

CONSCRIPT for Firefox

September 24, 2010

Abstract

This document is a report on the work of porting the research idea of CONSCRIPT to Mozilla Firefox.

Contents

1	Installation	2
1.1	Plain usage	2
1.2	Building process	2
1.3	Code directories	3
2	Implementation	3
2.1	Research idea	3
2.2	General overview	3
2.3	Implementation details	4
2.3.1	advice_advice() function details	6
2.3.2	Implementation benchmarks	7
3	Testing on the Internet	7
4	Known problems/limitations	10
A	Javascript shell	12
B	Eclipse settings	12
	References	12

1 Installation

All steps explained in this document were tested on a GNU/Linux Ubuntu 10.04 machine (32bit).

1.1 Plain usage

In case of emergency, there is still the wiki of Mozilla on https://developer.mozilla.org/En/Developer_Guide/Build_Instructions .

There is a fully functional build of Mozilla Firefox with the CONSCRIPT extension integrated that you can download from <http://www.cqrit.be/conscript/bin/> . Download the .tar.bz2 file somewhere, let's say /tmp, and unpack it with `tar -xjvf conscript-for-firefox.tar.bz2` . You can start this new version of Firefox by executing `./firefox` .

In that same online directory you can also find a nightly build of the Greasemonkey extension (you need to use this version since current stable releases will not work on Firefox 4). After installing this extension you can surf to <http://www.cqrit.be/conscript/bin/conscript.user.js> and click on the install button.

1.2 Building process

Install Eclipse CDT. The CDT Project provides a fully functional C and C++ Integrated Development Environment based on the Eclipse platform. The download location can be found on <http://www.eclipse.org/cdt/> .

Download Mozilla source code. The Mozilla project uses Mercurial as a source code management tool. The most recent code can be found in the main development tree on <http://hg.mozilla.org/mozilla-central/>. Next step is to pull the code to a local folder via the command `hg clone http://hg.mozilla.org/mozilla-central/ src` . More information can be found on [https://developer.mozilla.org/en/Mozilla_Source_Code_\(Mercurial\)](https://developer.mozilla.org/en/Mozilla_Source_Code_(Mercurial)) .

Start new C++ project. Create a new C++ project in Eclipse and set the Location parameter. It is also possible to tweak the indexer and the build options (e.g. to build the Javascript shell from within Eclipse).

Configure Mozilla build options. In the /src directory, create a new file mozconfig with this content (debugging enabled, 4 jobs in parallel):

```
. $topsrcdir/browser/config/mozconfig
mk_add_options MOZ_OBJDIR=@TOPSRCDIR@/objdir-ff-debug
ac_add_options --enable-debug
ac_add_options --disable-optimize
# some options if you're planning
# on building a optimized variant
#ac_add_options --disable-tests
#ac_add_options --disable-debug
#ac_add_options --enable-optimize
#ac_add_options --without-system-nspr
#ac_add_options --without-system-zlib
#ac_add_options --without-system-jpeg
#ac_add_options --without-system-png
mk_add_options MOZ_MAKE_FLAGS="-j4"
```

TIP: It is also good to know that the Makefile contains scripts that will use the hg tool to update the source. It is good idea to renew your hg binary (typically installed in /usr/bin/hg) to something like hg_. You can of course also edit the Makefile to remove those lines.

Compilation process for:

Firefox. In the `/src` directory, start the compilation process for Firefox via `make -f client.mk1`. The result of this compilation process can be found in `/src/objdir-ff-debug/dist/bin/firefox`. You can build multiple version of the same code by changing the `mozconfig` file (this leaves you with the option to have both a debug-enabled and optimized Firefox).

Spidermonkey. Go to the directory `/src/js/src` and execute `./configure` (do this only the first time). The next step is to actually build the whole thing via `make -j4` (the `-j4` option tells the compiler to use 4 parallel jobs). This process should also compile the Javascript shell (should be accessible via `/src/js/src/js`). If you ever want to compile Firefox after compiling Spidermonkey, don't forget to do a `make distclean` first.

Creating a package. Goto the directory `/src/objdir-ff-debug` and type the command `make package`. This process should end with a `.tar.gz` file that can be used to distribute among other people.

1.3 Code directories

Some interesting code directories within the Mozilla development tree:

`/src/js/src` This directory contains all the files necessary to generate the Spidermonkey library. Interesting API functions can be found in the `jsapi.cpp` file.

`/src/js/src/shell` Spidermonkey comes with a Javascript shell (after compilation, this shell can be found in `/src/js/src/js`). This shell is a handy environment to do all sorts of tests. If debugging is enabled (you can add `--enable-debug` to the `./configure` command), you have access to all sorts of debugging functions.

`/src/dom` If you want to know how specific (or generic) DOM functions are implemented, this is the place to be.

2 Implementation

This section will give you a brief overview of the implementation details of CONSCRIPT for Firefox. The first parts explains the basic research ideas from the original paper. Next a more general overview of the solution is given. This section will be closed with more specific implementation details.

2.1 Research idea

The idea behind CONSCRIPT is to create a client-side advice implementation for security, built-in in the browser itself. Via CONSCRIPT, the hosting page can express fine-grained application-specific security policies that are enforced at runtime. In the original paper, the authors proposed a type system to be sure that policies are well written.

2.2 General overview

In order to port CONSCRIPT to Firefox, we must first introduce the idea of Aspect Oriented Programming into Spidermonkey. It is also good to know that Spidermonkey (and certainly the latest version called Jaegermonkey) makes heavily use of all kinds of JIT ideas (e.g. method-jitting and trace-jitting). This will affect the final solution because advice policies will influence the process of JIT-ing.

¹Instead of applying the TIP from above, you can also use the `make -f client.mk build` command.

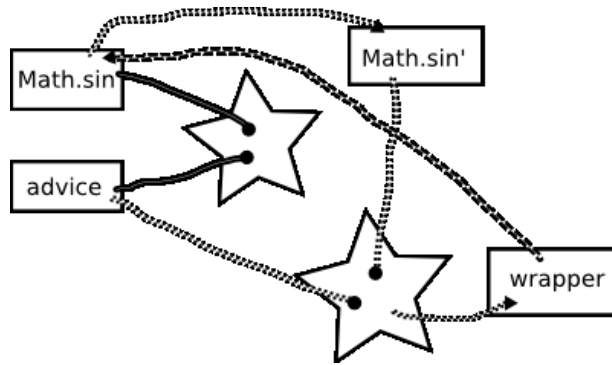


Figure 1: General overview of the implementation

The main idea is to (i) clone the original function (the one we want to advice) and make it unreachable for normal functions and (ii) replace that original function with some kind of wrapper function that calls the advice function with a reference to the cloned original function (see Fig. 1).

To prevent malicious actors of influencing the whole process, we create an new clean context where we introduce both the `advice` and `clone` (e.g. the `Math.sin'` function) functions. Next we generate a function in this new context that will return the result of the advice function called with the corresponding parameters.

This result (which is a function called `wrapper`) will be set as the new value of `advice`. To make things working, we need to clear the property cache for that context (because this cache still holds the data for the original `func` function). From now on, the original access point for a specific property (function) is replaced with the `wrapper` function. This `wrapper` function, however, still has a reference to the original (cloned) function `clone`.

2.3 Implementation details

Two new files are created within the `/src/js/src` directory. The file `jsadvice.cpp` (with its corresponding header file `jsadvice.h`) contains all code to implement the advice functionality (the main function itself is named `advice_advice()`). In order to use these files, you have to include them in the compilation process by adding the two files to `Makefile.in` (the `Makefile` is automatically generated from this one). Also, don't forget to actually copy the two files to `/src/js/src`.

```

1 --- ./srcclean/js/src/Makefile.in
2 +++ ./src/js/src/Makefile.in
3 @@ -162,6 +165,7 @@
4         jsxdraapi.cpp \
5         jsxml.cpp \
6         prmjtime.cpp \
7 + jsadvice.cpp \
8         $(NULL)
9
10 INSTALLED_HEADERS = \
11 @@ -233,6 +237,7 @@
12         jsval.h \
13         jsvalue.h \
14         prmjtime.h \
15 + jsadvice.h \
16         $(NULL)

```

The `js_InitAdviceFunctions()` function should be called in the Firefox source code whenever one wants access to the `advice()` function in a specific context (don't forget to include the `jsadvice.h` file). Currently this is done in three locations:

- ./content/base/src/nsInProcessTabChildGlobal.cpp:328

```

---srcclean/content/base/src/nsInProcessTabChildGlobal.cpp
+++ src/content/base/src/nsInProcessTabChildGlobal.cpp
@@ -50,6 +50,7 @@
4  #include "nsIJSContextStack.h"
   #include "nsFrameLoader.h"
   #include "nsIPrivateDOMEvent.h"
+ #include "jsadvice.h"

9  bool SendSyncMessageToParent(void* aCallbackData,
                               const nsAString& aMessage,
@@ -324,6 +324,7 @@
   NS_ENSURE_SUCCESS(rv, false);

14  JS_SetGlobalObject(cx, global);
+ js_InitAdviceFunctions(cx, global);
   DidCreateCx();
   return NS_OK;
}

```

- ./dom/base/nsJSEnvironment.cpp:3329

```

--- ./srcclean/dom/base/nsJSEnvironment.cpp
2  +++ src/dom/base/nsJSEnvironment.cpp
   @@ -96,6 +96,7 @@

   #include "jsdbgapi.h" // for JS_ClearWatchPointsForObject
   #include "jsxdrapi.h"
7  + #include "jsadvice.h"
   #include "nsIArray.h"
   #include "nsIObjectInputStream.h"
   #include "nsIObjectOutputStream.h"

12 @@ -3323,6 +3325,9 @@
   ::JS_DefineFunctions(mContext, globalObj, EthogramFunctions);
   #endif

+ js_InitAdviceFunctions(mContext, globalObj);
17  JSOptionChangedCallback(js_options_dot_str, this);

   return rv;

```

- ./js/src/shell/js.cpp:5160

```

1  --- ./srcclean/js/src/shell/js.cpp
   +++ ./src/js/src/shell/js.cpp
   @@ -77,6 +77,7 @@
   #include "jstracer.h"
   #include "jsxml.h"
6  #include "jsperf.h"
+ #include "jsadvice.h"

   #include "prmjtime.h"

11 @@ -5155,6 +5157,7 @@
                               its_setter, JSPROP_READONLY))
   return NULL;

+ js_InitAdviceFunctions(cx, glob);
16  return glob;

```

```
}  
}
```

2.3.1 advice_advice() function details

A detailed and commented version of the `advice_advice()` function is given in Listing 1.

Listing 1: Documented `advice()` function

```
static JSBool  
advice_advice (JSContext *cx, uintN argc, jsval *argv)  
3 {  
    /* We expect two arguments: advice(function, policy) */  
    if (argc == 2)  
    {  
        /* JS variables. */  
8        JSRuntime *rt = cx->runtime;  
        JSContext *newcx;  
        JSObject *global;  
        JSScript *script;  
        jsval rval;  
13        JSObject *wrapper;  
        JSFunction *orig;  
        JSObject *origclone;  
        JSFunction *adv;  
        jsval memberVal;  
18        jsval ifaceVal;  
  
        /* Create a context. */  
        newcx = JS_NewContext(rt, 8192);  
        if (newcx == NULL)  
23            return 1;  
  
        JS_SetOptions(cx, JSOPTION_VAROBJFIX);  
        JS_SetVersion(cx, JSVERSION_LATEST);  
  
28        /* Create the global object. */  
        global = JS_NewGlobalObject(newcx, &global_class);  
        if (global == NULL)  
            return 1;  
  
33        /* Populate the global object with the standard globals,  
        like Object and Array. */  
        if (!JS_InitStandardClasses(newcx, global))  
            return 1;  
  
38        /* Get a reference to the to functions, given as arguments */  
        orig = JS_ValueToFunction(cx, argv[ARG_1]);  
        adv = JS_ValueToFunction(cx, argv[ARG_2]);  
  
43        /* Introduce the given advice function as "adv" in the new context */  
        rval = OBJECT_TO_JSVAL(adv);  
        JS_SetProperty(newcx, JS_GetGlobalObject(newcx), "adv", &rval);  
  
        /* Save the contents of some reserved slot values */  
        JS_GetReservedSlot(cx, orig, 0, &ifaceVal);  
48        JS_GetReservedSlot(cx, orig, 1, &memberVal);  
  
        /* Create a clone of the given function to advice */  
        origclone = JS_CloneFunctionObject(cx, orig, orig->getParent());
```

```

53     /* Write the saved contents in the new reserved slots */
    JS_SetReservedSlot(cx, origclone, 0, ifaceVal);
    JS_SetReservedSlot(cx, origclone, 1, memberVal);

    /* Introduce the clone of this function as "orig" in the new context */
58     rval = OBJECT_TO_JSVAL(origclone);
    JS_SetProperty(newcx, JS_GetGlobalObject(newcx), "orig", &rval);

    /* Code of the wrapper function */
    char const *src = "function (a,b,c,d,e,f,g,h,i,j,k,l,m,n) { var oldThisAdv = this.
        adv; this.adv = adv; this.arguments = arguments; this.local = this; var result
        = adv.call(this, orig, a,b,c,d,e,f,g,h,i,j,k,l,m,n); this.adv = oldThisAdv;
        return result; }";
63

    /* Compile the wrapper function */
    script = JS_CompileScript(newcx, JS_GetGlobalObject(newcx), src, strlen(src), "",
        0);
    JS_ASSERT(/* if not */ script != NULL /* then we have some syntax error */);

68     /* Execute the wrapper function within the new context */
    JS_ExecuteScript(newcx, JS_GetGlobalObject(newcx), script, &rval);

    /* Some checks to see if everything worked as supposed */
    JS_ASSERT(JSVAL_IS_OBJECT(rval));
73     JS_ASSERT(JS_ObjectIsFunction(newcx, (rval)));

    /* Convert the new wrapper function and the original function
       * to a JSObject */
    JSObject *wrapperobj = (JSObject *)JSVAL_TO_OBJECT(rval);
78     JSObject *origobj = JS_GetFunctionObject(orig);

    /* Write the wrapper object over the original object */
    memcpy(origobj, wrapperobj, sizeof(JSObject));

83     /* Cleanup. */
    JS_DestroyScript(newcx, script);
    JS_DestroyContext(newcx);
    return JS_TRUE;
}
88     return JS_FALSE;
}

```

2.3.2 Implementation benchmarks

Good benchmark results were not a primary concern, but as shown by the graph in Fig. 2, it is clear that introducing the advice functionality comes only with a minimal overhead.

3 Testing on the Internet

In order to test CONSCRIPT for Firefox on the Internet, one should install the Greasemonkey extension. Next step is to create a custom script (see Listing 2) that will be included on every page. This script contains code to insert a new `script` tag before all other `script` tags in the `head`. The source of the `script` tag is set to point to a remote `policies.js` file that will contain all policy code.

Listing 2: Greasemonkey script

```

1 // ==UserScript==

```

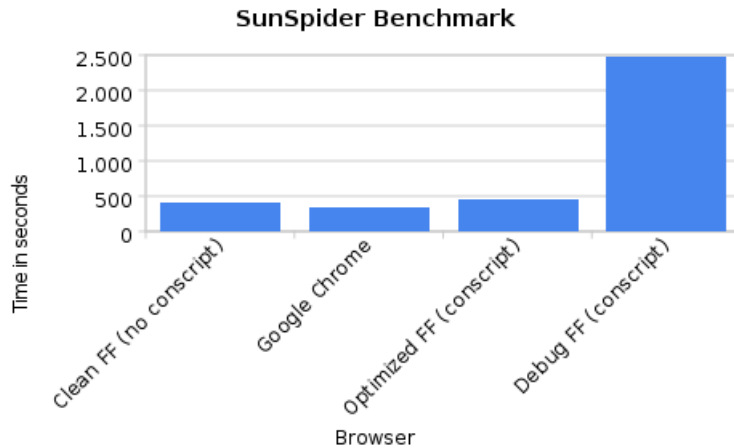


Figure 2: Simple benchmark results from the Sunspider Benchmark

```

// @name ConScript
// @namespace conscript
// @description Conscript policies
// @include *
6 // ==/UserScript==

var head= document.getElementsByTagName('head')[0];
var script= document.createElement('script');
script.type= 'text/javascript';
11 script.src= 'http://www.cqrit.be/conscript/policies.js';
head.parentNode.insertBefore(script, head);

```

With this construction, you can be sure that the policies will be set before any other script is executed. The last step is to define some interesting (and secure!) policies in the `policies.js` file (see Listing 3)

Listing 3: Advice policy file example

```

/**
 * Own implementation of the call() function. Use this version
3 * in your policies to protect you against prototype attacks like
 * Function.prototype.call = function (a,b,c,d) {};
 */
function uCall2 (o, f, p,q,r,s)
{
8   var old = o.f;
   o.f = f;
   try { var ret = o.f(p,q,r,s); } catch (e) { alert(e); var ret = old(p,q,r,s); }
   o.f = old;
   return ret;
13 }

/**
 * Forbid the user to send the word "javascript" via postMessage()
 */
18 function windowpostmessage_adv (f, msg, targetOrigin)
{
   alert("Window.postMessage(msg=" + msg + ", targetOrigin=" + targetOrigin + ")");

   if (msg.toLowerCase() == "javascript")
23     msg = "*** CENSORED BY CONSCRIPT ***";
}

```

```

    return uCall2(this, f, msg, targetOrigin);
}

28 /**
   * Ask the user if he really wants to open this new window
   */
function windowopen_adv (f, URL, windowName, windowFeatures)
{
33   if (confirm("Do you want to execute\nwindow.open(URL=" + URL + ", windowName=" +
        windowName + ", windowFeatures=" + windowFeatures + ")?"))
        return uCall2(this, f, window, URL,windowName, windowFeatures);
    else
        return false;
}

38 /**
   * Prevent the dynamic creation of script tags
   */
function script_adv (f, x)
43 {
    var el = uCall2(this,f,x);

    if (el.nodeName.toLowerCase() == "script")
    {
48     alert("prevented creation of script tag");
        throw "not allowed to create scripts";
    }
    else
        return el;
53 }

/**
   * Prevent the dynamic creation of iframe tags
   */
58 function iframe_adv (f, x)
{
    var el = uCall2(this,f,x);

    if (el.nodeName.toLowerCase() == "iframe")
63 {
        alert("prevented creation of dynamic IFRAME");
        throw "not allowed to create iframes";
    }
    else
68     return el;
}

/**
   * Only allow eval for JSON parsing and nothing else
73 */
function eval_adv (f, x)
{
    try
    {
78     var json = JSON.parse(x);
        alert("eval(" + x + ")");
        return json;
    }
    catch (e)
83 {
        throw "Invalid JSON data";
    }
}

```

```

    }
  }
88 /**
   * Ask the user if he is oke with sending the XMLHttpRequest
   */
function xmlhttprequestopen_adv (f, method, url, async, user, password)
{
93   if (!confirm("XMLHttpRequest.open(method=" + method + ", url=" + url + ")\nDo you want
       to allow this XMLHttpRequest?"))
       throw "Did not send data over an insecure line";

   return uCall2(this, f, method, url, async, user, password);
}
98 /**
   * Show the user what he sends over the wire with websockets
   */
function websocketsend_adv (f, data)
103 {
    alert("WebSocket.send(data=" + data + ")");
    return uCall2(this,f,x);
}

108 function adviceLoadingError (e)
{
    alert("Error while loading advice: " + e);
}

113 /**
   * Load all policies
   */
try { advice(eval, eval_adv); } catch (e) { adviceLoadingError(e); }
try { advice(HTMLDocument.prototype.createElement, iframe_adv); } catch (e) {
    adviceLoadingError(e); }
118 try { advice(Window.prototype.open, windowopen_adv); } catch (e) { adviceLoadingError(e);
    }
try { advice(window.postMessage, windowpostmessage_adv); } catch (e) { adviceLoadingError(
    e); }
//try { advice(HTMLDocument.prototype.getElementById, getelementbyid_adv); } catch (e) {
    adviceLoadingError(e); }
try { advice(XMLHttpRequest.prototype.open, xmlhttprequestopen_adv); } catch (e) {
    adviceLoadingError(e); }
try { advice(WebSocket.prototype.send, websocketsend_adv); } catch (e) {
    adviceLoadingError(e); }
123 /**
   * User scripts shouldn't have access to the contents of cookies
   * Work-around for the server not having implemented HTTP-Only cookies
   */
128 HTMLDocument.prototype.__defineGetter__("cookie", function () { return ""; });
HTMLDocument.prototype.__defineSetter__("cookie", function (x) { return ""; });

```

Depending on the behaviour of the defined policies, some website (or at least the functionality of it) will break. However, during the test periode, the browser didn't crash because of the implementation itself.

4 Known problems/limitations

- The type system is not implemented

- Never advice a function more than once
- Always use the function in the prototype
Instead of using `(new XMLHttpRequest()).open`, one must use `XMLHttpRequest.prototype.open`
- Limitations on the amount of parameters
The wrapper function currently supports only a limited amount of parameters (hard-coded in the wrapper function in `jsadvice.cpp`) and will probably fail with functions like `Math.max()`.
- Greasemonkey scripts are executed after loading the page, which means that you cannot protect yourself against a XSS attack (since this attack will be executed before your policies will be set).

A Javascript shell

To use the advice functionality within the Javascript shell, one must explicitly introduce the advice functions within the context used in the shell.

```
1 --- srcclean/js/src/shell/js.cpp 2010-09-20 13:02:18.000000000 +0200
+++ src/js/src/shell/js.cpp 2010-09-15 10:57:25.000000000 +0200
@@ -77,6 +77,7 @@
#include "jstracer.h"
#include "jspxml.h"
6 #include "jsperf.h"
+#include "jsadvice.h"

#include "prmjtime.h"

11 @@ -5155,6 +5157,7 @@
                its_setter, JSPROP_READONLY))

        return NULL;

+ js_InitAdviceFunctions(cx, glob);
16     return glob;
}
```

B Eclipse settings

In the properties of your project, you can make some changes to compile from within Eclipse (see Figures 1-3).

References

- [1] D. Flanagan, *JavaScript: the definitive guide*. O'Reilly Media, Inc., 2006.
- [2] L. Meyerovich and B. Livshits, "ConScript: Specifying and enforcing fine-grained security policies for Javascript in the browser," in *IEEE Symposium on Security and Privacy*, May 2010.
- [3] L. Meyerovich and B. Livshits, "ConScript: Specifying and enforcing fine-grained security policies for Javascript in the browser," Tech. Rep. MSR-TR-2009-158, Nov. 2009.
- [4] P. H. Phung, D. Sands, and A. Chudnov, "Lightweight self-protecting JavaScript," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pp. 47–60, ACM, 2009.
- [5] A. Guha, C. Saftoiu, and S. Krishnamurthi, "The essence of JavaScript," in *European Conference on Object Oriented Programming (ECOOP)*, 2010.
- [6] J. Magazinius, P. H. Phung, and D. Sands, "Safe Wrappers and Sane Policies for Self Protecting JavaScript,"
- [7] S. Maffei, J. C. Mitchell, and A. Taly, "Isolating JavaScript with filters, rewriting, and wrappers," *Computer Security-ESORICS 2009*, pp. 505–522, 2010.
- [8] B. S. Lerner, H. Venter, and D. Grossman, "Supporting Dynamic, Third-Party Code Customizations in JavaScript Using Aspects," 2010.
- [9] A. Gal, C. W. Probst, and M. Franz, "HotpathVM: an effective JIT compiler for resource-constrained devices," in *Proceedings of the 2nd international conference on Virtual execution environments*, p. 153, ACM, 2006.

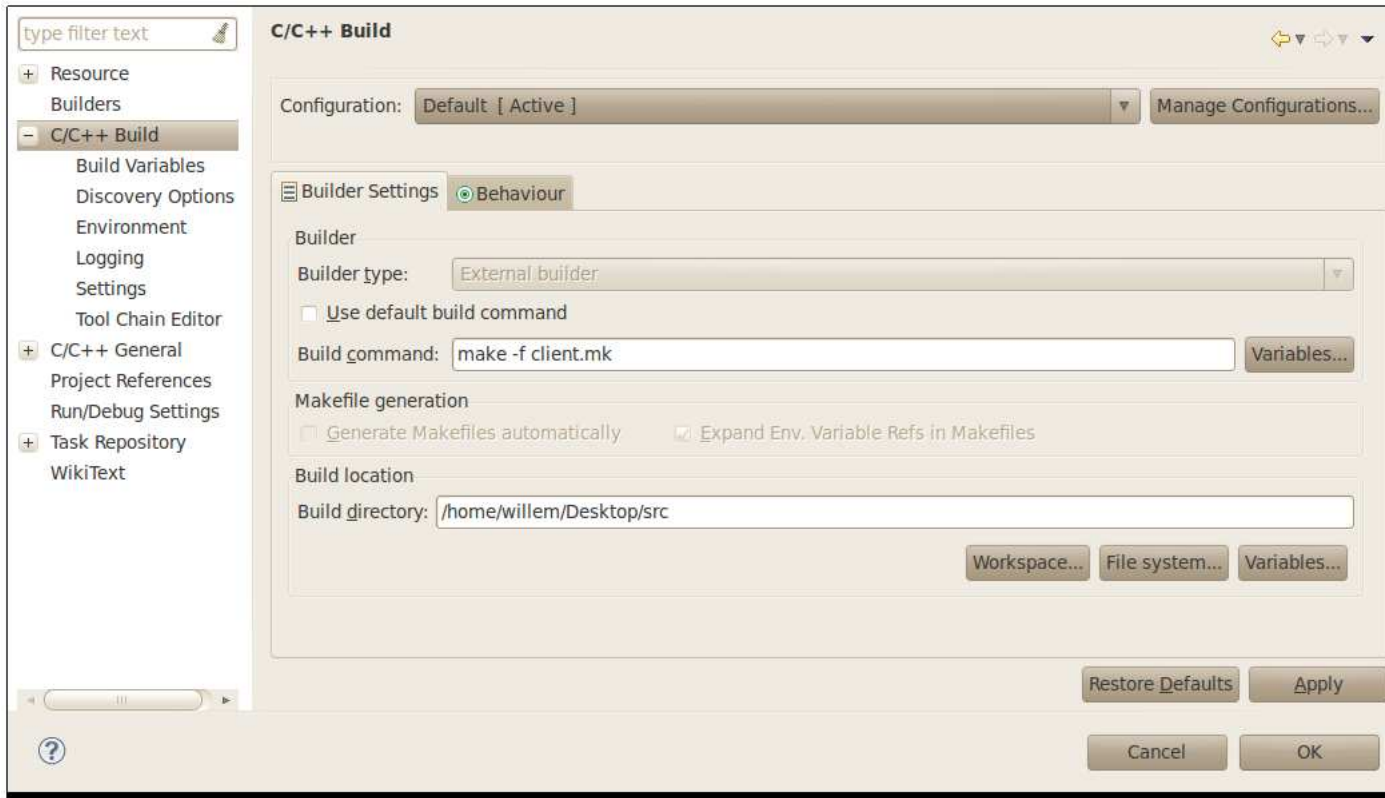


Figure 3: Eclipse Build Options

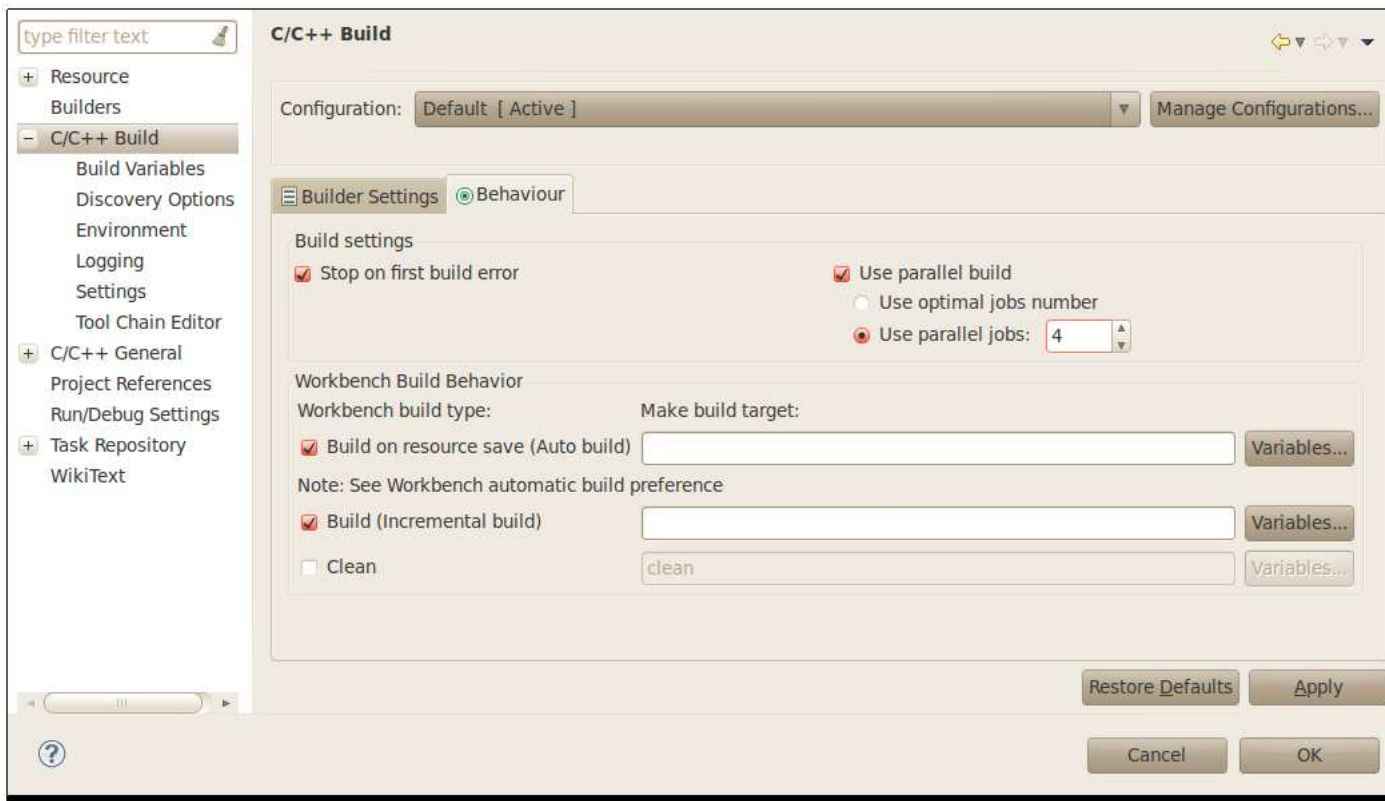


Figure 4: Eclipse Build Options

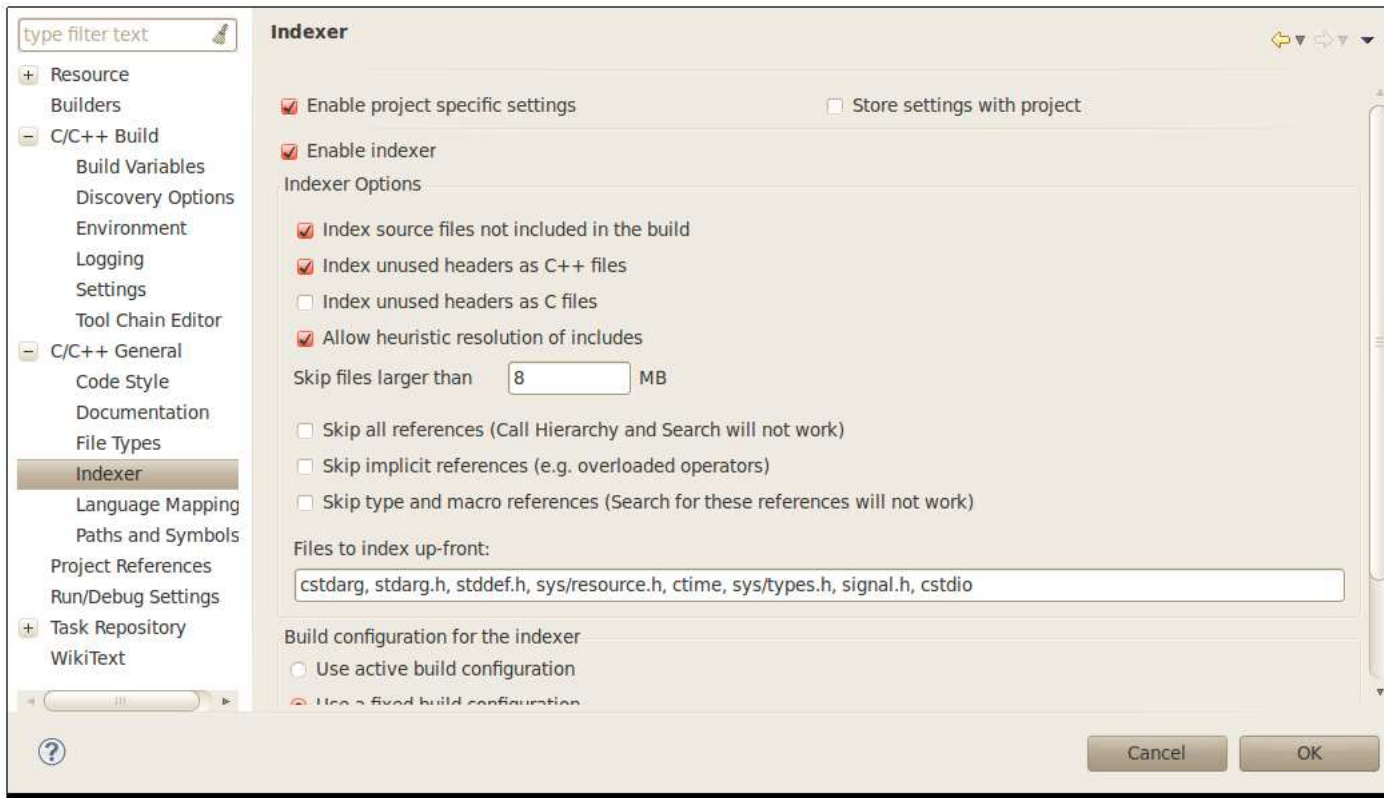


Figure 5: Eclipse Build Options