Visual C++ Browser Toolkit for Windows NT Library Reference

Microsoft Visual C++ Browser Toolkit for Windows NT

This document discusses the Microsoft Visual C++ Browser Toolkit API definitions. It explains how to use the browser library to access the contents of a browser database.

Contents

1 Using the Browser Library

The Browser Toolkit includes a powerful library of functions that give you complete access to browser database files created by Microsoft Visual C++. This browser library, named BSC.LIB, contains functions that access browser database files (.BSC), query database tables, and even generate information, such as calltrees and symbol reference tables. Additionally, BSC.LIB incorporates functions that access and manipulate information about C++ classes and objects that may be present in a browser database. For example, some of these functions create classtrees and return undecorated names of C++ functions (typically stored in decorated form).

To provide BSC.LIB users maximum flexibility, the Browser Toolkit allows you to implement several callback functions described below (see section 2.1).

A callback function is one that performs a particular system service required by the browser library. Some examples of such services are memory allocation, file I/O, and error-handling. The BSC.LIB file provides callbacks so that the browser library does not have to depend on the run-time library for these services. Instead, library functions "call back" to the function designated to perform a particular service without the library requiring any knowledge of how this service is performed. Because you have the option of writing your own callbacks, these functions provide a flexible interface to any environment that might require special access to run-time resources.

1.1 Compiling Programs

To compile a browser program (such as the BSCDUMP.C program used in this example) and link it with BSC.LIB at the MS-DOS prompt, you can use the following command:

```
cl -W3 -I. bscdump.c bsc.lib -link -nodefaultlib:libcmt.lib
  -defaultlib:libc.lib
```

NOTE: If you choose to define the callbacks (described in the following section) required by the library, they will not be linked from BSC.LIB. Defining these callbacks yourself allows BSC.LIB to be used in any custom environments you may have (such as Windows real mode, for example). Be sure to link with the appropriate file(s) containing your callback definitions

2 Browser Database Functions

The following sections describe various browser database functions.

2.1 Callback Functions

The BSC.LIB file provides default versions of all callback functions. These defaults should provide you with the necessary flexibility to develop programs for most environments. However, if an environment does not permit the use of normal run-time calls, you must define several callback functions to avoid dependency on the C standard I/O library. Table 2 outlines the prototypes for functions that handle basic I/O and memory allocation and deallocation. If you implement these functions yourself, ensure that your versions follow the specifications described below.

Table 2: Callback Functions

Callback To	Prototype	Result
Standard memory allocation function	LPV BSC_API LpvAllocCb (WORD cb)	A void far pointer must be returned to a block of size cb bytes. If the block of memory has not been allocated, then the return value should be NULL.
Standard memory deallocation function	VOID BSC_API FreeLpv (LPV lpv)	Given a void far pointer to a block of memory lpv, the function must deallocate the memory block.
Standard file open function	int BSC_API BSCOpen(LSZ lszFileName, int mode)	Opens the file lszFileName and prepares the file for subsequent reading or writing, as defined by mode. The function must return a file handle for the opened file. A return value of -1 should indicate an error. The mode parameter may be any combination of the O_* bits that are used in the standard open() call. At this time, however, the library will make this call only with mode = **O_BINARY | O_RDONLY**.
Standard file close function	int BSC_API BSCClose (int handle)	Closes the file with the given handle value. A return of 0 should indicate success.
Standard file read function	int BSC_API BSCRead (int handle, LPCH lpchBuf, WORD ch)	Reads a number of bytes ch from the file with the given handle into the far character buffer pointed to by lpchBuf. The return value should be the number of actual bytes that were read, or -1 if an error occurred.

Callback To	Prototype	Result
Standard file seek function	int BSC_API BSCSeek (int handle, long lPos, int mode)	Moves the file pointer associated with the file with the given handle to a new location, lpos bytes from the position in the file determined by mode. Here, mode can be any of **SEEK_CUR**, **SEEK_END**, or **SEEK_SET** as in the lseek() call from the C standard library. Currently, only **SEEK_SET** is used by BSC.LIB. The return value should be -1 on an error; any other value indicates success.
ASCII text output function	VOID BSC_API BSCOutput(LSZ lsz)	Writes the given zero-terminated string to the standard output. This function is called by the library whenever it wants to output any text (that is, for a calltree or classtree, outline, or other request). The default implementation writes to the standard output; however, a user implementation of this function can direct the output elsewhere.
Error-handling functions	VOID BSC_API SeekError (LSZ LszFilename) VOID BSC_API ReadError (LSZ LszFilename) VOID BSC_API BadBSCVer (LSZ LszFilename)	These functions should be provided to handle the occurrence of any one of the corresponding errors. The string parameter is the name of the file involved in the error. If these functions choose to exit the application, they should close any file(s) the browser has opened.

2.2 Opening and Closing a Browser Database

Two browser library functions open and close a browser database file. The following table details the prototypes for these functions.

Table 3: File Opening/Closing Functions

Purpose	Prototype	Result
Opens the .BSC file	BOOL BSC_API FOpenBSC (LSZ lszName)	Opens the specified database file lszName. Returns the value TRUE if successful, or FALSE if unsuccessful. The lszName parameter can be any valid path name to a .BSC file.
Closes the .BSC file	VOID BSC_API CloseBSC (VOID)	Closes the current database file. This will free any memory the library may be using.

2.3 Database Index Variables

Each object found in the browser database is represented by a numerical index value that uniquely identifies one database object. The name of every index variable type begins with the letter "I" and ends with letters referring to the kind of object indexed by the variable. Each of these index variable types and the functions that operate on them are described below.

2.3.1 Modules (IMOD Variables)

Each module in the database has an associated module index (IMOD) value. Using the functions detailed in the following table, you can convert a filename into a module index, convert a module index into a filename, or enumerate all the module index values in the database.

Table 4: IMOD Conversion Functions

Purpose	Prototype	Result
Converts from IMOD to module name	LSZ BSC_API LszNameFrMod (IMOD imod)	LszNameFrMod (given a module index imod) returns the zero-terminated string name for that symbol index.
Converts from module name to IMOD	IMOD BSC_API ImodFrLsz (LSZ lszModName)	ImodFrLsz returns the index, if one exists, from the symbol name lszModName. If that module name does not exist, then the return value is imodNil.
Determines largest IMOD value	IMOD BSC_API ImodMac (VOID)	The return value is the biggest IMOD index in this database; the range of these indexes is from zero to the return value less one.

2.3.2 Module Symbols (IMS Variables)

The browser database contains a list of all the symbol instances (see section 2.3.4, *Symbol Instances*) defined in any particular module. These module symbols also are referenced via a module symbol index (called an IMS value). Given a module index (IMOD) variable, certain library functions can find the range of valid module symbol index (IMS) values for that module. This range of values then can be used to reveal the contents of that module. Table 5 describes functions that either operate on or return IMS values.

Table 5: IMS Values

Purpose	Prototype	Result
Determines the range of IMS values for the given IMOD	VOID BSC_API MsRangeOfMod (IMOD imod, IMS far *pimsMin, IMS far *pimsMac)	The IMS values pointed to by pimsMin and pimsMac will contain the IMS start and end values for the specified module index. If pimsMin is equal to pimsMac, then the specified module (imod) contains no definitions and hence has no corresponding IMS values.
Gets the instance index (IINST) from the module symbol index (IMS)	IINST BSC_API IinstOfIms (IMS ims)	Given the index ims of a module symbol, this function returns the instance index for subsequent use in other calls. Refer to section 2.3.4 for a discussion of symbol instances.

2.3.3 Symbols (ISYM Variables)

All symbols (identifiers) of any type have an associated symbol index (ISYM); that is, there is a one-to-one correspondence between symbol indexes and symbol names known to the database. The range of valid symbol index values is determined by the call

```
ISYM BSC_API IsymMac (VOID)
```

which determines the size of a browser database's symbol table.

The return value is the biggest symbol index in the particular browser database being queried. The range of valid indexes is from zero to the return value less one. Symbol index values are assigned to symbols in alphabetical order (with one exception: a leading question mark in decorated names is ignored). For example, to obtain a list of symbols in sorted order, you need only obtain the text of symbols with ISYM values of 0, 1, 2 , 3, and so forth, up to the value returned by IsymMac less one.

Another library function determines the smallest symbol index whose value is greater than or equal to that of a given symbol name.

```
ISYM BSC_API IsymSupLsz (LSZ lszSymName)
```

Strings can be converted to symbol indexes (and vice versa) using the following calls:

```
LSZ BSC_API LszNameFrSym (ISYM isym)
ISYM BSC_API IsymFrLsz (LSZ lszSymName)
```

Given an index isym, LszNameFrSym returns the zero-terminated string name for the symbol index. IsymFrLsz returns the index, if one exists, from the symbol name lszSymName. If that symbol name does not exist in the database, the function returns isymNil.

To get data other than the name from a symbol index, you must enumerate its corresponding instance values, which correspond to the various instances of that symbol. These instance values are discussed next.

2.3.4 Symbol Instances (IINST Variables)

Every symbol in a browser database is associated with several instances, each of which corresponds to a different way the symbol is used in the program(s) described in the browser database (see Table 6). For example, the symbol "mysymbol" might be used as a static variable in one place, and as a function parameter in another. The browser library keeps track of these various instances using instance index values called IINST variables.

Table 6: IINST Values

Purpose	Prototype	Result
Determines the range of valid IINST values for a particular symbol (isym)	VOID BSC_API InstRangeOfSym(ISYM isym, IINST far *piinstMin, IINST far *piinstMac)	Fills in the values of piinstMin and piinstMac in an analogous way to MsRangeOfMod() described in Table 5.
Finds the range of valid IINST values for the whole database	IINST BSC_API IinstMac (VOID)	As with ImodMac() and IsymMac(), IinstMac returns the upper bound on valid IINST values. IINST values range from zero to the return value less one.
Retrieves information about a particular instance	VOID BSC_API InstInfo (IINST iinst, ISYM far *pisyminst, TYP far *ptype, ATR far *pattr)	Fills in the corresponding symbol, type, and attribute values (pointer to ISYM, TYP, and ATR, respectively) given an IINST. ISYM values are discussed in section 2.3.3.
Finds the IINST value that best represents the context at the location of the given IREF. For example, if you have an IREF for some variable, the context IINST would be the function that the variable was defined in.	IINST IinstContextFrIref (IREF iref)	Returns the IINST value for the instance that best represents the context at the reference indicated by the given iref. If it finds no such instance index, the function returns iinstNil. IREF variables are discussed in section 2.3.5. This function uses a heuristic, "best guess" algorithm that is highly accurate, though not 100% so.
Finds the IINST which contains the given IREF.	IINST IinstFrIref (IREF iref)	Returns the IINST value for the instance that contains the reference indicated by iref. If it finds no such instance index, the function returns iinstNil. This function will fail if iref is an invalid reference index. IREF variables are discussed in section 2.3.5.

Purpose	Prototype	Result
Finds the IINST which contains the given IDEF.	IINST IinstFrIdef (IDEF idef)	Returns the IINST value for the instance that contains the definition indicated by idef. If it finds no such instance index, the function returns iinstNil. This function will fail if idef is an invalid definition index. IDEF variables are discussed in section 2.3.5.

The type value (refer to InstInfo in Table 6) will be exactly one of the following:

INST_TYP_FUNCTION	**INST_TYP_LABEL**	**INST_TYP_PARAMETER**
INST_TYP_VARIABLE	**INST_TYP_CONSTANT**	**INST_TYP_MACRO**
INST_TYP_TYPEDEF	**INST_TYP_STRUCNAM**	**INST_TYP_ENUMNAM**
INST_TYP_ENUMMEM	**IINST_TYP_UNIONNAM**	**INST_TYP_SEGMENT**
INST_TYP_GROUP	**INST_TYP_PROGRAM**	**INST_TYP_CLASSNAM**
INST_TYP_MEMFUNC	**INST_TYP_MEMVAR**	

Note that three of these constants refer to C++ symbols: **INST_TYP_CLASSNAM**, **INST_TYP_MEMFUNC**, and **INST_TYP_MEMVAR.**

The attribute value describes the storage class and/or scope of the instance. Any combination of the attribute bits may be set, although some combinations do not make sense and thus do not occur in practice. These bits can be tested using the C/C++ languages' bitwise operators, or through the use of the FInstFilter function (described in Table 7). The attribute bits are as follows:

INST_ATR_LOCAL	**INST_ATR_STATIC**	**INST_ATR_SHARED**
INST_ATR_NEAR	**INST_ATR_COMMON**	**INST_ATR_DECL_ONLY**
INST_ATR_PUBLIC	**INST_ATR_NAMED**	**INST_ATR_MODULE**
INST_ATR_VIRTUAL	**INST_ATR_PRIVATE**	**INST_ATR_PROTECT**

Additionally, for inheritance mode attributes, the following constants have been defined:

IMODE_VIRTUAL	**IMODE_PRIVATE**	**IMODE_PUBLIC**
IMODE_PROTECT		

Several functions designed to help with some typical operations on instance index values are shown in Table 7.

Table 7: Attribute-Testing Functions

Purpose	Prototype	Result
Acts as an attribute filter	BOOL BSC_API FInstFilter (IINST iinst, MBF mbf)	FInstFilter returns TRUE if the given IINST value has a TYP value that corresponds with the value passed in mbf. See the list of possible MBF values below this table.

Purpose	Prototype	Result
Gets an ASCII version of an IINST variable type	LSZ BSC_API LszTypInst (IINST iinst)	Returns the zero-terminated ASCII string from the instance index iinst. These correlate directly with the types given above as INST_TYP_LABEL and so on.
Dumps a single instance	VOID BSC_API DumpInst (IINST iinst)	Outputs information about the instance indexed by iinst to the standard output using the BSCOutput() callback. The format of the output is described below.

FInstFilter requires an instance type filter as one of its parameters. The instance type (called an MBF) can be any bitwise-OR combination of the following constants:

mbfNil **mbfVars** **mbfFuncs** **mbfMacros**
mbfTypes **mbfClass** **mbfAll**

The instance information output provided by DumpInst has the following format:

<ascii instance name> <ascii type name> <ascii attribute names>

2.3.5 Definitions and References (IDEF and IREF Variables)

Each instance in the database has an associated set of definitions and references, which also are tracked using index variables (IDEF and IREF). The two functions below return the range of valid definition and reference index values for a given instance.

```
VOID BSC_API DefRangeOfInst (IINST iinst,
                             IDEF far *pidefMin,
                             IDEF far *pidefMac)

VOID BSC_API RefRangeOfInst (IINST iinst,
                             IREF far *pirefMin,
                             IREF far *pirefMac)
```

The first of these functions returns index limits in two IDEF variables; the second returns the limits in IREF variables. From a definition or reference index, the actual filename and line number of a particular instance reference or definition can be determined using one of the following functions:

```
VOID BSC_API DefInfo (IDEF idef,
                      LSZ far *plszName,
                      WORD far *pline)

VOID BSC_API RefInfo (IDEF iref,
                      LSZ far *plszName,
                      WORD far *pline)
```

NOTE: Study the prototype for these two functions carefully. The LSZ argument is handled in an atypical way. Instead of placing text in a buffer that the caller provides, the library actually

fills in a pointer to one of its internal buffers where the filename can be found. The LSZ pointer is set to point to this buffer. If you intend to use the contents of this buffer after making another call to the library, be sure to copy these contents, or they may be lost in the next library call.

2.3.6 Uses/Used-by Information (IUSE and IUBY Variables)

Each symbol in the database has an associated set of instances that use the given symbol or that the symbol uses. These "uses" and "used-by" instances have associated index variables called IUSE and IUBY, respectively. For a particular instance, these uses/used-by instances can be accessed by first determining the range of valid uses/used-by index values via one of the following functions:

```
VOID BSC_API UseRangeOfInst (IINST iinst,
                             IUSE far *piuseMin,
                             IUSE far *piuseMac)

VOID BSC_API UbyRangeOfInst  (IINST iinst,
                              IUBY far *piubyMin,
                              IUBY far *piubyMac)
```

From a "uses" or "used-by" instance index, the corresponding instance that uses or is used by the queried instance can be found, as can the number of times such use occurs.

```
VOID BSC_API  UseInfo (IUSE iuse,
                       IINST far *piinst,
                       WORD far *pcnt)

VOID BSC_API UbyInfo (IUBY iuby,
                      IINST far *piinst,
                      WORD far *pcnt)
```

After a call to one of the above functions, piinst will point to the instance index (IINST) value for the instance to which the IUSE or IUBY variable refers. The pcnt variable will point to the number of times the respective use occurs.

2.3.7 Base and Derived Classes (IBASE and IDERV Variables)

Some instances in the browser database could refer to classes that they are based on or from which they are derived. In similar form to other index-range functions, two library functions provide the valid base class or derived class index range:

```
VOID BSC_API  BaseRangeOfInst (IINST iinst,
                               IBASE far *pibaseMin,
                               IBASE far *pibaseMac)

VOID BSC_API DervRangeOfInst (IINST iinst,
                              IDERV far *pidervMin,
                              IDERV far *pidervMac)
```

If an instance does not represent a class, then the Min/Mac pointers will point to the same value.

Given an index to a base, you can obtain information about the base or derived classes using the following functions:

```
VOID BSC_API BaseInfo(IBASE ibase,
                      IINST far *piinst,
                      WORD far *pmode)

VOID BSC_API DervInfo(IDERV iderv,
                      IINST far *piinst,
                      WORD far *pmode)
```

For a given IBASE or IDERV index, these functions return the instance for the base/derived class and the inheritance mode.

2.3.8 Friend (Incoming/Outgoing) Classes (IFRIN and IFROUT Variables)

Assume that your code has a class named A. An incoming friend of class A is a class that *gives* class A friend access. An outgoing friend of class A is a class that *obtains* friend access from class A. Note that it is possible for a function to have incoming friends but no outgoing friends (that is, a function can be given friend access, but the function does not give such access). For each type of friend, index variables (either IFRIN or IFROUT) track all friend instances. For a particular instance, there may be several other instances that give or obtain friend access to or from other instances. Given an instance index, the following functions return the valid range of friend indexes for that instance:

```
VOID BSC_API FrinRangeOfInst(IINST iinst,
                             IFRIN far *pifrinMin,
                             IFRIN far *pifrinMac)

VOID BSC_API FroutRangeOfInst(IINST iinst,
                              IFROUT far *pifroutMin,
                              IFROUT far *pifroutMac)
```

Given a single friend index (either incoming or outgoing), the following functions return the instance of the corresponding friend:

```
VOID BSC_API FrinInfo(IFRIN ifrin,
                      IINST far *piinst)

VOID BSC_API FroutInfo(IFROUT ifrout,
                       IINST far *piinst)
```

2.4 Generating Calltrees and Classtrees

A very important feature of the browser database is the ability to generate calltrees from any function and classtrees from any C++ class instance or name. These trees may be generated in both forward and backward directions from the given instance index or symbol name. The following functions generate the calltree or classtree from either an instance index or a symbol name, respectively:

Calltrees:

```
VOID BSC_API CallTreeInst (IINST iinst)
```

```
VOID BSC_API RevTreeInst (IINST iinst)

VOID BSC_API FCallTreeLsz (LSZ lszName)
VOID BSC_API FRevTreeLsz (LSZ lszName)
```

Classtrees:

```
VOID BSC_API ClassTreeInst (IINST iinst)
VOID BSC_API RevClassTreeInst (IINST iinst)

VOID BSC_API FRevClassTreeLsz (LSZ lszname)
VOID BSC_API FClassTreeLsz (LSZ lszname)
```

For calltrees, the symbol name may be either the name of a function or the name of a module. In the latter case, the root of the tree is the module name; the child subtrees are the functions contained in that module. For classtrees, the symbol name may be the name of a function, variable, or class.

The result is ASCII text output to the standard output. Indentations denote different levels of the tree. If any node is visited more than once, due to cycles in the call graph, then the node's name will be succeeded by an ellipsis (...). If the source script of the function is not known, then it will be followed by a question mark (?).

The following is sample source code that will be used to demonstrate the output of these and, in subsequent sections, other functions.

```
#include <stdio.h>
void one (void);
void two (void);
void three (void);
void five (void);

void one()
{
   printf("One, ");
}

void two()
{
   printf("Two, ");
   three();
   five();
}

void three()
{
   printf("Three, ");
}

void five()
{
   printf("Five\n");
}

main()
{
```

```
    printf("This program prints the first ");
    printf("five Fibonacci numbers . . .\n");
    one();
    one();
    two();
}
```

2.4.1 Sample Calltrees

The following are five separate forward calltrees for the symbols main, one, two, three, and five from the source script listed above:

```
main                    one       two                 three       five
|__printf[2]?   |__printf? |__printf?    |__printf?  |__printf?
|__one[2]                                |__three
| |__printf?                  | |__printf?
|__two                            |__five
  |__printf?                    |__printf?
  |__three
  | |__printf?
  |__five
    |__printf?
```

The two reverse calltrees listed below are for the symbols printf and five from the sample source:

```
printf                      five
|__one                      |__two
| |__main[2]                  |__main
|__two
| |__main...
|__three
| |__two...
|__five
| |__two...
|__main[2]...
```

2.4.2 Sample Classtrees

The tree on the left is a forward classtree for the symbol CWnd from the HELLOAPP program provided with Visual C++ (see the MFC SAMPLES directory). The tree on the right is a reverse classtree for the symbol CHelloWindow from the same program.

```
 CWnd                          CHelloWindow
 |__CDialog                    |__CFrameWnd
 | |__CModalDialog         |__CWnd
 |__CStatic                      |__CCmdTarget
 |__CButton                        |__CObject
 | |__CBitmapButton                  |__CObjectRoot
 |__CListBox
 |__CComboBox
 |__CEdit
 |__CScrollBar
 |__CFrameWnd
```

```
| |__CMDIFrameWnd
| |__CMDIChildWnd
| |__CHelloWindow
|__CView
   |__CScrollView
```

2.5 Listing Database References

The following function, ListRefs, lists references of all symbols in the browser database which meet the requirements set by the filter mask mbfReqd. These symbols will be dumped to the standard output. If the function is unable to complete its task, then the return value is FALSE; otherwise it is TRUE.

```
BOOL BSC_API ListRefs (MBF mbfReqd)
```

The filter mask can be set using the bitwise OR operator, as in

```
ListRefs (mbfFuncs | mbfMacros | mbfTypes)
```

which produces the following output for the sample script listed above:

```
FUNCTION        CALLED BY LIST

five:           two
main:
one:            main[2]
printf:  one    two
                three    five
                main[2]
three:  two
two:            main

MACRO           USED BY LIST

_MSC_VER:

TYPE            USED BY LIST

_iobuf:
FILE:
fpos_t:
size_t:
va_list:
```

2.6 File Outline Functions

The following function outputs an outline of a module:

```
VOID BSC_API OutlineMod (IMOD imod, MBF mbf)
```

Given a module symbol index imod and a filter mask mbf, this function sends to the standard output all of the symbols in that module with attributes and type that match the required filter. For example, a call of this fuction

OutlineMod (imod, mbfVars | mbfFuncs)

produces the following output:

```
fib.c
  five     (function:public)
  main     (function:public)
  one              (function:public)
  three    (function:public)
  two              (function:public)
```

The following function outputs the outline for the module(s) matching a name or pattern:

BOOL BSC_API FOutlineModuleLsz (LSZ lszName, MBF mbf)

This function performs similarly to OutLineMod, except the zero-terminated string lszName may contain the pattern-matching wildcards * and ?.

2.7 String Processing Functions

Table 8 describes the operation of string processing functions.

Table 8: String Processing Functions

Action	Prototype	Result
Simple formatter	VOID BSC_API BSCFormat (LPCH lpchOut, LSZ lszFormat, va_list va)	Format from the string lszFormat to the string lpchOut as specified by the standard switch (for example, %s).
Simple printf replacement	VOID BSC_API BSCPrintf (LSZ lszFormat, ...)	Takes one or more zero-terminated strings and outputs them to the standard output. Only two switches are supported, %d and %s.
Simple sprintf replacement	VOID BSC_API BSCSprintf(LPCH lpchOut, LSZ lszFormat, ...)	See BSCPrintf above.
Pattern-matching function	BOOL BSC_API FWildMatch (LSZ lszPat, LSZ lszText)	Returns TRUE if the pattern string lszPat matches the string lszText. The standard wildcards (* and ?) may be used in the pattern string. Matching occurs just as MS-DOS matches filenames.
Compares strings	INT BSC_API CaseCmp (LSZ lszl, LSZ lsz2)	Performs a case-(in)sensitive comparison, depending on the current case sensitivity, which can be set by SetCaseBSC (see Table 9). The comparison occurs in the order of symbols listed in the symbol table.

Compares prefixes of two strings	int BSC_API CaseCmpPrefix (LSZ lszprefix, LSZ lszdecor)	Nearly identical to CaseCmp except that only the prefixes of two strings are compared.

Action	Prototype	Result
Returns the base name of a path	LSZ BSC_API LszBaseName (LSZ lsz)	Given a path name lsz, the base name is extracted and returned. If an empty string is passed to the function, then the empty string is returned.
Obtains undecorated names	LSZ BSC_API FormatDname (LSZ lszname)	Returns the undecorated name for the function name indicated by lszname.
Returns the base name of a decorated name.	LSZ BSC_API LszBaseOfDname (LSZ lszname)	Returns the full name if lszname is undecorated, the member name if lszname is a member variable or member function, the function or variable name if lszname is just a decorated function or data item (global), or the class name if lszname is a constructor or destructor. All other inputs return NULL.
Returns the class name from a decorated name.	LSZ BSC_API LszClassOfDname (LSZ lszname)	Returns the class name from a decorated name or NULL if lszname contains no class name. If lszname is either not decorated or if the class name part of a decorated lszname is empty (as in a global operator), then the pointer return is NULL.

2.8 Miscellaneous Functions

Table 9 describes other useful browser functions:

Table 9: Miscellaneous Functions

Purpose	Prototype	Result
Assists formatting of output from database queries	WORD BSC_API BSCMaxSymLen (VOID)	Returns the length of the largest symbol in the browser database.
Determines case sensitivity	BOOL BSC_API FCaseBSC (VOID)	Returns TRUE if the database is built with case-sensitive language; returns FALSE otherwise.
Sets the case sensitivity	VOID BSC_API SetCaseBSC (BOOL fCase-Sensitive)	Overrides the case sensitivity of the database. Look-ups in the symbol table become case (in)sensitive as specified by fCaseSensitive.
Dumps database	VOID BSC_API StatsBSC()	Outputs statistics to the standard

statistics		output using BSCOutput.

As an example of the output from StatsBSC, consider the simple "Hello world!" script:

```
#include <stdio.h>
main()
{
        printf("Hello world!");
}
```

For this script, StatsBSC produces the following:

```
Totals
MOD             : 3
MODSYM          : 13
SYM             : 13
INST            : 13
REF     : 55
DEF     : 13
USE     : 6
UBY             : 6

Detail
<Unknown> Modsyms:2
C:\WINDEV\INCLUDE\stdio.h Modsyms:10
hello.c Modsyms:1
_base       (mem_var:public) DEF 1 REF 0 USE 0 UBY 1
_cnt        (mem_var:public) DEF 1 REF 0 USE 0 UBY 1
_file       (mem_var:public) DEF 1 REF 0 USE 0 UBY 1
_flag       (mem_var:public) DEF 1 REF 0 USE 0 UBY 1
_iobuf      (struct_name) DEF 1 REF 1 USE 5 UBY 0
_MSC_VER    (constant) DEF 1 REF 1 USE 0 UBY 0
_ptr        (mem_var:public) DEF 1 REF 0 USE 0 UBY 1
FILE        (typedef) DEF 1 REF 40 USE 0 UBY 0
fpos_t      (typedef) DEF 1 REF 2 USE 0 UBY 0
main        (function:public) DEF 1 REF 0 USE 1 UBY 0
printf      (function:decl_only:public) DEF 1 REF 2 USE 0 UBY 1
size_t      (typedef) DEF 1 REF 5 USE 0 UBY 0
va_list     (typedef) DEF 1 REF 4 USE 0 UBY 0
```

3 Browser Objects

The following sections describe browser objects (BOBs).

3.1 Creating and Manipulating BOB Variables

The browser library provides several high-level functions that help the client query the information stored in the database. Examples of such queries include:

- Information about the names of all modules in the database
- Functions an instance might call
- Occurrences of a symbol definition
- Friend access relationships

All the queries process one item called a browser object (BOB). A browser object is actually one of the index values that are encoded along with the type of index into a 32-bit quantity. Thus, once it is known, the type or "class" of the browser object can be extracted, as can its appropriate index value (see below).

The ClsOfBob() macro returns one of the following CLS (class) types for the given browser object:

clsMod	clsInst	clsSym	clsRef
clsDef	clsUse	clsUby	clsBase
clsDerv	clsFrin	clsFrout	

A browser object can be created from either a name of a symbol known to the database or from an index which is valid with respect to the database.

The following call creates a browser object from the name of the object:

```
BOB BSC_API BobFrName (LSZ lsz)
```

If no browser object can be created from the given name, the value bobNil is returned. If the name is found, then the browser object returned will be either of class clsMod (if the name was a module name) or of class clsInst (if the name was a symbol name). If more than one instance is associated with the given name, the browser object's value will be the first (smallest) such instance.

The following macros can create a browser object from one of the index values. Listed beside each macro is the macro that performs the reverse operation.

BobFrMod (IMOD x)	ImodFrBob (BOB b)
BobFrSym (ISYM x)	IsymFrBob (BOB b)
BobFrInst (IINST x)	IinstFrBob (BOB b)
BobFrRef (IREF x)	IrefFrBob (BOB b)
BobFrDef (IDEF x)	IdefFrBob (BOB b)
BobFrUse (IUSE x)	IuseFrBob (BOB b)
BobFrUby (IUBY x)	IubyFrBob (BOB b)
BobFrBase (IBASE x)	IbaseFrBob (BOB b)
BobFrDerv (IDERV x)	IdervFrBob (BOB b)
BobFrFrin (IFRIN x)	IfrinFrBob (BOB b)
BobFrFrout (IFROUT x)	IfroutFrBob (BOB b)

All of the macros on the left use the function BobFrClsIdx, which, given one of the indexes shown above, returns the appropriate BOB.

NOTE: The macros that convert an index into a browser object create a BOB whose class type corresponds to the type of index given. But the macros that convert a browser object back to an index do not check to make sure that the browser object has the correct class type value to do the conversion. It is up to the user of these macros to do whatever checking may be required before using them.

To retrieve the symbol name of a browser object, use the function

```
LSZ BSC_API LszNameFrBob  ( BOB bob )
```

which returns either a symbol name or a module name, depending on the class type of the browser object.

The browser library also has a function that retrieves a range of symbol indexes given only a prefix. To obtain this range, use the function

```
BOOL BSC_API FindPrefixRange ( LSZ lszprefix,
                               ISYM far *pfirstisym,
                               ISYM far *plastisym )
```

which returns TRUE if the search was successful, or FALSE otherwise. Matching for this function always is case insensitive.

3.2 Performing Database Queries

You can initiate several predefined queries once a browser object has been created. These are passed to the function InitBSCQuery along with the browser object itself.

```
BOOL BSC_API InitBSCQuery (QY qy, BOB bob)
```

The function returns TRUE if the query has been initiated successfully, FALSE otherwise. Not all queries can be performed on all browser objects (for example, trying to perform the query "What does this browser object call?" when the browser object type is actually that of a module index). The possible queries and their permitted browser object types are outlined in Table 10.

Table 10: Database Queries and BOB Types

Query	Input BOB Type	Return BOB Type	Query initiated for
qyFiles	(ignored)	module	Modules that are defined in the database.
qySymbols	(ignored)	instance	Instances of symbols that occur in the database.
qyContains	module	instance	Symbols that are defined in that module.
qyCalls	instance	instance	Instances that a function (instance) calls. If the instance in that BOB is not of the type function, then the list of instances returned will be of zero length.
qyUses	instance	instance	Instances that an instance uses.

qyCalledBy	instance	instance	Instances that call the input instance. These instances will be of the type function.
qyUsedBy	instance	instance	Instances that use the input instance.
qyUsedIn	instance	module	Module(s) that use/reference the instance.
Query	**Input BOB Type**	**Return BOB Type**	**Query initiated for**
qyDefinedIn	instance	module	Module(s) that define the instance.
qyRefs	instance, symbol	reference	References to all instances of the symbol, or, if the BOB is an instance, then the references to that particular instance itself.
qyDefs	instance, symbol	definition	Definitions of all instances of a symbol, or, if the BOB is an instance, then the definitions to that particular instance itself.
qyBaseOf	instance, class	instance, class	Instances that have the given symbol as their base class.
qyDervOf	instance, class	instance, class	Instances that are derived from the symbol.
qyMembers	instance, class	instance, mem_var, mem_funct	All members of the given class including bases.
qyMemberOf	instance, mem_var, mem_func	instance, class	All classes of which the given symbol is a member.
qyImpMembers	instance, class	instance, mem_var, mem_funct	All members implemented by the given class.
qyFriendIn	instance, class	instance, class	All classes that give friend access to the given class.
qyFriendOut	instance, class	instance, class	All classes that obtain friend access from the given class.
qyBaseSorted, qyBaseIncSorted, qyDervSorted, qyDervIncSorted	--	--	Reserved for internal use.

The result of a query is a collection of browser objects. To obtain the next browser object in the query information, use the following function:

BOB BSC_API BobNext (VOID)

Any one of the browser objects returned may be passed back to initiate another query. The name of the symbol for that browser object may be found using the function LszNameFrBob, which can also be used to obtain the next browser object in the collection. If there are no more browser objects left in the collection, the value bobNil is returned.

3.3 Generating Name Overloads

The GenerateOverloads and GenerateOverloadsEx functions find all possible browser objects that match a given name, such as all members of a given class, all class members having the given name, or all top-level items having the given name. These functions then pass each BOB to a given function (pointed to by PFN_BOB in GenerateOverloads and PFN_BOB_EX in GenerateOverloadsEx).

```
WORD BSC_API GenerateOverloads (LSZ, MBF, PFN_BOB)
WORD BSC_API GenerateOverloadsEx (LSZ, MBF, PFN_BOB_EX)
```

The following are the forms LSZ can have and their subsequent action:

class::	Generates a list of all members of the class.
class::mem	Generates a list of all members named mem. GenerateOverloadsEx also accepts wildcards, as in mem*, and nested classes (such as classA::classB::mem).
mem	Generates a list of all members mem in any class or top-level items.

The member mem can be a regular member or it can be operator+, operator new, and so forth.

GenerateOverloadsEx stops executing (returns) whenever the function pointed to by PFN_BOB_EX returns FALSE, whereas GenerateOverloads continues executing. This feature of GenerateOverloadsEx can be used, for example, to show the first n overloads in a list box without waiting for all the overloads to be computed.

4 Browser Library Index

This index lists all of the browser library functions, variables, and constants discussed in this reference document and also defined or declared in the header files BSC.H and BSCSUP.H, both of which provide the browser library interface. Each entry is followed by the section and/or table in which that particular item is discussed.

L

M

O

P

Q

R

S

U