal one?

Requirements:

These are requirements that may drive changes and additions to the BPatch interface.

1. Elimination of the pre-link limitation
2. Source granularity down to loop level
3. Access to variables and their types
4. Support for 64 bit applications
5. Support for threaded applications
6. Ability to insert probes in objects loaded with the load system call. This involves adding an interface to refresh the module list after a load or unload has been completed. Will refreshing the source structure after a load/unload be available just at BPatch level or at DPCL also?

dyninstAPI_init is called from BPatch constructor and returns success/failure -- should pass this on somehow. 1
Need to define peer BPatch enum of??? containg the following 27
Need to define peer BPatch enum of??? containg the following 49
Should be BPatch_variableExpr 68
BPatch_sourceObj.h - Is it ok to expose this header file to provide access to the declarations of its member functions? 88
Is the following correct? 94
Is the following correct? 95
Is the following correct? 96
Is the following correct? 97
Need to keep BPatch_sourceObj from being exposed. 99
Do we need a showCaughtSignals member? 107
What is the behavior of terminate (when True) if dumping core is not successful? Should termination not start until after the dump completes? If so we should say that upon failure, the dump did not occur and the process was not terminated. 112
should this have a return value to indicate that the BPatch_variableExpr had not been malloc'ed using BPatch_thread::malloc? 114
Do we need a showIgnoredSignals member? 117
Should return NULL on failure, but the function which it calls, inferiorMalloc, calls exit rather than returning an error, so this is not currently possible in the current implimentation. 123
Should return NULL on failure, but the function which it calls, inferiorMalloc, calls exit rather than returning an error, so this is not currently possible in the current implimentation. 124
What will an error return? 136
What is equal? name, description, size? 137
What about enumerated types? 138
What about arrays? 138
Does this eliminate the need for isStructure? 139
These constructors are not exposed currently in BPatch. To expose them, do we need to change the interface from passing in a process class to passing in a BPatch_thread? 141
What will this return for arrays? 145

1.0 Class BPatch

This is the main BPatch class. It contains information on all processes being traced. Only one of these objects is allowed per tracing program.

1.1 Constructors / Destructors

1.1.1 BPatch

Synopsis

```
#include <BPatch.h>
BPatch( )
```

Description

Constructor for BPatch. Performs one-time initialization needed by the library.

Note: dyninstAPI_init is called from BPatch constructor and returns success/failure -- should pass this on somehow.

1.1.2 ~BPatch

Synopsis

```
#include <BPatch.h>
~BPatch( )
```

Description

Destructor for BPatch. Free allocated memory.

1.2 attachProcess - proposed

Synopsis

```
#include <BPatch.h>
BPatch_thread *attachProcess(int pid )
```

Parameters

pid	The id of the process to attach to.

Description

Attach to a running process and return a BPatch_thread representing it.

Return value

On Success, return a BPatch_thread.
Returns NULL upon failure.

See Also

`createProcess`

1.3 createProcess

Synopsis

```
#include <BPatch.h>
BPatch_thread *createProcess(
     char *path,
     char *argv[],
     char *envp[] = NULL);
```

Parameters

path	The pathname of the executable for the new process.
argv	A list of the arguments for the new process, terminated by a NULL
envp	A list of values that make up the environment for the new process, terminated by a NULL. If envp is NULL, the new process will inherit the environment of the parent.

Description

Create a process and return a BPatch_thread representing it.

Return value

Upon Success, return a BPatch_thread.

Returns NULL upon failure.

See Also

attachProcess

1.4 getLastErrorCode

Synopsis

```
#include <BPatch.h>
short getLastErrorCode( void )
```

Description

Return the last error which occurred.

1.5 registerErrorCallback

Synopsis

```
#include <BPatch.h>

BPatchErrorCallback registerErrorCallback(
     BPatchErrorCallback func )
```

Parameters

func	the function to be called

Description

This function registers the error callback function with the Bpatch class. The error callback is explicitly registered (rather than using a pure a virtual function) so that Bpatch users can change the error callback during program execution (i.e., one error callback before a GUI is initialized, and a different one after).

Callback Data

```
typedef enum BPatchErrorLevel {
     BPatchFatal, BPatchSerious, BPatchWarning, BPatchInfo };

typedef void ( *BPatchErrorCallback ) (
     BPatchErrorLevel severity,
     int number,
     char **params );
```

This is the prototype for the error callback function. The severity indicates how important the error is (from fatal to information/status). The number is a unique number that identifies this error message. Params are the parameters that describe the detail about an error, For example, the process id where the error occurred. The number and meaning of params depends on the error, However, for a single error number, the number of parameters returned will always be the same.

Return value

Returns the address of the previously registered error callback function.

1.6 registerExitCallback - proposed

Synopsis

```
#include <BPatch.h>
BPatchThreadEventCallback registerExitCallback(
     BPatchThreadEventCallback func )
```

Parameters

func	the function to be called

Description

Registers a function that is to be called by the library when a thread terminates.

Callback Data

```
typedef void ( *BPatchThreadEventCallback ) (
     BPatch_thread *thread );
```

This is the prototype for most callback functions associated with events that occur in a thread. The thread parameter is the thread that the event has occurred in.

Return value

Returns the address of the previously registered exit callback function.

1.7 registerDynamicLinkCallback - proposed

Synopsis

```
#include <BPatch.h>
BPatchThreadEventCallback registerDynamicLinkCallback(
     BPatchThreadEventCallback func )
```

Parameters

func	the function to be called

Description

Registers a function that is to be called by the library after a load or unload call in the application has completed.

Callback Data

```
typedef void ( *BPatchThreadEventCallback ) (
     BPatch_thread *thread );
```

This is the prototype for most callback functions associated with events that occur in a thread. The thread parameter is the thread that the event has occurred in.

Return value

Returns the address of the previously registered dynamic link callback function.

2.0 Class BPatch_basicBlock - Proposed

This class represents a compiler basic block.

Following is the class hierarchy for the IBM implementation:

⇒ BPatch_basicBlock
  ⇒ BPatch_basicBlockLoop

2.1 getSources - proposed

Synopsis

```
#include <BPatch_basicBlock.h>

vector<BPatch_basicBlock> *getSources( void )
```

Description

Return the predecessors of the basic block.

Return value

Return a vector of all predecessors of the basic block. If the lookup fails to locate any predecessors of the basic block, a list with zero elements is returned.

See Also

`getTargets`

2.2 getTargets - proposed

Synopsis

```
#include <BPatch_basicBlock.h>

vector<BPatch_basicBlock> *getTargets( void )
```

Description

Return all successors of the basic block.

Return value

Return a vector of all successors of the basic block. If the lookup fails to locate any successors of the basic block, a list with zero elements is returned.

See Also

`getSources`

2.3 getSourceBlock - proposed

Synopsis

```
#include <BPatch_basicBlock.h>
BPatch_sourceBlock *getSourceBlock( void )
```

Description

Return the source block containing the basic block

3.0 Class BPatch_flowGraph - proposed

The flow graph is made up of compiler basic blocks. The BPatch_flowGraph is an attribute of a FunctionObj.

3.1 getEntryBasicBlock - proposed

Synopsis

```
#include <BPatch_flowGraph.h>

vector<BPatch_basicBlock> *getEntryBasicBlock( void )
```

Description

Return all basic blocks which do not have a predecessor (either we could not find one or it is the first basic block in the procedure).

Return value

Return a vector of all basic blocks which do not have a predecessor. If the lookup fails to locate any, a list with zero elements is returned.

3.2 getExitBasicBlock - proposed

Synopsis

```
#include <BPatch_flowGraph.h>
vector<BPatch_basicBlock> *getExitBasicBlock( void )
```

Description

Return all basic blocks which do not have a successor (either last blocks in procedure or the successor could not be determined).

Return value

Return a vector of all basic blocks which do not have a successor. If the lookup fails to locate any, a list with zero elements is returned.

3.3 getFunction - proposed

Synopsis

```
#include <BPatch_flowGraph.h>
Bpatch_function *getFunction( void )
```

Description

Returns the procedure containing this control flow graph (CFG).

Return value

Returns a Bpatch_function or NULL if the containing function cannot be determined.

3.4 getLoops - proposed

Synopsis

```
#include <BPatch_flowGraph.h>
vector<BPatch_basicBlockLoop> *getLoops( void )
```

Description

Return all outermost loops in the procedure.

Return value

Return a vector of all outermost loops in the procedure. If the lookup fails to locate any, a list with zero elements is returned.

4.0 Class BPatch_flowGraphLoop - proposed

This is a subclass of the BPatch_basicBlock class which represents a loop.

4.1 getBackEdges - proposed

Synopsis

```
#include <BPatch_basicBlockLoop.h>
vector<BPatch_basicBlock> *getBackEdges( void )
```

Description

Returns the basic blocks which are the sources of the back edges defining the loop.

Return value

Return a vector of the back edges. If the lookup fails to locate any back edges, a list with zero elements is returned.

4.2 getContainedLoops - proposed

Synopsis

```
#include <BPatch_basicBlockLoop.h>
vector<BPatch_basicBlock> *getContainedLoops( void )
```

Description

Get next level of contained loops.

Return value

Return a vector of all the contained loops. If the lookup fails to locate any contained loops, a list with zero elements is returned.

4.3 getLoopBasicBlocks - proposed

Synopsis

```
#include <BPatch_basicBlockLoop.h>
vector<BPatch_basicBlock> *getLoopBasicBlocks( void )
```

Description

Return all basic blocks belonging to the loop.

Return value

Return a vector of the loops basic blocks. If the lookup fails to locate any basic blocks belonging to the loop, a list with zero elements is returned.

4.4 getLoopHead - proposed

Synopsis

```
#include <BPatch_basicBlockLoop.h>
BPatch_basicBlock *getLoopHead( void )
```

Description

Return the basic block object of the loop head

4.5 getLoopIterators - proposed

Synopsis

```
#include <BPatch_basicBlockLoop.h>
vector<BPatch_variableExpr> *getLoopIterators( void )
```

Description

Return loop iteration.

Return value

Return a vector of the loop iterators. If the lookup fails to locate any loop iterators, a list with zero elements is returned.

5.0 Class BPatch_function

An object of this class represents a function in the application. A BPatch_image or BPatch_module object (see descriptions) can be used to retrieve a BPatch_function object representing a given function.

5.1 functionArgs - proposed

Synopsis

```
#include <BPatch_function.h>

vector<BPatch_variableExpr *> *functionArgs( void )
```

Description

Get a list of the functions arguments in order to have access to the input values to the function.

Return value

The return value contains a vector of BPatch_variableExpr's representing the functions arguments. If functionArgs fails to locate any function arguments, a list with zero elements is returned.

5.2 getFlowGraph - proposed

Synopsis

```
#include <BPatch_function.h>

BPatch_flowGraph *getFlowGraph( void )
```

Description

Get the Control Flow Graph of the function.

Return value

Return the Control Flow Graph, or NULL if none is available.

5.3 getLoops - proposed

Synopsis

```
#include <BPatch_function.h>
vector<BPatch_basicBlockLoop> *getLoops( void )
```

Description

Return all outermost loops in the function.

Return value

Return a vector of all outermost loops in the function. If the lookup fails to locate any, a list with zero elements is returned.

5.4 getMangledName - proposed

Synopsis

```
#include <BPatch_function.h>

const char *getMangledName( void )
```

Description

Get the mangled name for the function. If there is no name mangling done for the function, the name by which the linker knows the function will be returned.

Return value

Returns the mangled name of the function, or NULL is the name is not available.

5.5 getName

Synopsis

```
#include <BPatch_function.h>
const char *getName( void )
```

Description

Gets the name of the function.

Return value

Returns name of the function, or NULL is the name is not available.

5.6 getSharedLibType - proposed

Note: Need to define peer BPatch enum of??? containg the following

shared_public	public shared object
shared_private	private shared object
non-shared	not a shared object
unknown	unknown type

Synopsis

```
#include <BPatch_function.h>
BPatch_??? getSharedLibType( void )
```

Description

If the function is part of a shared library, get the type of the shared library which contains the function.

Return value

Return a BPatch_??? representing the containing shared library type.

5.7 getSourceBlocks - proposed

Synopsis

```
#include <BPatch_function.h>
vector<BPatch_sourceBlock *> *getSourceBlocks( void )
```

Description

Get a list of the source blocks contained within the function.

Return value

The return value contains a vector of source blocks contained within the function. If the lookup fails to locate any, a list with zero elements is returned.

5.8 moduleName - proposed

Synopsis

```
#include <BPatch_function.h>
const char *moduleName( void )
```

Description

Get the name of the source file that defines this function. Depending on whether the program was compiled for debugging or the symbol table stripped, this information may not be available.

Return value

The return value contains a pointer to a string containing the source file name.

non-NULL	contains the source file name
NULL	failure, source file name is unavailable

5.9 returnType - proposed

Synopsis

```
#include <BPatch_function.h>

BPatch_type *returnType( void )
```

Description

Get the type that the function returns.

Return value

The return value contains a BPatch_type which represents the type the function returns.

non-NULL	pointer to a BPatch_type
NULL	unable to get return type

5.10 sharedLibraryName - proposed

Synopsis

```
#include <BPatch_function.h>
const char *sharedLibraryName( void )
```

Description

Return the name of the shared library that contains the function. If the function is not part of a shared library, a NULL will be returned. The path will be returned if it is available.

Return value

The return value contains a char* representing the shared library name.

non-NULL	contains the shared library name
NULL	not part of a shared library

6.0 Class BPatch_image

Class BPatch_image defines a program image (the executable associated with a thread) which includes information from the loaded a.out, any objects that are loaded at exec time or later and information from the file which the objects were loaded from.

The only way to get a handle to a Bpatch_Image is via the Bpatch_thread member function getImage().

The program structure contains two views:

1) Source view - Gives the program source structure. This is an hierarcial tree consisting of modules, functions, blocks and variables

2) Compiler view - This is a directed graph which will be refered to as the control flow graph (CFG) with additional information relevant to the way the program executed (like Loops)

The image components may also contain points of interest in the code (instrumentation points)

6.1 findLinePoint - proposed

Synopsis

```
#include <BPatch_image.h>
const BPatch_point *findLinePoint(
          const char *filename,
          int line)
```

Parameters

filename	name of the file
line	line number

Description

Return the last instrumentation point in a given file and before a line number.

Return value

non-NULL	contains a BPatch_point* representing the line number
NULL	no such point exists

6.2 findProcedurePoint

Synopsis

```
#include <BPatch_image.h>
#include <BPatch_point.h>
vector<BPatch_point*> *findProcedurePoint(
          const char *name,
          const BPatch_procedureLocation loc )
```

Parameters

name	The name of the procedure in which to look up the points.
loc	The criteria which determines the points within the procedure to return.

Description

Return instrumentation points in a procedure according to criteria specified by the loc parameter.

Return value

Return a vector of specified points. If the lookup fails to locate any, a vector with zero elements is returned.

6.3 findType

Synopsis

```
#include <BPatch_image.h>

BPatch_type *findType(const char *name )
```

Parameters

name	the name of the type to look up

Description

Lookup and return a handle to the named type. The handle can be used as an argument to malloc in order to create new variables of the corresponding type.

Return value

non-NULL	contains a BPatch_type* representing the named type
NULL	no such type exists

6.4 getModules

Synopsis

```
#include <BPatch_image.h>
vector<BPatch_module *> *getModules( void )
```

Description

Returns a vector of all modules in the image

Return value

Return a vector of all modules in the image. If the lookup fails to locate any, a list with zero elements is returned.

6.5 getProcedures

Synopsis

```
#include <BPatch_image.h>
const vector<BPatch_function *> *getProcedures( void )
```

Description

Returns a vector of functions in the image. If the lookup fails to locate any functions, a list with zero elements is returned.

6.6 getUniqueString - Proposed

Synopsis

```
#include <BPatch_image.h>

const char *getUniqueString( void )
```

Description

Get a string which uniquely identifies the image.

A unique string will be returned, possibly containing the compile date and time which will allow determining when two images are the same and when they are different.

Return value

non-NULL	return value contains a unique string
NULL	unable to obtain a unique string

6.7 lpType - proposed

Synopsis

```
#include <BPatch_image.h>
BPatch_LpModel lpType( void )
```

Description

Find whether the image uses 32-bit or 64-bit addressing.

Return value

Returns a BPatch_LpModel , valid values are:

LP32	image is a 32-bit application
LP64	image is a 64-bit application
UNKNOWN_LP	addressing mode unknown

6.8 programName - proposed

Synopsis

```
#include <BPatch_image.h>
const char *programName( void )
```

Description

Get the name of the program as it was passed to exec.

If the string passed to exec contained an absolute or relative pathname, it will not be returned prepended to the program name.

Return value

non-NULL	returns the name of the program
NULL	unable to get the name of the program

6.9 getLoadedFilenames - proposed

Synopsis

```
#include <BPatch_image.h>

vector<char *> *getLoadedFilenames( void )
```

Description

Get a vector of filenames of any objects that are used by the image. This includes objects such as shared libraries loaded at exec time and objects loaded using the load system call.

Return value

Return a vector of the loaded filenames in the image. If the lookup fails to locate any, a list with zero elements is returned.

7.0 Class BPatch_module

A BPatch_module represents the objects that lie between the entire BPatch image and the BPatch_function.

An object of this class represents a program module, which is part of a program's executable image. BPatch_module objects are obtained by calling the BPatch_image member function getModules().

In the simple case, the BPatch_module represents a source file which has been compiled to produce an object file. A more complex case is that of an archive of shared and non-shared objects, for example: libc.a. The shared library libc.a is in archive format and contains shared objects such as shr.o which in turn contain multiple compiled source objects. It is likely that libc.a also contains other simple compiled source objects. There will be a BPatch_module for libc.a itself, shr.o and its peer shared objects, the compiled source objects which are members of the shared objects and any other simple compiled source objects contained in the archive libc.a.

The filename attribute will be used to distinguish if the modules are part of a different file.

7.1 Supporting Data Types

7.1.1 BPatch_binding - proposed

Synopsis

```
#include <BPatch_module.h>
enum BPatch_binding {
     BPatch_static,
     BPatch_dynamic,
     BPatch_unknownBinding
}
```

Description

This enumeration type describes the type of binding that may be associated with the module.

7.1.2 BPatch_language - proposed

Synopsis

```
#include <BPatch_module.h>
enum BPatch_language {
     BPatch_c,
     BPatch_cpp,
     BPatch_fortran,
     BPatch_fortran77,
     BPatch_fortran90,
     BPatch_assembley,
     BPatch_mixed,
     BPatch_hpf,
     BPatch_unknownLanguage
}
```

Description

This enumeration type describes the language associated with the module.

7.2 bindingType - proposed

Synopsis

```
#include <BPatch_module.h>
Bpatch_Binding bindingType( void )
```

Description

Get the binding type of the module.

Return value

The return value contains the Bpatch_Binding associated with the module.

7.3 getDataRelocFactor - proposed

Synopsis

```
#include <BPatch_module.h>
int getDataRelocFactor( void )
```

Description

Get the data relocation factor.

Return value

Returns the data relocation factor or -1 for failure.

See Also

`getTextRelocFactor, setRelocFactor`

7.4 getLanguage - proposed

Synopsis

```
#include <BPatch_module.h>

BPatch_language getLanguage( void )
```

Description

Get the source language of the module.

Return value

The return value contains a BPatch_language which represents the source language in which the module was written.

See Also

```
enum BPatch_language
```

7.5 getName

Synopsis

```
#include <BPatch_module.h>
const char *getName( void )
```

Description

Gets the name of the module.

Return value

Returns name of the module, or NULL if the name is not available.

7.6 getProcedures

Synopsis

```
#include <BPatch_module.h>
const vector<BPatch_function *> *getProcedures( void )
```

Description

Returns a vector of functions in the module. If the lookup fails to locate any functions, a list with zero elements is returned.

7.7 getSharedLibType - proposed

Note: Need to define peer BPatch enum of??? containg the following

shared_public	public shared object
shared_private	private shared object
non-shared	not a shared object
unknown	unknown type

Synopsis

```
#include <BPatch_module.h>

BPatch_??? getSharedLibType( void )
```

Description

If the module is part of a shared library, get the type of the shared library which contains the module.

Return value

Return a BPatch_??? representing the containing shared library type.

7.8 getTextRelocFactor - proposed

Synopsis

```
#include <BPatch_module.h>
int getTextRelocFactor( void )
```

Description

Get the text relocation factor.

Return value

Returns the text relocation factor or -1 for failure.

See Also

`getDataRelocFactor, setRelocFactor`

7.9 getUniqueString - proposed

Synopsis

```
#include <BPatch_module.h>

const char *getUniqueString( void )
```

Description

Get a string which uniquely identifies the module.

A unique string will be returned, possibly containing the compile date and time which will allow determining when two modules are the same and when they are different.

Return value

non-NULL	return value contains a unique string
NULL	unable to obtain a unique string

7.10 setRelocFactor - proposed

Synopsis

```
#include <BPatch_module.h>
int setRelocFactor(
          unsigned long text = 0,
          unsigned long data = 0 )
```

Description

Get the text relocation factor.

Return value

Returns the text relocation factor or -1 for failure.

See Also

`getDataRelocFactor, getTextRelocFactor`

7.11 sharedLibraryName - proposed

Synopsis

```
#include <BPatch_module.h>
const char *sharedLibraryName( void )
```

Description

Return the name of the shared library that contains the module. If the module is not part of a shared library, a NULL will be returned. The path will be returned if it is available.

Return value

The return value contains a char* representing the shared library name.

non-NULL	contains the shared library name
NULL	not part of a shared library

8.0 Class BPatch_point

An object of this class represents a location in an application's code at which the library can insert instrumentation. A BPatch_image, BPatch_module, BPatch_function, BPatch_sourceBlock or BPatch_basicBlock (see descriptions) are used to retrieve a BPatch_point representing a desired point in the application.

8.1 Supporting Data Types

8.1.1 BPatch_address - proposed

Synopsis

```
typedef BPatch_address unsigned long long;
```

Description

This typedef describes a type long enough to hold a 64 bit address.

8.1.2 BPatch_procedureLocation - proposed

Synopsis

```
#include <BPatch_point.h>
enum BPatch_procedureLocation {
    BPatch_unknownType,
    BPatch_entry,                 // entry to procedure
    BPatch_exit,                  // exit from procedure
    BPatch_subroutine,            // location of calls to
                                  //   other subroutines
    BPatch_longJump,              // long jump statements
    BPatch_basicBlockEntry,       // compiler BB entry
    BPatch_basicBlockExit,        // compiler BB exit
    BPatch_sourceBlockEntry,      // source block entry
    BPatch_sourceBlockExit,       // source block exit
    BPatch_sourceLoop,            // loop
    BPatch_sourceLoopEntry,       // entry to source loop
    BPatch_sourceLoopExit,        // exit of source loop
    BPatch_basicBlockLoopEntry,   // compiler BB loop entry
    BPatch_basicBlockLoopExit,    // compiler BB loop exit
    BPatch_statement,             // beginning of a statement
    BPatch_varInitStart,          // start var initialization
    BPatch_varInitEnd,            // end of var initilization
```

```
        BPatch_allLocations          // all of the above
    }
```

Description

This enumeration type describes the type of instrumentation point.

8.2 getAddress - proposed

Synopsis

```
#include <BPatch_point.h>
BPatch_address getAddress( void )
```

Description

Get the address of the first instruction at this point.

Return value

non-NULL	the return value contains the address of the first instruction
NULL	unable to get the address of the first instruction

8.3 getCalledFunction

Synopsis

```
#include <BPatch_point.h>

BPatch_function *getCalledFunction( void )
```

Description

Returns a BPatch_function representing the function that is called at the point. If the point is not a function call site or the target of the call cannot be determined, then this function returns NULL.

Return value

non-NULL	success, returns BPatch_function*
NULL	failure, point is not a function call site or the target of the call cannot be determined

8.4 getDisplacedInstruction - proposed

Synopsis

```
#include <BPatch_point.h>
unsigned int getDisplacedInstruction( void )
```

Description

Return the instruction to be relocated at this point.

8.5 getPointType - proposed

Synopsis

```
#include <BPatch_point.h>
const BPatch_procedureLocation getPointType( void )
```

Description

Get the type of the instrumentation point.

Return value

Return the type of the instrumentation point. BPatch_unknown is returned if the type could not be determined.

8.6 instPtLine - proposed

Synopsis

```
#include <BPatch_point.h>
unsigned int instPtLine( void )
```

Description

Get the approximate source line where the instrumentation point occurs. The source line number is offset from the beginning of the file.

Return value

non-zero	success, returns the source line number
zero	failure, unable to get the source line number

8.7 showInstPoint - proposed

Synopsis

```
#include <BPatch_point.h>
instPoint *showInstPoint( void )
```

Description

??? Need a definition for showInstPoint. If we don’t have one, lets delete this.

Return value

Returns ???.

9.0 Class BPatch_snippet

A snippet is an abstract representation of code to insert into a program. Snippets are defined by creating a new instance of the correct subclass of a snippet. For example, to create a snippet to call a function, you create a new instance of the class BPatch_funcCallExpr. Creating a snippet does not result in code being inserted into an application. Code is generated when a request is made to insert a snippet at a specific point in a program. Sub-snippets may be shared by different snippets (i.e. a handle to a snippet may be passed as an argument to create two different snippets), but whether the generated code is shared (or replicated) between two snippets is implementation dependent.

Following classes are implemented as derived classes of BPatch_snippet:

9.0.1 BPatch_snippet

Synopsis

```
#include <BPatch_snippet.h>
BPatch_snippet( void ) : ast( NULL )
```

Description

9.0.2 BPatch_snippet

Synopsis

```
#include <BPatch_snippet.h>
BPatch_snippet(
     const BPatch_snippet &src )
```

Parameters

src

Description

Copy constructor for BPatch_snippet.

9.0.3 BPatch_snippet &operator

Synopsis

```
#include <BPatch_snippet.h>
BPatch_snippet &operator=( const BPatch_snippet & )
```

Parameters

Description

9.0.4 ~BPatch_snippet

Synopsis

```
#include <BPatch_snippet.h>
virtual ~BPatch_snippet( )
```

Description

Destructor for BPatch_snippet. Deallocates memory allocated by the snippet.

9.0.5 getCost - proposed

Synopsis

```
#include <BPatch_snippet.h>
float getCost( void )
```

Description

Return an estimate of the number of seconds it would take to execute the snippet. The problems with accurately estimating the cost of executing code are numerous and out of the scope of this document. But, it is important to realize that the returned cost value is (at best) an estimate.

Return value

non-NULL	returns an estimate of the number of seconds it would take to execute the snippet
NULL	unable to get an estimate

9.0.6 getType

Synopsis

```
#include <BPatch_snippet.h>
const BPatch_type *getType( )
```

Description

Get the type of the snippet.

Return value

non-NULL	returns the type of the snippet
NULL	unable to get the type of the snippet

9.1 Class BPatch_arithExpr : public BPatch_snippet

This is implemented as a derived class of BPatch_snippet.

9.1.1 BPatch_arithExpr

Synopsis

```
#include <BPatch_snippet.h>
BPatch_arithExpr(
     BPatch_unOp op,
     const BPatch_snippet &Operand )
```

Parameters

op	the desired unary operation
Operand	the operand of the operation

Description

Define a snippet consisting of a unary operator. The available unary operators are BPatch_negate, and BPatch_address. BPatch_negate takes an integer snippet and returns the negation of the snippet. BPatch_address takes a variable reference snippet and returns a pointer to it. This is equivalent to the C operator (&) and is useful for call-by-reference parameters.

9.1.2 BPatch_arithExpr - proposed

Synopsis

```
#include <BPatch_snippet.h>
BPatch_arithExpr(
     BPatch_binOp op,
     const BPatch_snippet &lOperand,
     const BPatch_snippet &rOperand )
```

Parameters

op	the desired binary operation
lOperand	the left operand for the operation
rOperand	the right operand for the operation

Description

Construct a snippet representing a binary arithmetic operation.

The available binary operators are:

Operator	Description
BPatch_assign	assign the value of rOperand to lOperand
BPatch_plus	add lOperand and rOperand
BPatch_minus	subtract rOperand from lOperand
BPatch_divide	divide rOperand by lOperand
BPatch_times	multiply rOperand by lOperand
BPatch_mod	compute the remainder of dividing rOperand into lOperand Not yet implemented.
BPatch_ref	array reference of the form lOperand[rOperand]

Note: Should be BPatch_variableExpr

BPatch_seq	define a sequence of two expressions (similar to comma in C)
BPatch_min	return the smaller of two operands
BPatch_max	return the larger of two operands
BPatch_bit_and	return the bitwise and of the two operands
BPatch_bir_or	return the bitwise or of the two operands
BPatch_bit_xor	return the bitwise exclusive or of the two operands
BPatch_left_shift	bitwise shift to the left the lOperand rOperand times
BPatch_right_shift	bitwise shift to the right the lOperand rOperand times

9.2 Class BPatch_boolExpr : public BPatch_snippet

This is implemented as a derived class of BPatch_snippet.

9.2.1 BPatch_boolExpr - proposed

Synopsis

```
#include <BPatch_snippet.h>
BPatch_boolExpr(
     BPatch_unRelOp op,
     const BPatch_snippet &Operand )
```

Parameters

op	the operator for the boolean expression
Operand	the operand of the boolean operation

Description

Constructs a snippet representing a unary boolean expression.

The available operators are:

Operator	Function
Bpatch_not	Return !Operand

The type of the returned snippet is boolean, and the operands are type checked.

9.2.2 BPatch_boolExpr

Synopsis

```
#include <BPatch_snippet.h>
BPatch_boolExpr(
     BPatch_relOp op,
     const BPatch_snippet&lOperand,
     const BPatch_snippet &rOperand )
```

Parameters

op	the operator for the boolean expression
lOperand	the left operator
rOperand	the right operator

Description

Constructs a snippet representing a binary boolean expression.

The available operators are:

Operator	Function
Bpatch_lt	Return lOperand < rOperand
Bpatch_eq	Return lOperand == rOperand
Bpatch_gt	Return lOperand > rOperand
Bpatch_le	Return lOperand <= rOperand
Bpatch_ne	Return lOperand != rOperand
Bpatch_ge	Return lOperand >= rOperand
Bpatch_and	Return lOperand && rOperand (Boolean and)
Bpatch_or	Return lOperand || rOperand (Boolean or)

The type of the returned snippet is boolean, and the operands are type checked.

9.3 Class BPatch_constExpr : public BPatch_snippet

This is implemented as a derived class of BPatch_snippet.

9.3.1 BPatch_constExpr

Synopsis

```
#include <BPatch_snippet.h>
BPatch_constExpr( int value )
```

Parameters

value the desired value

Description

Construct a constant snippet of the type integer. This constructor creates a constant containing the value specified by the parameter, which is copied into the application's address space. TheBPatch_constExpr that is created refers to the location where the integer constant exists.

9.3.2 BPatch_constExpr

Synopsis

```
#include <BPatch_snippet.h>
BPatch_constExpr( float value )
```

Parameters

value the desired value

Description

Construct a constant snippet of the type float. This constructor creates a constant containing the value specified by the parameter, which is copied into the application's address space. The BPatch_constExpr that is created refers to the location where the float constant exists.

9.3.3 BPatch_constExpr

Synopsis

```
#include <BPatch_snippet.h>
BPatch_constExpr(
     const char *value )
```

Parameters

value the desired constant string

Description

Construct a constant snippet of the type char*. This constructor creates a constant string; the null-terminated string beginning at the location pointed to by the parameter is copied into the application's address space. The BPatch_constExpr that is created refers to the location to which the string was copied.

9.4 Class BPatch_funcCallExpr : public BPatch_snippet

This is implemented as a derived class of BPatch_snippet.

9.4.1 BPatch_funcCallExpr

Synopsis

```
#include <BPatch_snippet.h>
BPatch_funcCallExpr(
    const BPatch_function& func,
    const vector<BPatch_snippet *> &args )
```

Parameters

func	the function to be called
args	a vector of the arguments to be passed to the function

Description

Construct a call to a function, the passed function must be valid for the current code region. Args is a list of arguments to pass to the function. If type checking is enabled, the types of the passed arguments are checked against the function to be called (Availability of type checking depends on the source language of the application and program being compiled for debugging).

9.5 Class BPatch_ifExpr : public BPatch_snippet

This is implemented as a derived class of BPatch_snippet.

9.5.1 BPatch_ifExpr

Synopsis

```
#include <BPatch_snippet.h>

BPatch_ifExpr(

     const BPatch_boolExpr &conditional,

     const BPatch_snippet &tClause )
```

Parameters

conditional	contains a boolean expression used to determine whether to execute the snippet clause
tClause	snippet to execute if the boolean clause evaluates to true

Description

This constructor creates an if statement. The first argument, conditional , should be a Boolean expression that is will be evaluated to decide if the clause should be executed. The second argument, tClause , is the snippet to execute if the conditional evaluates to true.

9.5.2 BPatch_ifExpr

Synopsis

```
#include <BPatch_snippet.h>

BPatch_ifExpr(

     const BPatch_boolExpr &conditional,

     const BPatch_snippet &tClause,

     const BPatch_snippet &fClause )
```

Parameters

conditional	contains a boolean expression used to determine which of the snippet clauses to execute
tClause	snippet to execute if the boolean clause evaluates to true
fClause	snippet to execute if the boolean clause evaluates to false

Description

Constructs a snippet representing a conditional expression with an else clause. The first argument, conditional , should be a Boolean expression that is will be evaluated to decide which clause should be executed. The second argument, tClause , is the snippet to execute if the conditional evaluates to true. The third argument, fClause , is the snippet to execute if the conditional evaluates to false. Else-if statements, can be constructed by making the fClause of an if statement another if statement.

9.6 Class BPatch_nullExpr : public BPatch_snippet

This is implemented as a derived class of BPatch_snippet.

9.6.1 BPatch_nullExpr

Synopsis

```
#include <BPatch_snippet.h>

BPatch_nullExpr( void )
```

Description

Construct a null snippet that can be used as a place holder. This snippet contains no executable statements; however it is a useful place holder for the destination of a goto.

9.7 Class BPatch_paramExpr : public BPatch_snippet

This is implemented as a derived class of BPatch_snippet.

9.7.1 BPatch_paramExpr

Synopsis

```
#include <BPatch_snippet.h>

BPatch_paramExpr( int paramNum )
```

Parameters

paramNum	the position of the parameter (0 is the first parameter, 1 the second, and so on)

Description

This constructor creates an expression whose value is a parameter being passed to a function. paramNum specifies the number of the parameter to return (starting at 0). Since the contents of parameters may be changed during subroutine execution, this snippet type is only valid at points that are entries to subroutines, or when inserted at a call point with the when parameter set to BPatch_callBefore.

9.8 Class BPatch_retExpr : public BPatch_snippet

This is implemented as a derived class of BPatch_snippet.

9.8.1 BPatch_retExpr

Synopsis

```
#include <BPatch_snippet.h>

BPatch_retExpr( void )
```

Description

This snippet results in an expression that evaluates to the return value of a subroutine. This snippet type is only valid at BPatch_exit points, or at a call point with the when parameter set to BPatch_callAfter.

9.9 Class BPatch_sequence : public BPatch_snippet

This is implemented as a derived class of BPatch_snippet.

9.9.1 BPatch_sequence

Synopsis

```
#include <BPatch_snippet.h>
BPatch_sequence( const vector<BPatch_snippet *> &items )
```

Parameters

items	the list of snippets that are to make up the sequence

Description

Construct a snippet representing a sequence of snippets. The passed snippets will be executed in the order in which they appear in the list.

10.0 Class BPatch_sourceBlock - proposed

This class is part of the source view and represents a source level block (text inside brackets for C code) such as:

loop blocks	The code contained within the loop body.
conditional blocks	The code executed depending on the result of a condition such as an if statement.
scope blocks	A set of curly braces in C which are not associated with a function, loop or conditional statement. This block can be used to further define variable scope within a function.

10.1 Supporting Data Types

10.1.1 BPatch_sourceBlockType - proposed

Synopsis

```
#include <BPatch_module.h>
enum BPatch_sourceBlockType{
     BPatch_sourceLoop,
     BPatch_sourceOuterLoop,
     BPatch_sourceLoopHead,
     BPatch_sourceStatement,
     BPatch_sourceUnknown
}
```

Description

This enumeration type describes the type of source block.

10.2 getSourceBlocks - proposed

Synopsis

```
#include <BPatch_sourceBlock.h>
vector<BPatch_sourceBlock *> *getSourceBlocks( void )
```

Description

Get a list of the source blocks contained within the source block.

Return value

The return value contains a vector of source blocks contained within the source block.If the lookup fails to locate any source blocks, a list with zero elements is returned.

10.3 getSourceBlockType - proposed

Synopsis

```
#include <BPatch_sourceBlock.h>
BPatch_sourceBlockType getSourceBlockType( void )
```

Description

Determine if the source block is a loop, outerloop, loop head, statement...

Return value

Return a BPatch_sourceBlockType representing the type of the source block.

11.0 Class BPatch_sourceLoop - proposed

This class is a special case of BPatch_sourceBlock. Class BPatch_sourceLoop represents a source program block containing a loop.

11.1 getContainedLoops - proposed

Synopsis

```
#include <BPatch_sourceLoop.h>

vector <BPatch_sourceLoop> *getContainedLoops( void )
```

Description

Return all top level contained loops.

Return value

Return a vector of all top level contained loops. If the lookup fails to locate any top level contained loops, a list with zero elements is returned.

See Also

getContainingLoops

11.2 getContainingLoop - proposed

Synopsis

```
#include <BPatch_sourceLoop.h>
BPatch_sourceLoop *getContainingLoop( void )
```

Description

Return containing loop in the loop hierarchy. For outermost loops NULL will be returned

See Also

getContainedLoops

11.3 getLoopIterators - proposed

Synopsis

```
#include <BPatch_sourceLoop.h>
vector<BPatch_variableExpr> *getLoopIterators( void )
```

Description

Return loop iterators.

Return value

Return a vector of all loop iterators. If the lookup fails to locate any loop iterators, a list with zero elements is returned.

12.0 Class BPatch_sourceObj- proposed

This class BPatch_sourceObj itself will not be exposed. The member functions will be inherited and available to its derived classes.

Following is the class hierarchy for the IBM implementation.

- ⇒ BPatch_sourceObj
  - ⇒ BPatch_image
  - ⇒ BPatch_module
  - ⇒ BPatch_function
  - ⇒ BPatch_sourceBlock
    - ⇒ BPatch_sourceLoop
    - ⇒ BPatch_statement

???: BPatch_sourceObj.h - Is it ok to expose this header file to provide access to the declarations of its member functions?

12.1 Supporting Data Types

12.1.1 BPatch_sourceType

Synopsis

```
#include <BPatch_sourceObj.h>
enum BPatch_sourceType {
     BPatch_unknown_type,
     BPatch_program,
     BPatch_module,
     BPatch_function,
     BPatch_outerLoop,
     BPatch_loop,
     BPatch_block,
     BPatch_statement,
     BPatch_LAST_TYPE
}
```

Description

This enumeration type describes whether the source object to which it applies represents a whole program, module, function, data object, *etc*.

12.2 findAllPoints - proposed

Synopsis

```
#include <BPatch_sourceObj.h>
vector<BPatch_point *> *findAllPoints(
     const BPatch_procedureLocation loc )
```

Parameters

loc	the type of points within the object to return

Description

Lookup the instrumentation points from an object and in its included objects.

The points returned depends on the object being used.

BPatch_image	points contained within the programs contained objects below
BPatch_module	points contained within the modules contained objects below
BPatch_function	points contained within the function and its contained objects below
BPatch_sourceBlock	points contained within the source block and its contained objects below
BPatch_sourceLoop	points contained within the source loop
BPatch_sourceStatement	points contained within the source statement

Return value

Returns a vector of the objects's instrumentation points inclusively. If the lookup fails to locate any points of the requested type, a list with zero elements is returned.

See Also

`findPoint, BPatch_point`

12.3 findPoint - proposed

Synopsis

```
#include <BPatch_sourceObj.h>
vector<BPatch_point *> *findPoint(
     const BPatch_procedureLocation loc )
```

Parameters

loc	the type of points within the object to return

Description

Lookup the instrumentation points from an object, but not in included objects.

The points returned depends on the object being used.

BPatch_image	returns an empty list
BPatch_module	returns an empty list
BPatch_function	BPatch_entry and BPatch_exit points within the function
BPatch_sourceBlock	all but BPatch_entry and BPatch_exit points in the source block
BPatch_sourceLoop	all but BPatch_entry and BPatch_exit points in the source loop
BPatch_sourceStatement	

Return value

Returns a vector of the objects's instrumentation points exclusively. If the lookup fails to locate any points of the requested type, a list with zero elements is returned.

See Also

`findAllPoints, BPatch_point`

12.4 findVariable

Synopsis

```
#include <BPatch_sourceObj.h>

const BPatch_variableExpr *findVariable(const char *name )
```

Parameters

name	the name of the variable to look up

Description

Lookup and return a handle to the named variable within the object. The returned BPatch_variableExpr can be used to create references (uses) of the variable in subsequent snippets.

First look for the name with an “_” prepended to it, and if that is not found try the original name.

The type of variable depends on the object being used.

BPatch_image	global variables
BPatch_module	global and static variables defined at the module level
BPatch_function	function parameters
BPatch_sourceBlock	variables declared in the source block
BPatch_sourceLoop	variables declared within the source loop
BPatch_sourceStatement	returns NULL

Return value

non-NULL	contains a BPatch_variableExpr* representing the named global variable
NULL	failure, no such variable exists

See Also

`getVariables`

12.5 getAddressRange - proposed

Synopsis

```
#include <BPatch_sourceObj.h>
bool getAddressRange(
     BPatch_address startAddress
     BPatch_address endAddress )
```

Parameters

startAddress	this gets set to the address of the start of the object
endAddress	this gets set to the address of the end of the object

Description

Get the beginning and ending addresses of the object. For example, if the object is a function, the addresses of the starting and ending instructions of the function will be returned.

Return value

True	start and end addresses have been set
False	unable to get start and end addresses

12.6 getAllBasicBlocks - proposed

Synopsis

```
#include <BPatch_sourceObj.h>
vector<BPatch_basicBlock *> *getAllBasicBlocks( void )
```

Description

Return a list of all of the basic blocks included in this source object and any of its contained source objects.

Return value

The return value contains a vector of BPatch_basicBlocks. If the lookup fails to locate any BPatch_basicBlocks, a list with zero elements is returned.

???: Is the following correct?

When this function is used from the following objects, an empty list is always returned:

BPatch_image

BPatch_module

See Also

`getBasicBlockStart, getBasicBlockEnd, getBasicBlocks`

12.7 getBasicBlocks - proposed

Synopsis

```
#include <BPatch_sourceObj.h>
vector<BPatch_basicBlock *> *getBasicBlocks( void )
```

Description

Return a list of all of the basic blocks included in this source object but not from any of its contained source objects.

Return value

The return value contains a vector of BPatch_basicBlocks. If the lookup fails to locate any BPatch_basicBlocks, a list with zero elements is returned.

???: Is the following correct?

When this function is used from the following objects, an empty list is always returned:

BPatch_image

BPatch_module

See Also

`getBasicBlockStart, getBasicBlockEnd, getAllBasicBlocks`

12.8 getBasicBlockEnd - proposed

Synopsis

```
#include <BPatch_sourceObj.h>

vector<BPatch_basicBlock *> *getBasicBlockEnd( void )
```

Description

Return a list of all basic blocks which have a successor outside the object or which have an unknown successor.

Return value

The return value contains a vector of BPatch_basicBlocks. If the lookup fails to locate any BPatch_basicBlocks, a list with zero elements is returned.

???: Is the following correct?

When this function is used from the following objects, an empty list is always returned:

BPatch_image

BPatch_module

See Also

`getBasicBlockStart, getAllBasicBlock, getBasicBlocks`

12.9 getBasicBlockStart - proposed

Synopsis

```
#include <BPatch_sourceObj.h>

vector<BPatch_basicBlock *> *getBasicBlockStart( void )
```

Description

Return a list of all basic blocks which have a predecessor outside the object or which do not have a known predecessor.

Return value

The return value contains a vector of BPatch_basicBlocks. If the lookup fails to locate any BPatch_basicBlocks, a list with zero elements is returned.

???: Is the following correct?

When this function is used from the following objects, an empty list is always returned:

BPatch_image

BPatch_module

See Also

`getBasicBlockEnd, getAllBasicBlock, getBasicBlocks`

12.10 getLineNumbers - proposed

Synopsis

```
#include <BPatch_sourceObj.h>
bool getLineNumbers(
     int &start,
     int &end )
```

Parameters

start	this gets set to the first source line number of the object
end	this gets set to the last source line number of the object

Description

Gets the first and last line number in the object.

Use this function to get the approximate starting and ending source line numbers of the object which are relative to the start of the containing file.

For functions, the start line will reflect the first line number in the line number table that falls between the starting and ending address of the function. This is generally the first source line that generates executable code.

Return value

The return value indicates if getLineNumbers was able to get the source line numbers.

True	start and end have been set
False	unable to get start and or end

Function getLineNumbers returns false if their is no line number information available (i.e., stripped or compiled without debugging).

For the following objects, false will always be returned:

BPatch_image

See Also

`getRange`

12.11 getObjParent - proposed

Synopsis

```
#include <BPatch_sourceObj.h>
BPatch_sourceObj getObjParent( void )
```

Note: Need to keep BPatch_sourceObj from being exposed.

Description

This function returns the parent object of this object. For example, the parent object of a function object is a module object. The parent object of a program object is itself.The parent object of a block may be another block object, a loop block object or a function object.

Return value

Returns the parent object of the object.

See Also

`getSrcType`

12.12 getRange - proposed

Synopsis

```
#include <BPatch_sourceObj.h>
bool getRange(
     BPatch_statement *starts,
     BPatch_statement *ends )
```

Parameters

starts	this gets set to the first BPatch_statement of the object
ends	this gets set to the last BPatch_statement of the object

Description

Gets the first and last statement in the object.

Return value

The return value indicates if getRange was able to get the statements.

True	start and end have been set
False	unable to get start and or end

For the following objects, false will always be returned:

BPatch_image

See Also

`getLineNumbers`

12.13 getSrcType - proposed

Synopsis

```
#include <BPatch_sourceObj.h>
BPatch_sourceType getSrcType( void )
```

Description

Get the type of the object. The source object type corresponds to various objects within a program, such as modules, functions, variables, etc. If the source object does not correspond to a program or part of a program, the source object type is “unknown”

Return value

Returns the type of the object.

See Also

`getObjParent`

12.14 getType - proposed

Synopsis

```
#include <BPatch_sourceObj.h>
#include <BPatch_type.h>
Bpatch_type getType(char *name )
Bpatch_type getType(int typeNumber )
```

Parameters

name	The name of the type.
typeNumber	The typeNumber is an unique number (per BPatch_module) given by the compiler to any type appearing in the module.

Description

Get the specified type.

Return value

Return the required type or NULL for failure.

See Also

`class BPatch_type`

12.15 getVariables - proposed

Synopsis

```
#include <BPatch_sourceObj.h>
vector<BPatch_variableExpr *> *getVariables( void )
```

Description

Get a list of variables contained in the scope of the object.

The type of variables in the list depends on the object being used.

BPatch_image	returns empty list
BPatch_module	global and static variables defined in the module
BPatch_function	function parameters
BPatch_sourceBlock	variables declared in the source block
BPatch_sourceLoop	variables declared within the source loop
BPatch_sourceStatement	returns empty list

Return value

The return value contains a vector of BPatch_variableExpr's representing the variables contained in the object. If the lookup fails to locate any variables, a list with zero elements is returned.

See Also

```
findVariable
```

13.0 Class BPatch_statement - proposed

This class represents a single statement in the source view.

13.1 isCall - proposed

Synopsis

```
#include <BPatch_statement.h>
bool isCall( void )
```

Description

Returns true if the statement contains a call site

14.0 Class BPatch_thread

The BPatch_thread class operates on (and creates) code in execution.

14.1 catchSignal - proposed

???: Do we need a showCaughtSignals member?

Synopsis

```
#include <sys/signal.h>
#include <BPatch_thread.h>
bool catchSignal( sigset_t sa_mask )
```

Parameters

sa_mask	used to specify the individual signals to be caught

Description

The catchSignal call allows setting the BPatch reaction to particular signals that occur in the application. After catchSignal has completed successfully, signals represented in the signal_mask parameter will be placed in the list of caught signals.

A process under ptrace control executes normally until it encounters a signal. Processes that were started by the daemon using createProcess or attached to with attachProcess are under ptrace control. They will continue to be under ptrace control until detach is issued. After the process under ptrace control encounters a signal, the process enters the stopped state and the daemon is notified with a SIGCHLD.

When the application encounters a “caught” signal, the application will remain in the stopped state until being restarted by the daemon.

Then the application encounters an “ignored” signal, BPatch will restart the application as soon as possible and allow the application to handle the signal in its own fashion.

The following signals are caught by default:

???

The following signals should always be caught:

SIGSTOP	used by the bpatch library
SIGTRAP	used to implement debugger breakpoints

The following signals should always be ignored:

SIGPROF	used to implement Dais phases

The following signals should always be ignored when instrumenting poe jobs running with the signal based MPI library:

SIGALRM	used by MPI
SIGPIPE	used by MPI
SIGIO	used by MPI

The following signals should always be ignored when instrumenting poe jobs running with the threads based MPI library:

SIGPIPE	used by MPI
SIGIO	used by MPI

Return value

True	the signals specified by sa_mask will be caught
False	failure, no change has been made to the list of signals which were previously caught

See Also

`ignoreSignal, stopSignal, signal.h`

14.2 continueExecution

Synopsis

```
#include <BPatch_thread.h>
bool continueExecution( void )
```

Description

Puts the thread into the running state.

Return value

The return value indicates if continueExecution was able to get the thread to run.

True	the thread was run
False	unable to run the thread

See Also

stopExecution, terminateExecution, isStopped, isTerminated, stopSignal

14.3 deleteSnippet

Synopsis

```
#include <BPatch_thread.h>

bool deleteSnippet( BPatchSnipperHandle *handle )
```

Parameters

handle	The handle returned by insertSnippet when the instance to be deleted was created.

Description

Deletes an instance of a snippet.

Return value

The return value indicates if the handle passed in was valid for the thread.

True	valid handle passed in
False	invalid handle passed in

See Also

`insertSnippet, setInheritSnippets`

14.4 detach

Synopsis

```
#include <BPatch_thread.h>

void detach( bool cont )
```

Parameters

cont	True if the thread should be continued as the result of the detach, false if it should not.

Description

Detach from the thread represented by this object.The thread must be stopped to call this function. The cont parameter is used to indicate if the thread should be continued as a result of detaching.

Return value

none

See Also

stopExecution, isStopped

14.5 dumpCore

Synopsis

```
#include <BPatch_thread.h>
bool dumpCore(
     const char *file,
     bool terminate )
```

Parameters

file	the name of the file to which the state should be written
terminate	indicates whether or not the thread should be terminated after dumping core. True indicates that it should, false that it should not.

Description

Causes the thread to dump its state to a file, and optionally to terminate. If terminate is set to false and the application is to continue, the application will not continue until after the dump is complete.

???: What is the behavior of terminate (when True) if dumping core is not successful? Should termination not start until after the dump completes? If so we should say that upon failure, the dump did not occur and the process was not terminated.

Return value

Returns true upon successfully completing a dump, and false upon failure.

True	success
False	failure

See Also

dumpImage

14.6 dumpImage - proposed

Synopsis

```
#include <BPatch_thread.h>

bool dumpImage(const char *file )
```

Parameters

file	the name of the file to which the image should be written

Description

This function causes the thread to write the in-memory version of the program to the specified file.

Return value

Returns true upon successfully completing an image dump, and false upon failure.

True	success
False	failure

See Also

dumpCore

14.7 free

Synopsis

```
#include <BPatch_thread.h>

void free( const BPatch_variableExpr &ptr )
```

Parameters

ptr	A BPatch_variableExpr representing the memory to free.

Description

Free memory that was allocated with BPatch_thread::malloc.The programmer is responsible to verify that all code that could reference this memory will not execute again (either by removing all snippets that refer to it, or by analysis of the program).

Return value

none

???: should this have a return value to indicate that the BPatch_variableExpr had not been malloc'ed using BPatch_thread::malloc?

See Also

```
BPatch_thread::malloc
```

14.8 getImage

Synopsis

```
#include <BPatch_thread.h>
const BPatch_image *getImage( void )
```

Description

Get the executable file associated with this BPatch_image object and return a handle to it.

Return value

Returns a BPatch_image*.

14.9 getPid

Synopsis

```
#include <BPatch_thread.h>
pid_t getPid( void )
```

Description

Return the process id represented by the current object.

14.10 ignoreSignal - proposed

???: Do we need a showIgnoredSignals member?

Synopsis

```
#include <sys/signal.h>
#include <BPatch_thread.h>
bool ignoreSignal( sigset_t sa_mask )
```

Parameters

sa_mask	used to specify the individual signals to be ignored

Description

The ignoreSignal call allows setting the BPatch reaction to particular signals that occur in the application. After ignoreSignal has completed successfully, signals represented in the signal_mask parameter will be placed in the list of ignored signals.

A process under ptrace control executes normally until it encounters a signal. Processes that were started by the daemon using createProcess or attached to with attachProcess are under ptrace control. They will continue to be under ptrace control until detach is issued. After the process under ptrace control encounters a signal, the process enters the stopped state and the daemon is notified with a SIGCHLD.

When the application encounters a “caught” signal, the application will remain in the stopped state until being restarted by the daemon.

Then the application encounters an “ignored” signal, BPatch will restart the application as soon as possible and allow the application to handle the signal in its own fashion.

The following signals are caught by default:

???

The following signals should always be caught:

SIGSTOP	used by the bpatch library
SIGTRAP	used to implement debugger breakpoints

The following signals should always be ignored:

SIGPROF	used to implement Dais phases

The following signals should always be ignored when instrumenting poe jobs running with the signal based MPI library:

SIGALRM	used by MPI
SIGPIPE	used by MPI
SIGIO	used by MPI

The following signals should always be ignored when instrumenting poe jobs running with the threads based MPI library:

SIGPIPE	used by MPI
SIGIO	used by MPI

Return value

True	the signals specified by sa_mask will be ignored
False	failure, no change has been made to the list of signals which were previously ignored

See Also

`catchSignal, stopSignal, signal.h`

14.11 insertSnippet

Synopsis

```
#include <BPatch_thread.h>
BPatchSnippetHandle *insertSnippet(
     const BPatch_snippet &expr,
     const BPatch_point &point,
     BPatch_callWhen when = BPatch_callBefore,
     BPatch_snippetOrder order = BPatch_firstSnippet )
```

Parameters

expr	the snippet to insert
point	the point at which to insert it
when	when the snippet is to be called
order	insertion point relative to any other snippets already installed at the same point

Description

Insert a code snippet at a given instrumentation point. The when argument specifies when the snippet is to be called; a value of BPatch_callBefore indicates that the snippet should be inserted just before the specified point or points in the code, and a value of BPatch_callAfter indicates that it should be inserted just after. The order argument specifies where the snippet is to be inserted relative to any other snippets previously inserted at the same point. The values BPatch_firstSnippet and BPatch_lastSnippet can be used to indicate that the snippet should be inserted be-fore or after all such snippets, respectively.

Return value

Returns a handle to the created instance of the snippet.

Success	returns the snippet handle
Failure	returns NULL

See Also

`deleteSnippet, setInheritSnippets`

14.12 insertSnippet

Synopsis

```
#include <BPatch_thread.h>
BPatchSnippetHandle *insertSnippet(
    const BPatch_snippet &expr,
    const vector<BPatch_point *> &points,
    BPatch_callWhen when = BPatch_callBefore,
    BPatch_snippetOrder order = BPatch_firstSnippet )
```

Parameters

expr	the snippet to insert
points	the list of points at which to insert it
when	when the snippet is to be called
order	insertion point relative to any other snippets already installed at the same point

Description

Insert a code snippet at each of a list of instrumentation points. The when argument specifies when the snippet is to be called; a value of BPatch_callBefore indicates that the snippet should be inserted just before the specified point or points in the code, and a value of BPatch_callAfter indicates that it should be inserted just after. The order argument specifies where the snippet is to be inserted relative to any other snippets previously inserted at the same point. The values BPatch_firstSnippet and BPatch_lastSnippet can be used to indicate that the snippet should be inserted be-fore or after all such snippets, respectively.

Return value

Returns a handle to the created instances of the snippet, which can be used to delete them (as a unit).

Success	returns the snippet(s) handle
Failure	returns NULL

See Also

`deleteSnippet, setInheritSnippets`

14.13 isStopped

Synopsis

```
#include <BPatch_thread.h>
bool isStopped( void )
```

Description

This function queries the status of the thread to see if it is currently stopped. If the thread is stopped, then stopSignal can be called to find out what signal caused the thread to stop.This function may be called multiple times and will not affect the state of the thread.

Valid states for a thread are:

neonatal	thread is involved in BPatch initialization
running	thread is executing
stopped	thread is stopped, which allows certain BPatch calls to be made
exited	thread has terminated and is no longer available for BPatch control

Return value

The return value indicates if the thread is currently stopped.

True	the thread is stopped
False	the thread is not stopped

See Also

```
continueExecution, stopExecution, terminateExecution,
isTerminated, stopSignal
```

14.14 isTerminated

Synopsis

```
#include <BPatch_thread.h>
bool isTerminated( void )
```

Description

This function queries the status of the thread to see if it has terminated. This function may be called multiple times and will not affect the state of the thread.

Valid states for a thread are:

neonatal	thread is involved in BPatch initialization
running	thread is executing
stopped	thread is stopped, which allows certain BPatch calls to be made
exited	thread has terminated and is no longer available for BPatch control

Return value

The return value indicates if the thread has terminated.

True	the thread is terminated
False	the thread is not terminated

See Also

`continueExecution, stopExecution, terminateExecution, isStopped, stopSignal`

14.15 malloc

Synopsis

```
#include <BPatch_thread.h>

BPatch_variableExpr *malloc(int n )
```

Parameters

n	The number of bytes to allocate

Description

Allocate memory in the thread's address space.

Return value

Returns a pointer to a BPatch_variableExpr representing the memory.

Returns NULL on failure.

Note: Should return NULL on failure, but the function which it calls, inferiorMalloc, calls exit rather than returning an error, so this is not currently possible in the current implimentation.

See Also

`BPatch_thread::free`

14.16 malloc

Synopsis

```
#include <BPatch_thread.h>
BPatch_variableExpr *malloc( const BPatch_type &type )
```

Parameters

type	The type of variable for which to allocate space.

Description

Allocate memory in the thread’s address space for a variable of the given type

Return value

Returns a pointer to a BPatch_variableExpr representing the memory.

Returns NULL on failure.

Note: Should return NULL on failure, but the function which it calls, inferiorMalloc, calls exit rather than returning an error, so this is not currently possible in the current implimentation.

See Also

`BPatch_thread::free`

14.17 oneTimeCode - proposed

Synopsis

```
#include <BPatch_thread.h>
void oneTimeCode( const BPatch_snippet &expr )
```

Parameters

expr	the snippet to insert

Description

Cause snippet to be evaluated once at the next available opportunity. This interface is useful to cause an initialization function to be called in the application. The process must be stopped to call this function.

Return value

none

14.18 setInheritSnippets - proposed

Synopsis

```
#include <BPatch_thread.h>
void setInheritSnippets( bool inherit )
```

Parameters

inherit	true if snippets should be inherited by the forked process

Description

Set a flag to indicate if instrumentation snippets should be inherited when the thread forks. By default, instrumentation snippets are inherited by the child process.

Return value

none

14.19 stopExecution

Synopsis

```
#include <BPatch_thread.h>

bool stopExecution( void )
```

Description

Puts the thread into the stopped state.

Return value

The return value indicates whether the thread was stopped by stopExecution or not.

True	the thread was stopped
False	the thread was not stopped by stopExecution for one of a variety of reasons such as, already stopped, thread is before the initial stop point, thread has exited...

See Also

```
continueExecution, terminateExecution, isStopped,
isTerminated, stopSignal
```

14.20 stopSignal

Synopsis

```
#include <BPatch_thread.h>

int stopSignal( void )
```

Description

This function queries the signal number which caused the thread to stop. It can only be called if the thread is in the stopped state. Use function isStopped to find out if the thread is in the stopped state. This function may be called multiple times and will not affect the state of the thread.

If stopSignal is called when the thread is not in the stopped state, the previous signal encountered by the application will be reported.

Valid states for a thread are:

neonatal	thread is involved in BPatch initialization
running	thread is executing
stopped	thread is stopped, which allows certain BPatch calls to be made
exited	thread has terminated and is no longer available for BPatch control

Return value

The return value contains the signal number which caused the thread to stop.

See Also

```
continueExecution, stopExecution, terminateExecution,
isStopped, isTerminated, signal.h
```

14.21 terminateExecution

Synopsis

```
#include <BPatch_thread.h>
bool terminateExecution( void )
```

Description

Terminates execution of the thread.

Return value

The return value indicates if the thread was terminated by terminateExecution.

True	the thread was terminated
False	the thread was not terminated by terminateExecution for one of a variety of reasons such as, thread has exited, not a valid BPatch_thread, more?

See Also

`continueExecution, stopExecution, isStopped, isTerminated, stopSignal`

15.0 Class BPatch_type - proposed

This class represents the type information of a variable.

15.1 Supporting Data Types

15.1.1 BPatch_typeType

Synopsis

```
#include <BPatch_type.h>
enum BPatch_typeType {
     BPatch_unknown_type,
     BPatch_primitive,
     BPatch_scalar,
     BPatch_enumerated,
     BPatch_structure,
     BPatch_union,
     BPatch_array,
     BPatch_type_number,
     BPatch_pointer,
     BPatch_LAST_TYPE
}
```

Description

This enumeration type describes the general types available for BPatch_type.

15.2 Constructors - proposed

Synopsis

```
#include <BPatch_type.h>
BPatch_type(char *name )
BPatch_type(
     char *name,
     BPatch_module *module,
     bool nulltype=false )
```

Parameters

name	The type name (can be NULL)
module	Module where the type is defined.
nulltype	If true then this is typeless.

Description

The first constructor with just the name parameter creates a BPatch_type which is available in all modules.

The second constructor creates a BPatch_type which is available in the module specified.

15.3 getComponents - proposed

Synopsis

```
#include <BPatch_type.h>
vector <BPatch_variableExpr> *getComponents(
          BPatch_variableExpr var )
```

Parameters

var	The complex variable to expand.

Description

Get a vector containing the fields of the passed in structure or union.

Return value

Return a list of the components contained in the complex variable designated by the var parameter. If the components of the variable cannot be found or if var is a scaler type variable, then a list of zero elements is returned.

15.4 getDescription - proposed

Synopsis

```
#include <BPatch_type.h>
vector<char *> *getDescription( void )
```

Description

This API should be the complete description for all the posible types in the executable. The type is classified with the type enumerated (from the type() API) and according to that, the user will know the format of the vector of strings returned by this API. For now we wil return the string appearing at the XCOFF file, parsing is user dependent. Structures and unions are described with the getComponents API

Structure or Union:

fieldName, fieldTypeNumber, fieldOffset, fieldName, fieldTypeNumber, fieldOffset, etc...

Enumerated (always integer):

eumValue, value, enumValue, value. etc...

Array

arrayTypeNumber, dimension

Pointer :

*, typeNumberPointed

TypeNumber

typeNumber

Primitive

NULL (primitive type has always a name)

Return value

Return a list of strings which describe the types contained in the executable. A list with no elements is returned if the type descriptions cannot be determined.

15.5 getName - proposed

Synopsis

```
#include <BPatch_type.h>
const char *getName( void )
```

Description

Get the name of the type.

Return value

Return the name of the type or NULL of the name is not available.

15.6 getSize - proposed

Synopsis

```
#include <BPatch_type.h>
int getSize( void )
```

Description

Get the size of the type.

Return value

Return the size of the type or -1 if the size cannot be determined.

15.7 getTypeNumber - proposed

Synopsis

```
#include <BPatch_type.h>
int getTypeNumber( void )
```

Description

Get the number of the type. The type number will be negative if it is a primitive type.

Return value

Return the number of the type.

???: What will an error return?

See Also

dbxstclass.h

15.8 isCompatible - proposed

Synopsis

```
#include <BPatch_type.h>
bool isCompatible( const BPatch_type &otype )
```

Parameters

otype	The other type to compare with this type.

Description

Compare two types. Return true if either one of the type is typeless or they are equal.

???: What is equal? name, description, size?

15.9 isStructure - proposed

Synopsis

```
#include <BPatch_type.h>
bool isStructure( void )
```

Description

Return true if the type is a structure or union.

???: What about enumerated types?

???: What about arrays?

Return value

Return true for complex types and false for scaler types.

15.10 type - proposed

Synopsis

```
#include <BPatch_type.h>

BPatch_Typetype type( void )
```

Description

Get the type of the object.

???: Does this eliminate the need for isStructure?

Return value

Return an enum representing the type of the object. If the type cannot be determined, return BPatch_unknown_type.

16.0 Class BPatch_variableExpr

This class represents both a variable expression in the source view and also a variable which may be used in building an expression.

The BPatch_variableExpr class is another class derived from snippet. It represents a variable or area of memory in a thread's address space. A BPatch_variableExpr can be obtained from a BPatch_thread using the malloc member function, or from a BPatch_image using the findVariable member function. BPatch_variableExpr provides additional member functions not provided by other types of snippets.

This is implemented as a derived class of BPatch_snippet.

16.1 Constructors

???: These constructors are not exposed currently in BPatch. To expose them, do we need to change the interface from passing in a process class to passing in a BPatch_thread?

Synopsis

```
#include <BPatch_variableExpr.h>
BPatch_variableExpr(
     process *in_process,
     void *in_address,
     const BPatch_type *type )
BPatch_variableExpr(
     char *name,
     process *in_process,
     void *in_address,
     const BPatch_type *type )
```

Parameters

name	name of the new variable
in_process	process for which to generate the variable
in_address	address in process space for the variable
type	type of the variable

Description

Create a BPatch_variableExpr. If in_address is NULL, the variable will be allocated in the process heap.

16.2 duplicateVariable - proposed

Synopsis

```
#include <BPatch_variableExpr.h>
BPatch_variableExpr *getComponent( const char *dupname )
```

Parameters

dupname	name of the new duplicate variable

Description

Returns a duplicate of the variable.

Return value

Return NULL if the variable cannot be duplicated.

16.3 findPoint - proposed

Synopsis

```
#include <BPatch_variableExpr.h>

vector<BPatch_point *> *findPoint( void )
```

Description

Returns all instrumentation points defined for the variable. Instrumentation points for a variable are before and after initialization.

Return value

Returns a vector of the variables's instrumentation points. If the lookup fails to locate any points, a list with zero elements is returned.

16.4 getComponent - proposed

Synopsis

```
#include <BPatch_variableExpr.h>
BPatch_variableExpr *getComponent(const char *name )
```

Parameters

name	name of the component

Description

Get a component of a complex variable.

Return value

Returns a component of a complex variable.
Returns NULL if the variable is scalar.

16.5 getComponents - proposed

Synopsis

```
#include <BPatch_variableExpr.h>
vector <BPatch_variableExpr> *getComponents( void )
```

Description

Get the components of a complex variable such as a structure or union.

Return value

Returns a vector of the components of a complex variable. If the lookup fails to locate any components (which will happen for a scalar variable), a list with zero elements is returned.

???: What will this return for arrays?

16.6 getSize - proposed

Synopsis

```
#include <BPatch_variableExpr.h>
int getSize( void )
```

Description

Get the size of the variable.

Return value

Return the size of the variable or -1 if the size cannot be determined.

16.7 getType - proposed

Synopsis

```
#include <BPatch_variableExpr.h>
BPatch_type *getType( void )
```

Description

Get the type of the variable.

Return value

Return the type of the variable or NULL is no type information is available.

16.8 writeValue

Synopsis

```
#include <BPatch_snippet.h>

void writeValue(const void *src )
```

Parameters

src	A pointer to a buffer in which to place the value of the variable. It is assumed to be the same type as the variable.

Description

Changes the value of the variable in an application's address space that is represented by this BPatch_variableExpr. The src parameter should point to a value of the variable's type.

Return value

none

16.9 writeValue

Synopsis

```
#include <BPatch_snippet.h>
void writeValue(
     const void *src,
     int size )
```

Parameters

src	buffer in which to place the value of the variable
size	number of bytes to write

Description

Changes the value of the variable in an application’s address space that is represented by this BPatch_variableExpr for size bytes.

Return value

none

References:

1. Jeffrey K. Hollingsworth & Bryan Buck, “DyninstAPI Programmer’s Guide”
2. Avi Zehavi, “A.OUT Reader New API’s”
3. DyninstAPI source files
4. Avi Zehavi & Guillermo Rabinovich, “BPatch Design”