Visual C++ Browser Toolkit for Windows NT Tutorial

Microsoft Visual C++ Browser Toolkit for Windows NT

This tutorial illustrates the use of the browser library (BSC.LIB) included in the Microsoft Visual C++ Browser Toolkit for Windows NT. Specifically, the tutorial describes the Toolkit's sample application, BSCDUMP, which calls many of the browser library's functions. You may want to print a copy of BSCDUMP.C for easy reference.

Contents

1 Overview of BSCDUMP

The sample application BSCDUMP included with the Browser Toolkit demonstrates how you can use the Toolkit's browser library to access the contents of browser database files, which have the extension .BSC. The database contains all symbols defined in the program's source and include files, the location of these symbols, references to each, and linkage information. These symbols represent the names of all functions, types (such as the names of all classes, structures, enumerations, and their members), macros, and variables in the files. Visual C++ stores a program's function names as decorated names, which indicate the class to which the function belongs, parameter types, calling convention, and return type. The database also contains such information as the calltree of a function, the symbol table, the definition table, and the reference table. You can query each of these tables using the browser APIs to obtain information about program symbols.

BSCDUMP is a program that displays the contents of .BSC files. The program accomplishes most of its tasks by calling browser library functions, which perform all the information retrieval work. The include files, BSCSUP.H and BSC.H, provide the necessary interface to all the browser library's functions. The declarations in each of these header files follow the specifications explained in the BSCREF.DOC file. You should note the special use of the browser library "callback" functions since they can enhance environment integration.

A callback function is one that performs a particular system service required by the browser library (such as memory allocation, file I/O, or error-handling). Programs that use BSC.LIB can still use functions from the C run-time library, however callback functions can be used to provide an interface to any environment that might not allow access to the C run-time library. See BSCREF.DOC for a more detailed discussion of the browser library's callback functions and their specifications.

The Browser Toolkit includes the batch file MAKE.BAT that instructs the Microsoft Visual C++ for Windows NT compiler to create the BSCDUMP executable.

Some of the general operations BSCDUMP performs include listing file references, redundant definitions, and symbol definitions and references. The program also displays symbol calltrees and classtrees, BSC file summary and statistics, and raw symbols. The following sections provide more specific information about these operations and how they are accomplished.

2 BSCDUMP Arguments

BSCDUMP has the following usage syntax:

```
BSCDUMP [options] <file.bsc>
```

Table 1 lists the available options and their respective actions. Note that if no option is selected, BSCDUMP produces an ASCII dump of the .BSC file. The table also indicates the function that is called to produce the desired action. Function names preceded by an asterisk refer to functions defined in the BSCDUMP.C file itself. All other functions are defined in the browser library. This tutorial describes each of these options in the order listed in Table 1.

Table 1: BSCDUMP Options

Argument	**Action**	**Function Called**
-l[FCVMT]	List references	**ListRefs**
-u[FCVMT]	List unused definitions	***ListUnused**
-O <file>	Outline functions with source comments	***DumpFunction-Comments**
-s	List BSC file summary/statistics	**StatsBSC**
-S	List raw symbols	***ListRawSyms**
-o[FCVMT] <symbol>	List outline for given symbol (module)	**FOutlineModuleLsz**
-i[FCVMT] <symbol>	List information about given symbol	**GenerateOverloads**
-r <symbol>	List references to symbol (case sensitive)	***DumpDefRefLsz**
-d <symbol>	List definitions of symbol (case sensitive)	***DumpDefRefLsz**
-R <symbol>	List references to symbol (case insensitive)	***DumpDefRefLsz**

-D <symbol>	List definitions of symbol (case insensitive)	***DumpDefRefLsz**
-t <symbol>	Provide calltree/classtree	**GenerateOverloads**
-b <symbol>	Provide backward calltree/classtree	**GenerateOverloads**
no argument	Dump entire BSC file contents	**DumpBSC**

The additional options [FCVMT] available for some arguments narrow the inquiry to the following specific symbols:

Argument	**Symbol Information**
F	Functions
C	Classes
V	Variables
M	Macros
T	Types

If none of these suboptions is selected explicitly, then they are all selected by default. The following sections describe each of the BSCDUMP options in more detail.

2.1 General List Commands

For each of the options selected that also take suboptions, BSCDUMP creates a filter mask (called an MBF for historical reasons) representing these suboptions. If no suboptions are selected, this mask is set to all options.

Refer to BSCREF.DOC for a more detailed discussion of browser library functions referred to in BSCDUMP.C.

-l[FCVMT] (List References)

The program simply passes the suboption filter mask to the browser library function ListRefs, which accomplishes the task of listing all symbol references.

-u[FCVMT] (List Unused Definitions)

BSCDUMP depends on the function ListUnused to list unused definitions. This is not a browser library function and therefore is a good example of one way to extend the library's functionality. Here is the function listing:

```
void ListUnused(MBF mbf)
{
    ISYM isym, isymMac;
    IINST iinst, iinstMac;
    IUBY iubyFirst, iubyLast;

    isymMac = IsymMac();
    for (isym = 0 ; isym < isymMac ; isym++) {
            InstRangeOfSym(isym, &iinst, &iinstMac);
            for ( ; iinst < iinstMac ; iinst++) {
                if (!FInstFilter(iinst, mbf))
                        continue;
                UbyRangeOfInst(iinst, &iubyFirst, &iubyLast);
                if (iubyFirst == iubyLast) {
                        DumpInst(iinst);
                        BSCPrintf("\n");
                }
            }
    }
}
```

Initially, ListUnused calls a browser library function, IsymMac, that returns the biggest symbol index in the browser database. This number provides the upper bound in a search for redundant symbol definitions. The following for loop allows the remaining part of the function to query each symbol individually with the range of isym = 0 to isym = isymMac - 1. The function then queries another library function, InstRangeofSym, to determine the range of instances associated with each symbol in the symbol table. This function stores the lower and upper bounds in iinst and iinstMac, respectively. A call to FInstFilter determines if the current symbol matches the one requested. If it does not, then the function proceeds with the next iteration of the for loop.

Using the two bounds for the symbols, ListUnused calls on the library function UbyRangeOfInst, which returns the associated range called by the table. The function then determines if the upper and lower bounds of this range are identical. If the current symbol is redundant, ListUnused calls the function DumpInst, which sends the following information to the standard output regardless of symbol instance type:

```
<ASCII instance name> <ASCII type name> <ASCII attribute name>
```

Finally, another library function, BSCPrintf, produces the new line on the standard output via the BSCOutput callback function.

-O <file> (Outline functions with source comments)

The BSCDUMP function DumpFunctionComments displays a list of all functions and associated comments in the given source file(s). Wildcards can be used in the filename given to this option.

To display the source file comments, DumpFunctionComments calls another BSCDUMP.C function, DumpComments. This function uses a heuristic rule to search for comment blocks before functions in the source. If no comments can be found, only the function name is displayed.

The following is the output of bscdump -O bscdump.c bscdump.bsc:

```
Browser Data Base: bscdump.bsc
```

```
[[[[[[[[[[[[ c:\browser\bscdump.c ]]]]]]]]]]]]

## DumpBobInfo  (function:public)
{
// dump the instance info from a BOB, the BOB must be of clsInst
//
}

## DumpComments (function:public)
{
// having opened a particular file, we are now looking for comments at
// or before the given line number
//
}

## DumpDefRef   (function:public)
{
// dump definitions or references for a single BOB, suitable for use
// in GenerateOverloads
//
}

## DumpDefRefLsz   (function:public)
{
// dump definition or reference starting from possibly overloaded symbol name
//
}

## DumpFunctionComments (function:public)
{
// show any comment headers that are associated with functions
// in the given module.  The module may include wildcards.
//
}

## DumpFwdTree  (function:public)
{
// dump a forward tree, either class tree or call tree depending on
// the type of BOB that we get.  The BOB must be an instance BOB.
//
}

## DumpRevTree  (function:public)
{
// dump a reverse tree, either class tree or call tree depending on
// the type of BOB that we get.  The BOB must be an instance BOB.
//
}

## GenericError  (function:public)
{
// non-specific error message about a given symbol
//
}

## ListRawSyms   (function:public)
{
// list the raw text of all the symbols in the database, useful
// for grepping.  Note sorting order is subject to change so
// use a comparison API to compare...
//
}

## ListUnused    (function:public)
{
// list unused definitions, these are items which are not used by some other
// symbol known to the browser.  Note, entry points like "main" should appear
// in the list.  This output is helpful in locating dead code.
//
}

## main         (function:public)
```

```
{
// main entry point, parse args and dispatch workers...
//
}

## MbfFromLsz   (function:public)
{
// get a mask of object types (nobody knows how this ended up getting
// called an MBF) from a string...
//
}

## ParseArgs    (function:public)
{
// parse arguments into global variables and open a database
//
}

## Usage        (function:public)
{
// emit usage information
//
}
```

-s (List Browser File Summary/Statistics)

The browser library function StatsBSC performs all the necessary work required to list statistics and a summary of a browser database. The statistics include the table sizes for the following tables and lists:

- Module Table
- Symbols listed by module name
- Symbol list
- Properties list
- Definition table
- Reference table
- Call table
- Called by table

In addition, the StatsBSC function outputs each symbol and the associated instance information counts.

-S (List Raw Symbols)

To list raw symbols in a browser database, BSCDUMP calls its function ListRawSyms, shown below. This function calls the library function IsymMac, which returns the biggest symbol index in the database. BSCDUMP then prints all symbols with index smaller than this largest index.

```
void ListRawSyms()
{
   ISYM isym, isymMac;
   isymMac = IsymMac();

   for (isym = 0 ; isym < isymMac ; isym++)
           BSCPrintf("%s\n", LszNameFrSym(isym));
}
```

2.2 Symbol-Specific List Commands

-o[FCVMT] <symbol> (List Outlines)

The -o option produces a list of all symbols in the module with attribute types indicated by the suboptions (F, V, M, C, or T). With the exception of argument parsing, all of the work required to produce this list is done by the library function FOutlineModuleLsz. Here is the actual call to this function:

```
FOutlineModuleLsz(psymbol, mbf)
```

-i [FCVMT] <symbol> (List Symbol Information)

This option causes BSCDUMP to call the GenerateOverloads library function, which takes three arguments: a symbol, a symbol mask, and a pointer to a function. GenerateOverloads returns all possible browser objects (BOBs) that match a given overloaded name. GenerateOverloads then applies whatever function it was given to each of the BOBs found, making GenerateOverloads a very powerful library function. In this case, the function passed to GenerateOverloads is DumpBobInfo, which dumps a single instance of a browser object.

```
void BSC_API DumpBobInfo(BOB bob)
{
   if (ClsOfBob(bob) != clsInst)
          return;
   DumpInst(IinstFrBob(bob));
   BSCPrintf("\n");
}
```

-r <symbol> and -d <symbol> (List References and Definitions to Symbol; Case Sensitive)

The -r and -d options allow the user to list each reference to and definition of, respectively, a given symbol. Its implementation again demonstrates the power of GenerateOverloads. BSCDUMP uses its DumpDefRef function and GenerateOverloads, as shown below, to generate references and definitions.

```
void DumpDefRefLsz(LSZ lszSym)
{
          if (!GenerateOverloads(lszSym, mbfAll, DumpDefRef))
             GenericError(lszSym);
}
```

-R <symbol> and -D <symbol> (List References and Definitions to Symbol; Case Insensitive)

These two options are identical to their respective -r and -d counterparts, with the exception that the database queries are case insensitive. In fact, they both result in the same function, DumpDefRefLsz, being called. BSCDUMP achieves this dual-case functionality by calling the SetCaseBSC library function, which overrides the browser database case sensitivity. The argument to SetCaseBSC is a Boolean which determines if sensitivity is on (TRUE) or off (FALSE).

-t <symbol> (Provide Calltree/Classtree)

This option provides the calltree or classtree from a symbol name, which may be the name of either a function or a module. BSCDUMP calls the GenerateOverloads library function with a pointer to BSCDUMP's DumpFwdTree function. Depending on the symbol mask filter, the latter determines the type of tree requested (call or class) and calls one of the two library functions CallTreeInst or ClassTreeInst.

-b <symbol> (Provide Backward Calltree/Classtree)

For a backward calltree or classtree, BSCDUMP also calls GenerateOverloads, but with a pointer to BSCDUMP's DumpRevTree function instead. After determining which type of tree is requested, BSCDUMP calls one of two library functions, RevTreeInst or RevClassTreeInst.