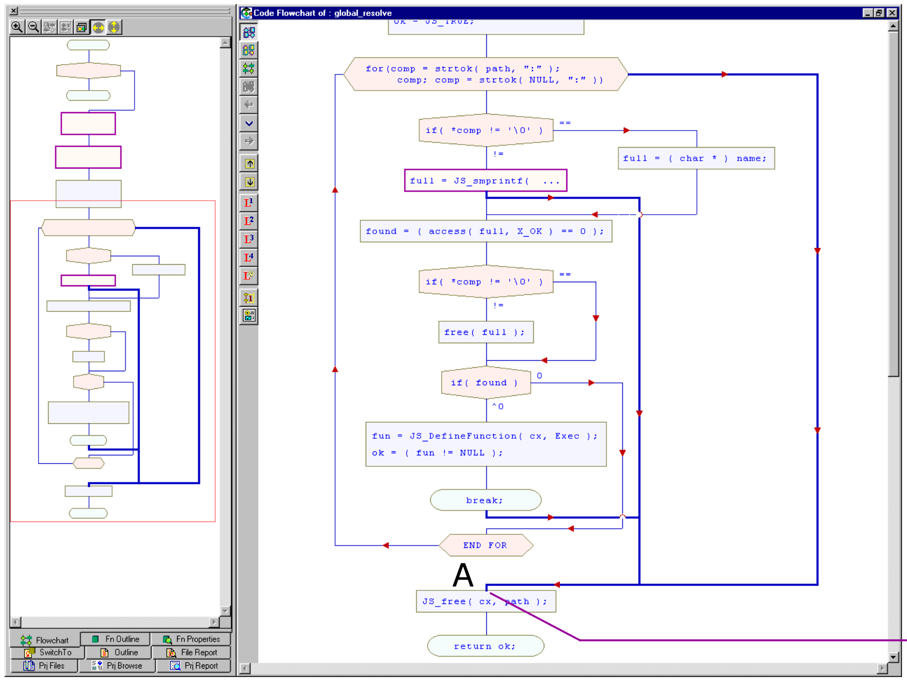
Understand Any Function in less time



← With the flowchart, you can understand this 50-line function in half the time.

How do you find all paths that go to point A in the function?

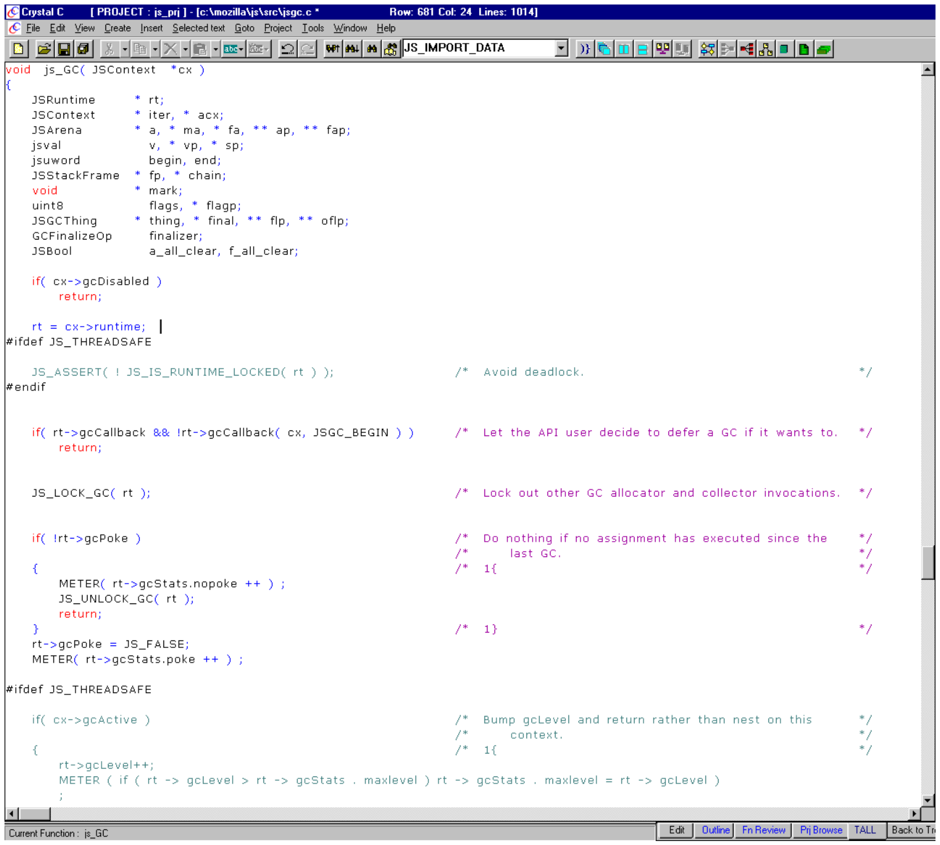
Answer:
Click on point A in the flowchart.

Crystal C/C++ automatically creates the flowchart of any function.

If you do not have flowcharts,

- how do you find all such paths?
- or examine them during code-review?

How to Attack a Very Long Function


Crystal C [PROJECT : js_prj] - [c:\mozilla\js\src\jsgc.c * Row: 681 Col: 24 Lines: 1014]
File Edit View Create Insert Selected text Goto Project Tools Window Help
JS_IMPORT_DATA
Current Function : js_GC
Edit
Outline
Fn Review
Prj Browse
TALL
Back to Tr

```
void  js_GC( JSContext  *cx )
{
    JSRuntime       * rt;
    JSContext       * iter, * acx;
    JSArena         * a, * ma, * fa, ** ap, ** fap;
    jsval             v, * vp, * sp;
    jsuword           begin, end;
    JSStackFrame    * fp, * chain;
    void            * mark;
    uint8             flags, * flagp;
    JSGCThing       * thing, * final, ** flp, ** oflp;
    GCFinalizeOp      finalizer;
    JSBool            a_all_clear, f_all_clear;

    if( cx->gcDisabled )
        return;

    rt = cx->runtime;
#ifdef JS_THREADSAFE

    JS_ASSERT( ! JS_IS_RUNTIME_LOCKED( rt ) );                /*  Avoid deadlock.                                       */
#endif


    if( rt->gcCallback && !rt->gcCallback( cx, JSGC_BEGIN ) )      /*  Let the API user decide to defer a GC if it wants to.   */
        return;


    JS_LOCK_GC( rt );                                         /*  Lock out other GC allocator and collector invocations.  */


    if( !rt->gcPoke )                                         /*  Do nothing if no assignment has executed since the      */
                                                              /*      last GC.                                            */
    {                                                         /*  1{                                                      */
        METER( rt->gcStats.nopoke ++ ) ;
        JS_UNLOCK_GC( rt );
        return;
    }                                                         /*  1}                                                      */
    rt->gcPoke = JS_FALSE;
    METER( rt->gcStats.poke ++ ) ;

#ifdef JS_THREADSAFE

    if( cx->gcActive )                                        /*  Bump gcLevel and return rather than nest on this        */
                                                              /*      context.                                            */
    {                                                         /*  1{                                                      */
        rt->gcLevel++;
        METER ( if ( rt -> gcLevel > rt -> gcStats . maxlevel ) rt -> gcStats . maxlevel = rt -> gcLevel )
        ;
```

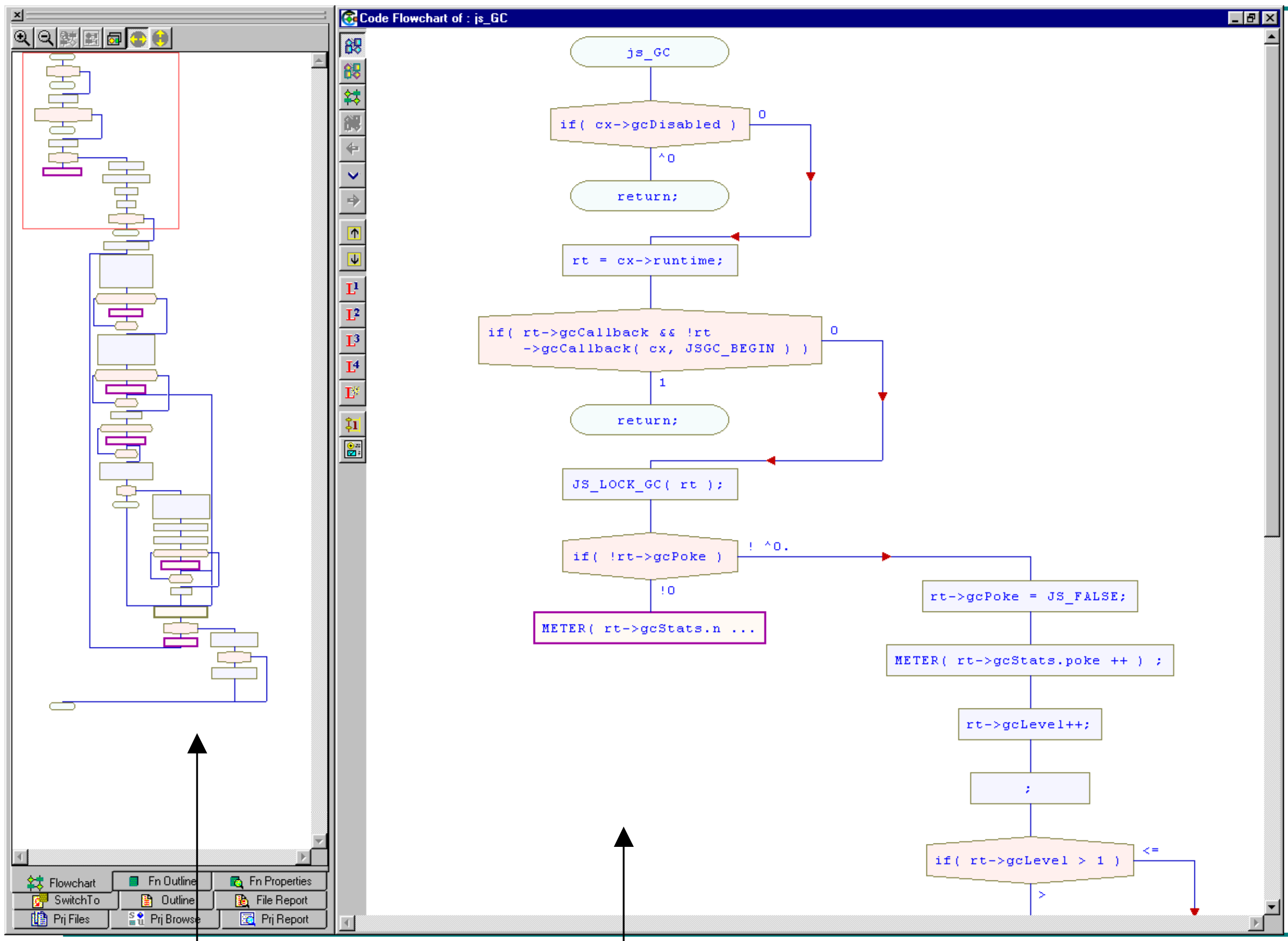
← This is js_GC , a 350-line function from Mozilla source code. (seven pages of your window)

How would you review and understand this function's code?

Answer: Click the flowchart icon .

Understand the Top-Level Logic

in less than 5 minutes



For large functions, Crystal C/C++ automatically creates a top-level flowchart.

← Top-level flowchart of the function js_GC.

You will need less than five minutes to go through this flowchart.

If you do not have flowcharts, how do you construct the top-level view of a long function*?*

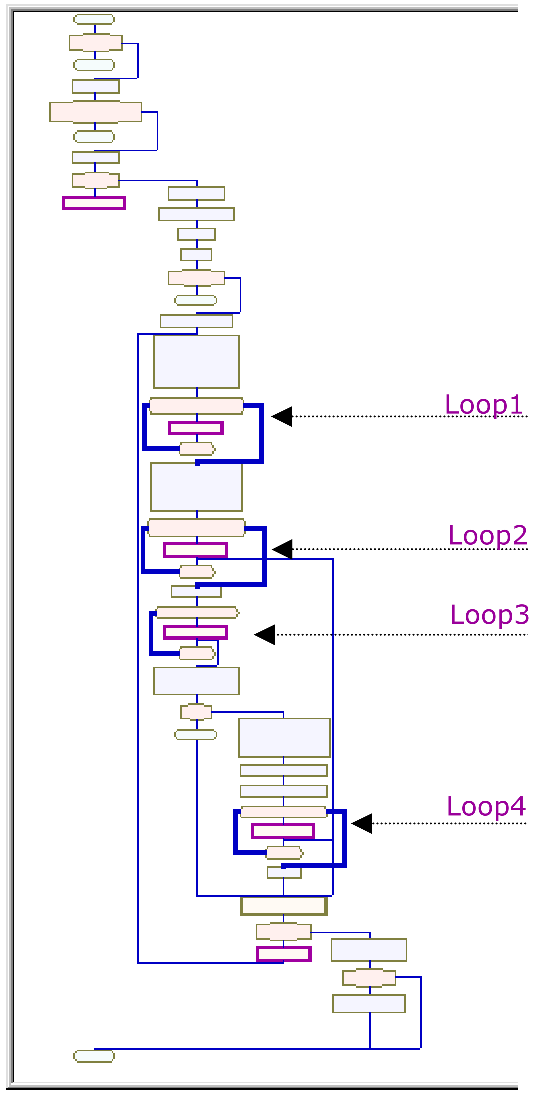
Condensed View

Detailed View

Divide and Conquer:

You just viewed the top-level flowchart;

Next: Flowcharts of the Inner Parts.



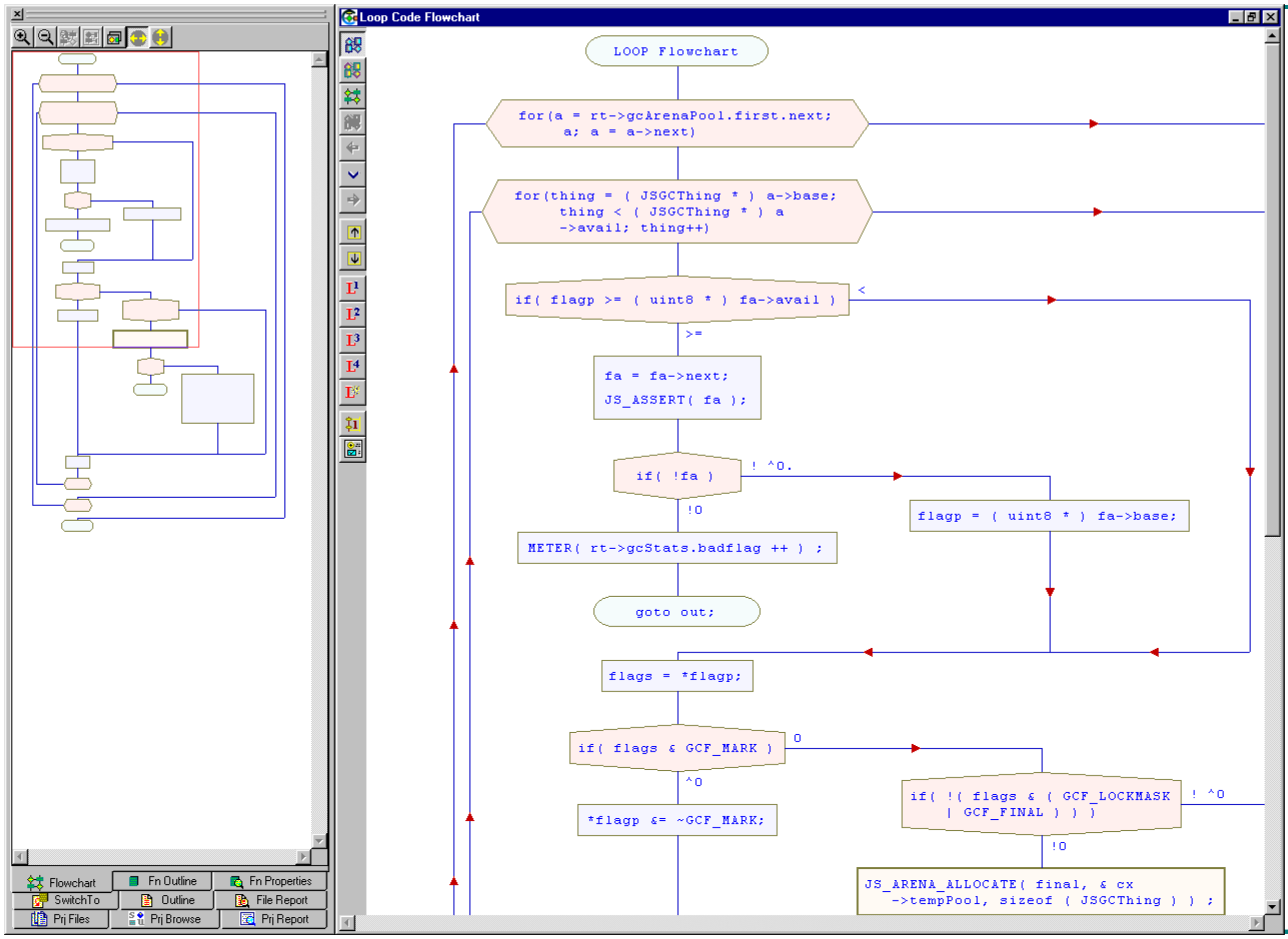
Click on a loop or switch symbol to create its flowchart.

← The Major Loops in js_GC

A purple outline ▭ indicates a high-level symbol.

It hides the internal details of a loop, switch etc.

The 1st Major Loop in js_GC

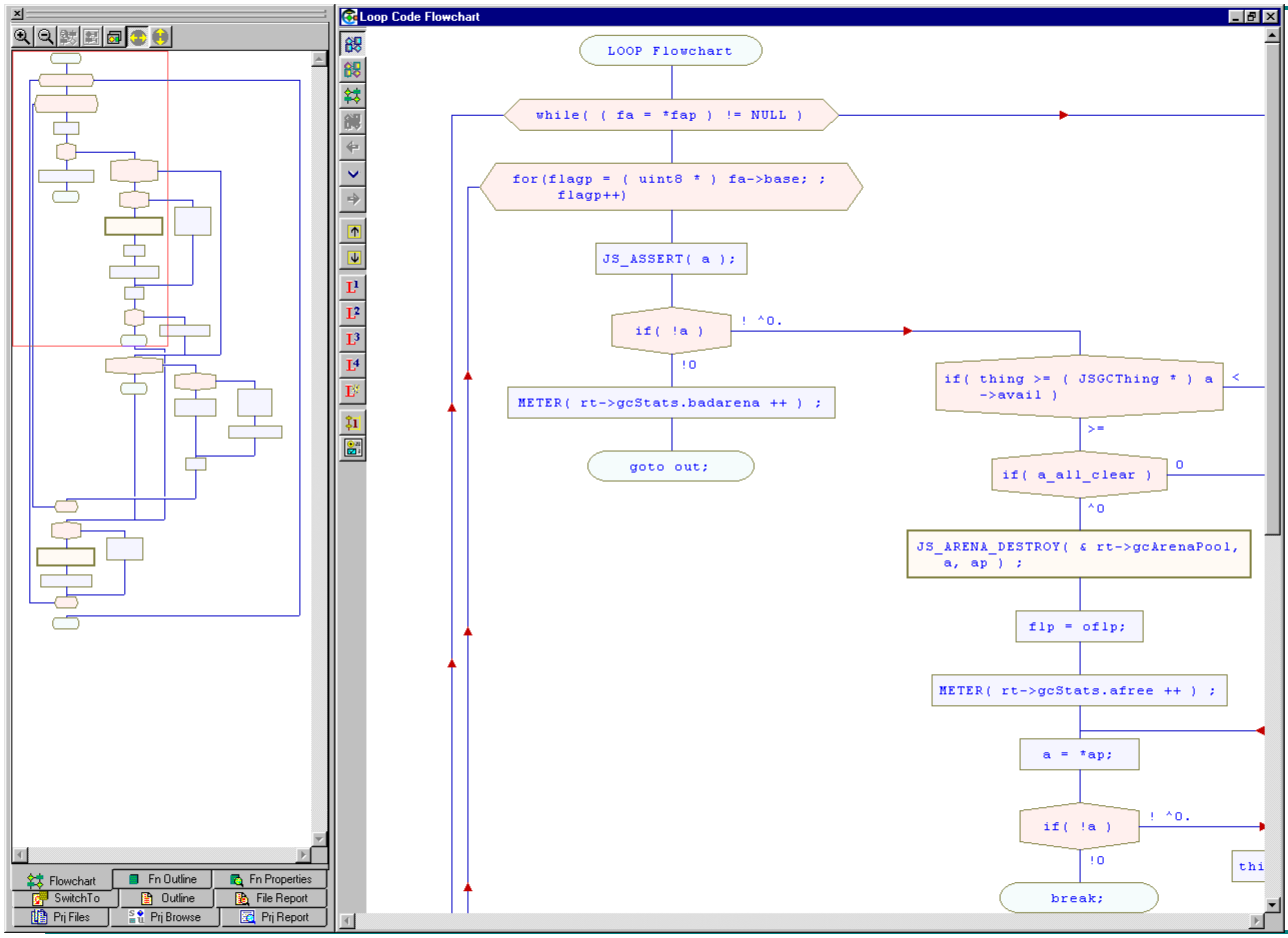


```
LOOP Flowchart
for(a = rt->gcArenaPool.first.next;
    a; a = a->next)
for(thing = ( JSGCThing * ) a->base;
    thing < ( JSGCThing * ) a
    ->avail; thing++)
if( flagp >= ( uint8 * ) fa->avail )
fa = fa->next;
JS_ASSERT( fa );
if( !fa )
flagp = ( uint8 * ) fa->base;
METER( rt->gcStats.badflag ++ ) ;
goto out;
flags = *flagp;
if( flags & GCF_MARK )
*flagp &= ~GCF_MARK;
if( !( flags & ( GCF_LOCKMASK
  | GCF_FINAL ) ) )
JS_ARENA_ALLOCATE( final, & cx
  ->tempPool, sizeof ( JSGCThing ) ) ;
```

← Flowchart of 1st major loop in js_GC

You will need just two minutes to understand this loop.

The Next Major Loop in js_GC


Loop Code Flowchart
LOOP Flowchart
while((fa = *fap) != NULL)
for(flagp = (uint8 *) fa->base; ; flagp++)
JS_ASSERT(a);
if(!a)
! ^0.
!0
METER(rt->gcStats.badarena ++) ;
goto out;
if(thing >= (JSGCThing *) a ->avail)
<
>=
if(a_all_clear)
0
^0
JS_ARENA_DESTROY(& rt->gcArenaPool, a, ap) ;
flp = oflp;
METER(rt->gcStats.afree ++) ;
a = *ap;
if(!a)
! ^0.
!0
break;
thi
Flowchart
Fn Outline
Fn Properties
SwitchTo
Outline
File Report
Prj Files
Prj Browse
Prj Report

You will need less than five minutes to understand this loop.

The graphical view is easy to understand and **easy to recall.**

As we saw above, in about twenty minutes, you can clearly understand a 350-line function.

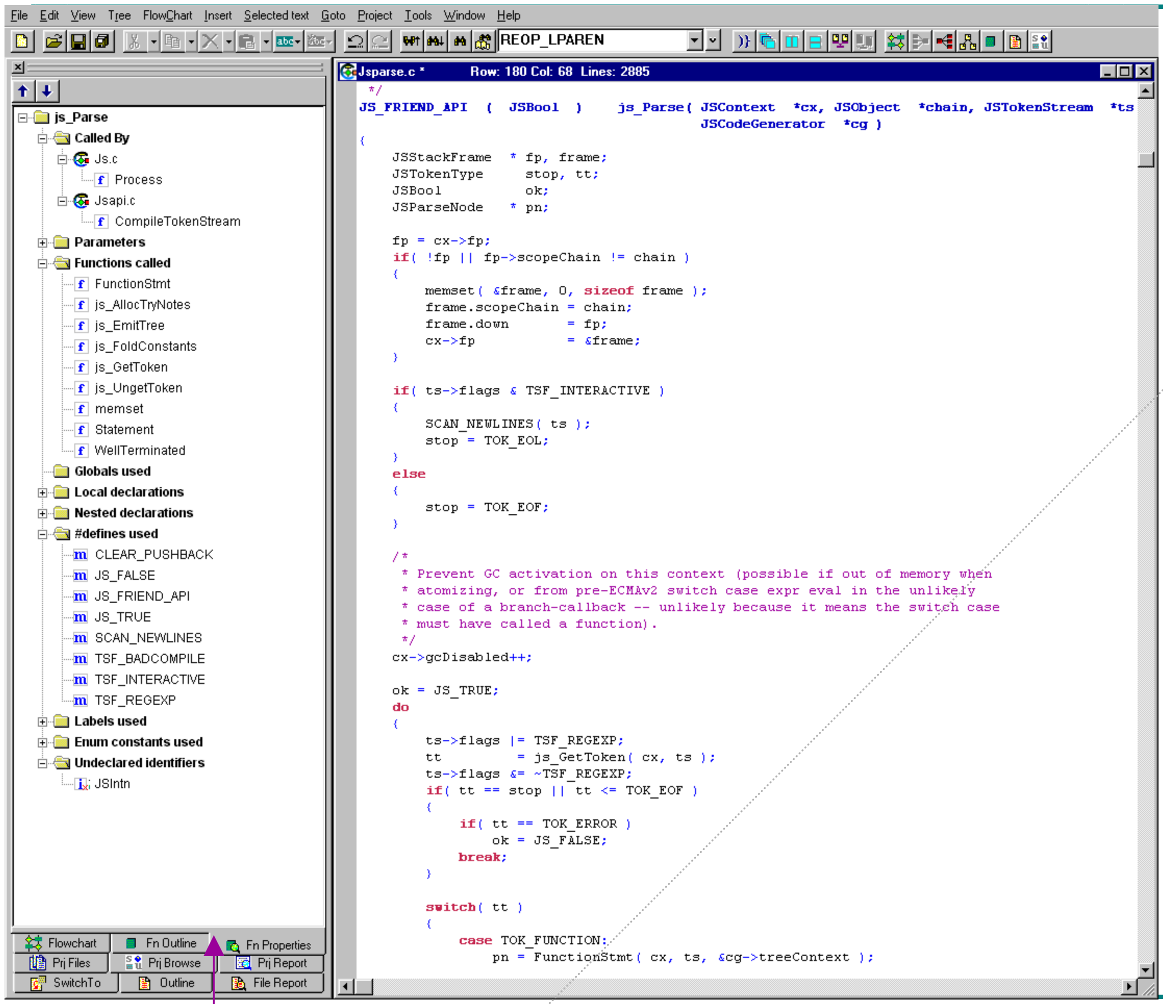
Without the flowcharts, it will take more than one hour.

Flowcharts are Valuable in many situations

1. after designing or modifying code – view the whole function;
   make sure all conditions are covered.
2. during code-reviews, the whole team saves a lot of time
3. during integration testing, you can easily undertand other team members' code.
4. when you inherit legacy code.
5. when you are a new member of the team.

You can export a flowchart as a .bmp or .jpg file for documentation.

Function Properties - Called By, Functions Called, Globals Used etc.


File Edit View Tree FlowChart Insert Selected text Goto Project Tools Window Help
REOP_LPAREN
Jsparse.c * Row: 180 Col: 68 Lines: 2885
Flowchart
Fn Outline
Fn Properties
Prj Files
Prj Browse
Prj Report
SwitchTo
Outline
File Report

```
js_Parse
  Called By
    Js.c
      Process
    Jsapi.c
      CompileTokenStream
  Parameters
  Functions called
    FunctionStmt
    js_AllocTryNotes
    js_EmitTree
    js_FoldConstants
    js_GetToken
    js_UngetToken
    memset
    Statement
    WellTerminated
  Globals used
  Local declarations
  Nested declarations
  #defines used
    CLEAR_PUSHBACK
    JS_FALSE
    JS_FRIEND_API
    JS_TRUE
    SCAN_NEWLINES
    TSF_BADCOMPILE
    TSF_INTERACTIVE
    TSF_REGEXP
  Labels used
  Enum constants used
  Undeclared identifiers
    JSIntn
```

```
 */
JS_FRIEND_API  (  JSBool  )    js_Parse( JSContext  *cx, JSObject  *chain, JSTokenStream  *ts
                                          JSCodeGenerator  *cg )
{
    JSStackFrame  * fp, frame;
    JSTokenType     stop, tt;
    JSBool          ok;
    JSParseNode   * pn;

    fp = cx->fp;
    if( !fp || fp->scopeChain != chain )
    {
        memset( &frame, 0, sizeof frame );
        frame.scopeChain = chain;
        frame.down       = fp;
        cx->fp           = &frame;
    }

    if( ts->flags & TSF_INTERACTIVE )
    {
        SCAN_NEWLINES( ts );
        stop = TOK_EOL;
    }
    else
    {
        stop = TOK_EOF;
    }

    /*
     * Prevent GC activation on this context (possible if out of memory when
     * atomizing, or from pre-ECMAv2 switch case expr eval in the unlikely
     * case of a branch-callback -- unlikely because it means the switch case
     * must have called a function).
     */
    cx->gcDisabled++;

    ok = JS_TRUE;
    do
    {
        ts->flags |= TSF_REGEXP;
        tt          = js_GetToken( cx, ts );
        ts->flags &= ~TSF_REGEXP;
        if( tt == stop || tt <= TOK_EOF )
        {
            if( tt == TOK_ERROR )
                ok = JS_FALSE;
            break;
        }

        switch( tt )
        {
            case TOK_FUNCTION:
                pn = FunctionStmt( cx, ts, &cg->treeContext );
```

♦ Place the cursor in a function.

Click the "Fn Properties" tab in the browse-window.

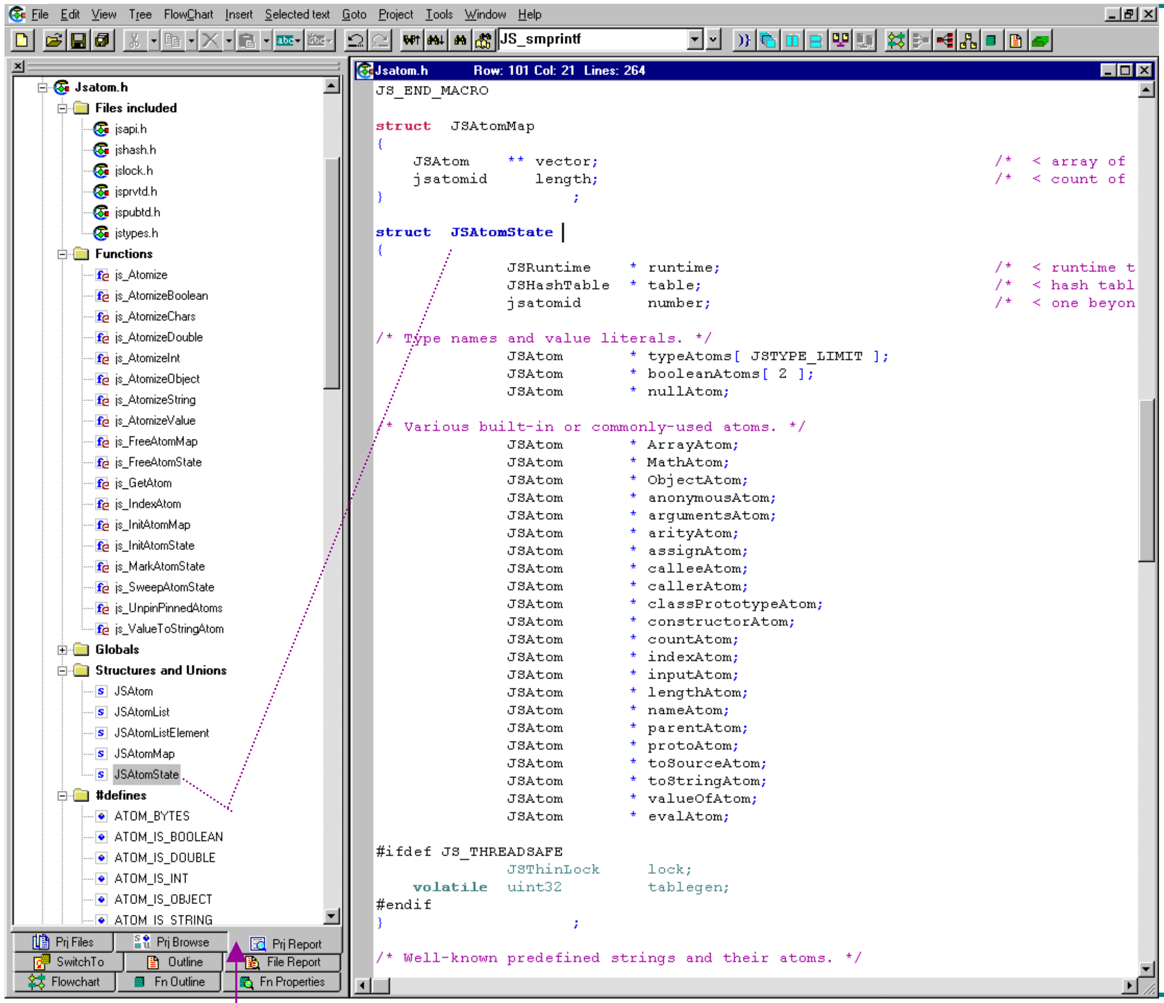
♦ Double-click on a node.

Also, try right-click on a node.

Function Properties

Overview of Globals, Functions, Structs etc.

from each Source File


File Edit View Tree FlowChart Insert Selected text Goto Project Tools Window Help
JS_smprintf
Jsatom.h Row: 101 Col: 21 Lines: 264
Prj Files
Prj Browse
Prj Report
SwitchTo
Outline
File Report
Flowchart
Fn Outline
Fn Properties

```
Jsatom.h
  Files included
    jsapi.h
    jshash.h
    jslock.h
    jsprvtd.h
    jspubtd.h
    jstypes.h
  Functions
    js_Atomize
    js_AtomizeBoolean
    js_AtomizeChars
    js_AtomizeDouble
    js_AtomizeInt
    js_AtomizeObject
    js_AtomizeString
    js_AtomizeValue
    js_FreeAtomMap
    js_FreeAtomState
    js_GetAtom
    js_IndexAtom
    js_InitAtomMap
    js_InitAtomState
    js_MarkAtomState
    js_SweepAtomState
    js_UnpinPinnedAtoms
    js_ValueToStringAtom
  Globals
  Structures and Unions
    JSAtom
    JSAtomList
    JSAtomListElement
    JSAtomMap
    JSAtomState
  #defines
    ATOM_BYTES
    ATOM_IS_BOOLEAN
    ATOM_IS_DOUBLE
    ATOM_IS_INT
    ATOM_IS_OBJECT
    ATOM_IS_STRING
```

```
JS_END_MACRO

struct  JSAtomMap
{
    JSAtom    ** vector;                                        /*  < array of
    jsatomid     length;                                        /*  < count of
}                 ;

struct  JSAtomState
{
              JSRuntime    * runtime;                           /*  < runtime t
              JSHashTable  * table;                             /*  < hash tabl
              jsatomid       number;                            /*  < one beyon

/* Type names and value literals. */
              JSAtom       * typeAtoms[ JSTYPE_LIMIT ];
              JSAtom       * booleanAtoms[ 2 ];
              JSAtom       * nullAtom;

/* Various built-in or commonly-used atoms. */
              JSAtom       * ArrayAtom;
              JSAtom       * MathAtom;
              JSAtom       * ObjectAtom;
              JSAtom       * anonymousAtom;
              JSAtom       * argumentsAtom;
              JSAtom       * arityAtom;
              JSAtom       * assignAtom;
              JSAtom       * calleeAtom;
              JSAtom       * callerAtom;
              JSAtom       * classPrototypeAtom;
              JSAtom       * constructorAtom;
              JSAtom       * countAtom;
              JSAtom       * indexAtom;
              JSAtom       * inputAtom;
              JSAtom       * lengthAtom;
              JSAtom       * nameAtom;
              JSAtom       * parentAtom;
              JSAtom       * protoAtom;
              JSAtom       * toSourceAtom;
              JSAtom       * toStringAtom;
              JSAtom       * valueOfAtom;
              JSAtom       * evalAtom;

#ifdef JS_THREADSAFE
              JSThinLock     lock;
    volatile  uint32         tablegen;
#endif
}                 ;

/* Well-known predefined strings and their atoms. */
```

Project Report

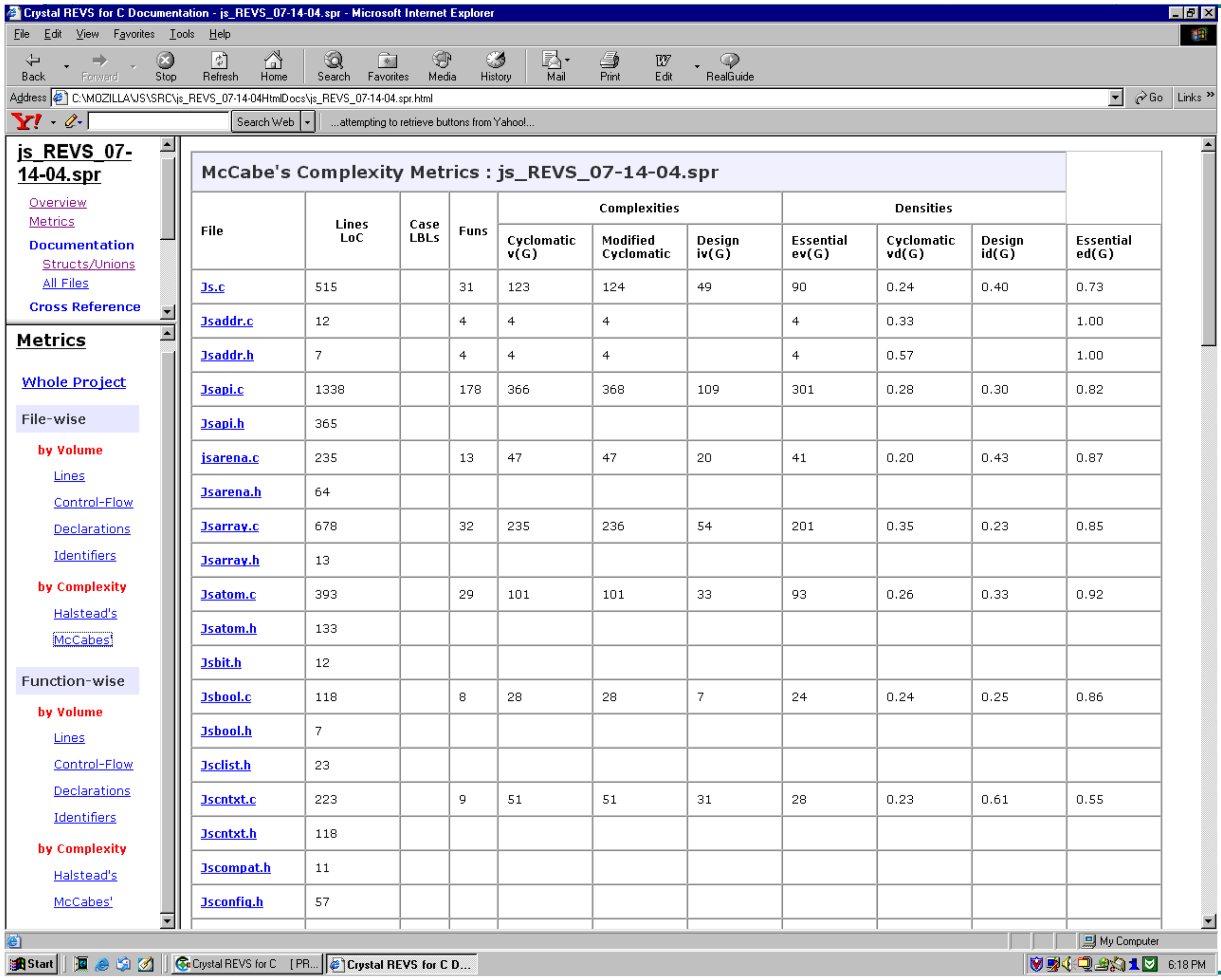
Get an overview of source files:

- a file-by-file report of all the globals in the project;
- all the functions in the project,
- all the structs in the project, etc.

To go to the definition of a global or a function :

double-click on an entry in the overview report.

Software Metrics



Crystal REVS for C Documentation - js_REVS_07-14-04.spr - Microsoft Internet Explorer

Address C:\MOZILLA\JS\SRC\js_REVS_07-14-04HtmlDocs\js_REVS_07-14-04.spr.html

...attempting to retrieve buttons from Yahoo!...

js_REVS_07-14-04.spr

- Overview
- Metrics
- **Documentation**
  - Structs/Unions
  - All Files
- **Cross Reference**

Metrics

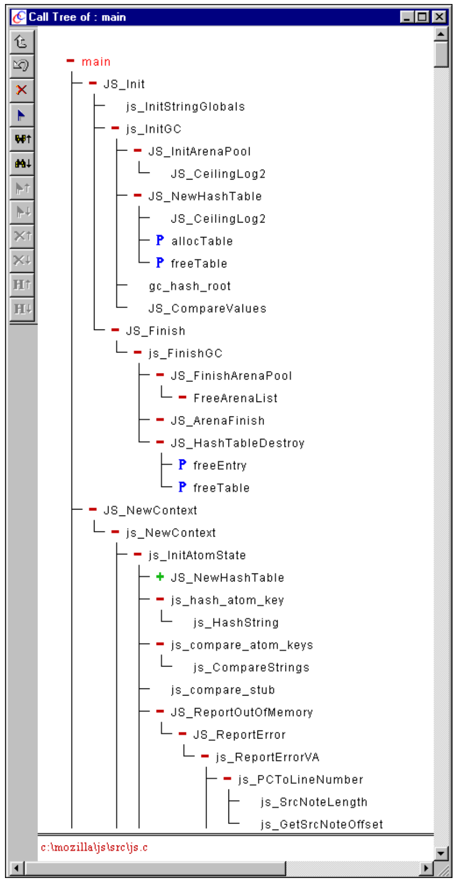
- Whole Project
- File-wise
  - by Volume
    - Lines
    - Control-Flow
    - Declarations
    - Identifiers
  - by Complexity
    - Halstead's
    - McCabes'
- Function-wise
  - by Volume
    - Lines
    - Control-Flow
    - Declarations
    - Identifiers
  - by Complexity
    - Halstead's
    - McCabes'

McCabe's Complexity Metrics : js_REVS_07-14-04.spr

File	Lines LoC	Case LBLs	Funs	Complexities				Densities		
				Cyclomatic v(G)	Modified Cyclomatic	Design iv(G)	Essential ev(G)	Cyclomatic vd(G)	Design id(G)	Essential ed(G)
Js.c	515		31	123	124	49	90	0.24	0.40	0.73
Jsaddr.c	12		4	4	4		4	0.33		1.00
Jsaddr.h	7		4	4	4		4	0.57		1.00
Jsapi.c	1338		178	366	368	109	301	0.28	0.30	0.82
Jsapi.h	365									
jsarena.c	235		13	47	47	20	41	0.20	0.43	0.87
Jsarena.h	64									
Jsarray.c	678		32	235	236	54	201	0.35	0.23	0.85
Jsarray.h	13									
Jsatom.c	393		29	101	101	33	93	0.26	0.33	0.92
Jsatom.h	133									
Jsbit.h	12									
Jsbool.c	118		8	28	28	7	24	0.24	0.25	0.86
Jsbool.h	7									
Jsclist.h	23									
Jscntxt.c	223		9	51	51	31	28	0.23	0.61	0.55
Jscntxt.h	118									
Jscompat.h	11									
Jsconfig.h	57									

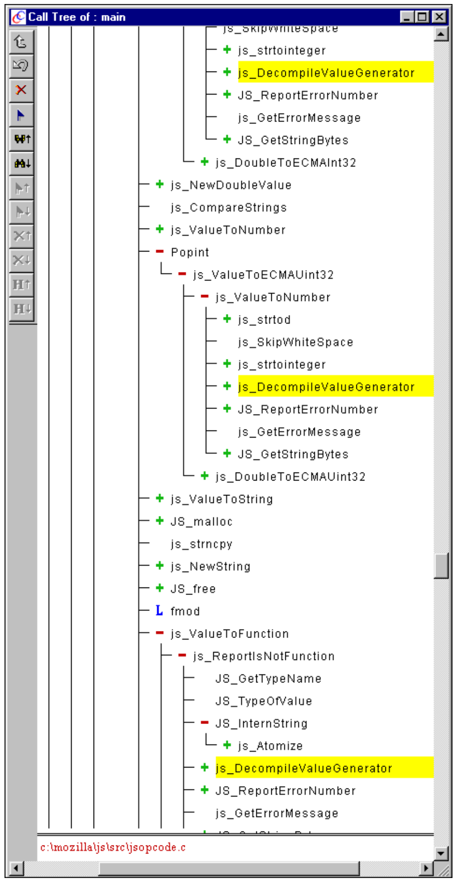
You can also create javadoc-like HTML Reports.

Call-Trees, Caller-Trees, File-Trees


Call Tree of : main

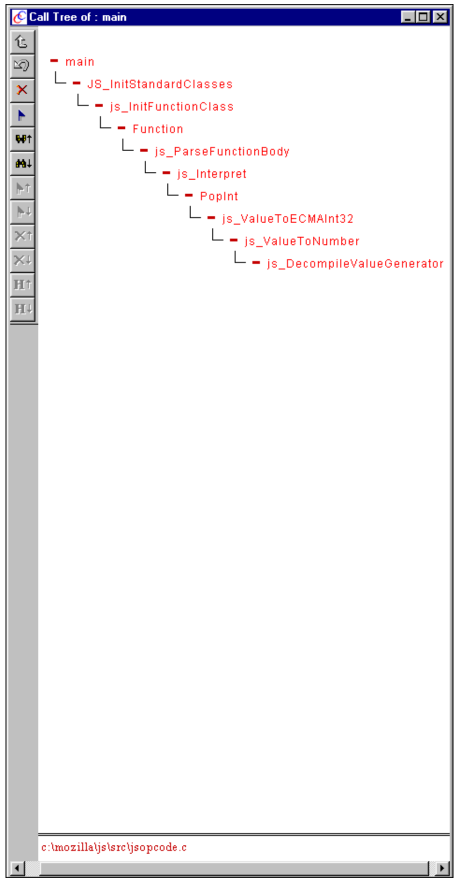
```
- main
  - JS_Init
      js_InitStringGlobals
    - js_InitGC
      - JS_InitArenaPool
          JS_CeilingLog2
      - JS_NewHashTable
          JS_CeilingLog2
        P allocTable
        P freeTable
        gc_hash_root
        JS_CompareValues
    - JS_Finish
      - js_FinishGC
        - JS_FinishArenaPool
          - FreeArenaList
        - JS_ArenaFinish
        - JS_HashTableDestroy
          P freeEntry
          P freeTable
  - JS_NewContext
    - js_NewContext
      - js_InitAtomState
        + JS_NewHashTable
        - js_hash_atom_key
            js_HashString
        - js_compare_atom_keys
            js_CompareStrings
          js_compare_stub
        - JS_ReportOutOfMemory
          - JS_ReportError
            - js_ReportErrorVA
              - js_PCToLineNumber
                  js_SrcNoteLength
                  js_GetSrcNoteOffset
```

c:\mozilla\js\src\js.c


Call Tree of : main

```
        js_SkipWhiteSpace
      + js_strtointeger
      + js_DecompileValueGenerator
      + JS_ReportErrorNumber
        js_GetErrorMessage
      + JS_GetStringBytes
    + js_DoubleToECMAInt32
+ js_NewDoubleValue
  js_CompareStrings
+ js_ValueToNumber
- PopInt
  - js_ValueToECMAUint32
    - js_ValueToNumber
      + js_strtod
        js_SkipWhiteSpace
      + js_strtointeger
      + js_DecompileValueGenerator
      + JS_ReportErrorNumber
        js_GetErrorMessage
      + JS_GetStringBytes
    + js_DoubleToECMAUint32
+ js_ValueToString
+ JS_malloc
  js_strncpy
+ js_NewString
+ JS_free
L fmod
- js_ValueToFunction
  - js_ReportIsNotFunction
      JS_GetTypeName
      JS_TypeOfValue
    - JS_InternString
      + js_Atomize
    + js_DecompileValueGenerator
    + JS_ReportErrorNumber
      js_GetErrorMessage
```

c:\mozilla\js\src\jsopcode.c


Call Tree of : main

```
- main
  - JS_InitStandardClasses
    - js_InitFunctionClass
      - Function
        - js_ParseFunctionBody
          - js_Interpret
            - PopInt
              - js_ValueToECMAInt32
                - js_ValueToNumber
                  - js_DecompileValueGenerator
```

c:\mozilla\js\src\jsopcode.c

1. Call Tree of main()

2. The result of a node-search for js_DecompileValueGenerator() in the tree.

3. **View just the call-path:** Crystal C hides all nodes except those in the call-path of js_DecompileValueGenerator()

If you are not using Crystal C/C++,

you are spending a lot more time than you need to.