Bug 987069: [jsdbg2] Unicode characters in script URLs are not correctly encoded in Debugger.Script.url - Sebastian Zartner [:sebo] <sebastianzartner@gmail.com> - 3/24/2014 [reviews] [help]

Attachment 8550523: make ScriptSource filename encoding consistent - Tom Tromey :tromey <ttromey@mozilla.com> - Fri 4:42:00 PM [diff] [details]

Show Quick Help

Navigation: Overview | All Files | ~~nsScriptSecurityManager.cpp~~ | ~~WebSocket.cpp~~ | ~~nsFrameMessageManager.cpp~~ | ~~nsGlobalWindow.cpp~~ | ~~nsJSUtils.cpp~~ | ~~nsScriptLoader.cpp~~ | ~~EventListenerManager.cpp~~ | ~~nsJSProtocolHandler.cpp~~ | ~~nsNPAPIPlugin.cpp~~ | ~~PromiseCallback.cpp~~ | ~~chrome.ini~~ | ~~test_encoding.html~~ | ~~RuntimeService.cpp~~ | ~~ScriptLoader.cpp~~ | ~~WorkerPrivate.cpp~~ | ~~nsXBLProtoImplField.cpp~~ | ~~nsXBLProtoImplMethod.cpp~~ | ~~nsXBLProtoImplProperty.cpp~~ | ~~nsXBLPrototypeHandler.cpp~~ | ~~nsXULElement.cpp~~ | ~~XPCShellEnvironment.cpp~~ | ~~CharacterEncoding.h~~ | ~~AsmJSModule.cpp~~ | ~~Eval.cpp~~ | ~~TestingFunctions.cpp~~ | ~~Parser.cpp~~ | ~~Parser.h~~ | ~~TokenStream.h~~ | ~~gdb-tests.cpp~~ | ~~errorToExceptionEncoding.js~~ | ~~thisFilenameEncoding.js~~ | ~~Object-evalInGlobal-encoding.js~~ | ~~Source-displayURL-encoding.js~~ | ~~display-url-encoding.js~~ | ~~BaselineBailouts.cpp~~ | ~~C1Spewer.cpp~~ | ~~CodeGenerator.cpp~~ | ~~CompileInfo.h~~ | ~~Ion.cpp~~ | ~~JitFrames.cpp~~ | ~~JitSpewer.cpp~~ | ~~MIR.cpp~~ | ~~RematerializedFrame.cpp~~ | ~~testArrayBufferView.cpp~~ | ~~testChromeBuffer.cpp~~ | ~~testCloneScript.cpp~~ | ~~testEnclosingFunction.cpp~~ | ~~testFreshGlobalEvalRedefinition.cpp~~ | ~~testJSEvaluateScript.cpp~~ | ~~testMutedErrors.cpp~~ | ~~testPreserveJitCode.cpp~~ | ~~testScriptInfo.cpp~~ | ~~testScriptObject.cpp~~ | ~~testXDR.cpp~~ | ~~tests.cpp~~ | ~~tests.h~~ | ~~jsapi.cpp~~ | ~~jsapi.h~~ | ~~jsexn.cpp~~ | ~~jsfriendapi.cpp~~ | ~~jsfriendapi.h~~ | ~~jsfun.cpp~~ | ~~jsinfer.cpp~~ | ~~jsobj.cpp~~ | ~~jsopcode.cpp~~ | ~~jsreflect.cpp~~ | ~~jsscript.cpp~~ | ~~jsscript.h~~ | ~~js.cpp~~ | ~~error-encoding.js~~ | ~~reflect-parse.js~~ | ~~CharacterEncoding.cpp~~ | ~~Debugger.cpp~~ | ~~Debugger.h~~ | ~~ErrorObject.cpp~~ | ~~MemoryMetrics.cpp~~ | ~~Probes-inl.h~~ | ~~Probes.cpp~~ | ~~SPSProfiler.cpp~~ | ~~SavedStacks.cpp~~ | ~~SelfHosting.cpp~~ | ~~Stack.cpp~~ | ~~Stack.h~~ | ~~String.h~~ | ~~TraceLogging.cpp~~ | ~~mozJSComponentLoader.cpp~~ | ~~mozJSSubScriptLoader.cpp~~ | ~~Sandbox.cpp~~ | ~~XPCComponents.cpp~~ | ~~XPCConvert.cpp~~ | ~~XPCJSRuntime.cpp~~ | ~~XPCShellImpl.cpp~~ | ~~nsXPConnect.cpp~~ | ~~chrome.ini~~ | ~~test_xrayLogEncoding.xul~~ | ~~file_bug987069.js~~ | ~~mochitest.ini~~ | ~~test_bug987069.html~~ | ~~XrayWrapper.cpp~~ | ~~ProxyAutoConfig.cpp~~ | ~~ThreadStackHelper.cpp~~

Draft

```
# HG changeset patch
# User Tom Tromey <tromey@mozilla.com>

Bug 987069 - make ScriptSource filename encoding consistent
```

```
Largely fine, but large enough that the remainder is also kinda large.  PLEASE, for the love of all that is holy, post subsequent patches as separate patches that apply atop the base patch.  It was close to a
week to review this; I'd rather repeat as little of that experience as possible.  :-)

I see a couple relatively pervasive assumptions here:

* all __FILE__ paths are pure ASCII, or at least UTF-8-encoded (not specified by C++)

On Linux I think this is true these days.  Research suggests it's not entirely the case on OS X, tho.  And it's just not at all on Windows.  <http://stackoverflow.com/questions/23285759/fopen-file-name-
with-utf8-string-in-windows>  However, it seems like most of these uses are going to refer to paths to source files, in a source tree -- and we already require various things about where one puts a source tree
when building.  And probably the only way to deal with this is using a full-fledged file abstraction that understands native path encodings, along the lines of nsILocalFile.  So this is probably good enough
for now, I guess.  But I won't be surprised to find out that requiring UTF-8 here breaks something.

* fprintf(std{err,out}) will interpret output as UTF-8

Most Linux distros will handle UTF-8, and probably OS X does now, and these are probably mostly debugging uses, so what the heck.  We probably assume this a whole bunch of other places, what's one more.
```

dom/base/nsJSUtils.cpp

```
 37 nsJSUtils::GetCallingLocation(JSContext* aContext, nsACString& aFilename,
 38                               uint32_t* aLineno)
```

Please add a comment by the declaration of this function indicating that |aFilename| will be filled with a UTF-8 encoding of the string.

dom/promise/PromiseCallback.cpp

```
248     Promise* returnedPromise;
249     nsresult r = UNWRAP_OBJECT(Promise, valueObj, returnedPromise);
250
251     if (NS_SUCCEEDED(r) && returnedPromise == mNextPromise) {
252       JS::ConstUTF8CharsZ fileName;
```

= JS::ConstUTF8CharsZ("") so as to not possibly pass nullptr to JS_NewStringCopyUTF8Z.

dom/promise/tests/test_encoding.html

```
 19 var Cu = Components.utils;
 20
 21 function runTest() {
 22   // Anything outside of ASCII will do here.
 23   var URL = "file:///whee\u2708.js";
```

Please don't use the same URL for the sandbox principal, and for the eval code. They should be different so we know we're testing the right datum -- namely, the argument to evalInSandbox. I suggest

```
var principal = "file:///whee\u2708/";
var URL = principal + "\uD83D\uDEE7.js";
```

or something that puts distinct Unicode stuffs in both the principal *and the URL.

Well, actually: if it were me, I wouldn't use file: URLs for this. The problem with file: URLs is that the way they get mapped to security origins is unspecified across browsers, and our policy, while a relatively sensible one, isn't the only one on the block. (Specifically, our policy makes file: a multitude of origins, one per directory. Hence the necessity for URL here to be within the folder delineated by principal.) It would be better to use an HTTP URL with IDN in it -- say, http://天/ ("http://\u5929/") or http://□/ ("http://\uD808\uDC41/") -- so that you're relying on well-defined semantics here.

And, as with all the other tests, it'd be really good to have them test more than just a single character, but to include non-BMP, multiple UTF-8 character lengths, and combining characters, and so on.

dom/xbl/nsXBLProtoImplMethod.cpp

```
197
198   JSContext *cx = jsapi.cx();
199   JSAutoCompartment ac(cx, aClassObject);
200   JS::CompileOptions options(cx);
201   options.setFileAndLine(JS::ConstUTF8CharsZ(functionUri.get()),
```

This functionUri seems to derive from the same half-UTF-8, half-truncated-UTF-16 data as in nsXBLProtoImplProperty.cpp. Fixing the root problem there, will make this accurate as well.

dom/xbl/nsXBLProtoImplProperty.cpp

```
196     nsDependentString getter(getterText->GetText());
197     if (!getter.IsEmpty()) {
198       JSAutoCompartment ac(cx, aClassObject);
199       JS::CompileOptions options(cx);
200       options.setFileAndLine(JS::ConstUTF8CharsZ(functionUri.get()), getterText->GetLineNumber())
```

functionUri derives from aClassStr, which seems to derive from mClassName, which ultimately appears to derive from one of two places. See NS_NewXBLImpl.

First, most commonly, it could be the binding URI's spec, which is UTF-8-encoded, all good.

Second, it could be the name attribute on the <implementation> element -- <implementation name="XStringBundle"> being the only instance of this in mozilla-central, as far as I can tell. Which could contain anything, not just UTF-8. This DOM-land value is UTF-16, so how is it converted to the nsACString we have here? |impl->mClassName.AssignWithConversion(aClassName);| That corresponds to:

```
void
nsCString::AssignWithConversion( const nsAString& aData )
{
  LossyCopyUTF16toASCII(aData, *this);
}
```

And despite what the name says, "toASCII" is really "toLatin1". So the universal problem of interpreting Latin1 as UTF-8 arises, we have dataloss at least and buffer overread and other considerations at worst.

Given that this value can legitimately be either UTF-8 or not, we need to fix this at the source. Please file a new bug to do a UTF-16 to UTF-8 conversion in NS_NewXBLImpl for the name="" case, blocking this bug. We need to get mClassName to a single consistent encoding before we can proceed with this code. It's drudgery, but not drudgery worse than what you've already done here. The patch and any other related changes should probably be reviewed by an XBL peer. You could also heckle the original author and have jst (!) review it. <http://bonsai.mozilla.org/cvsview2.cgi?diff_mode=context&whitespace_mode=show&root=/cvsroot&subdir=mozilla/content/xbl/src&command=DIFF_FRAMESET&file=nsXBLProtoImpl.cpp&rev2=1.8&rev1=1.7> (This last suggestion is mostly unserious, although I will probably communicate his transgressions to him the next time I see him around the office. ;-))

```
242     nsDependentString setter(setterText->GetText());
243     if (!setter.IsEmpty()) {
244       JSAutoCompartment ac(cx, aClassObject);
245       JS::CompileOptions options(cx);
246       options.setFileAndLine(JS::ConstUTF8CharsZ(functionUri.get()), setterText->GetLineNumber())
```

Reusing the same inconsistently-encoded string, so the above fix deals with this too.

ipc/testshell/XPCShellEnvironment.cpp

```
154         if (!str)
```

```
155             return false;
156         JSAutoByteString filename;
157         filename.encodeUtf8(cx, str);
158         if (!filename)
```

if (!filename.encodeUtf8...)

js/public/CharacterEncoding.h

```
109     char *c_str() { return reinterpret_cast<char *>(get()); }
110 };
111
112 /* Similar to UTF8CharsZ, but the chars are const, and allows
113  * assignment.  */
```

Either use //-style comments here, or do

```
/*
 * Similar to UTF8CharsZ, ...
 * assignment.
 */
```

for style.

```
112 /* Similar to UTF8CharsZ, but the chars are const, and allows
113  * assignment.  */
114 class ConstUTF8CharsZ
115 {
116     const char *mData;
```

SpiderMonkey doesn't m/a/s-prefix names, so this should just be data or something.

```
119     ConstUTF8CharsZ() : mData(nullptr)
120     {
121     }
122
123     explicit ConstUTF8CharsZ(const char *aBytes)
```

and this bytes

```
122
123     explicit ConstUTF8CharsZ(const char *aBytes)
124       : mData(aBytes)
125     {
126     }
```

While it's certainly *convenient* to just make this a 100% lightweight wrapper around a const char*, I think this is highly likely to make it easy to have misuses, passing Latin1 strings into the constructor. I think we should add some assertions here that check that the input string is not immediately obviously not UTF-8, by checking some prefix of the string to see whether it's valid UTF-8 or not.

My eyes glazed over a bit reading this patch, so I wrote some code that can be used to sanity-check incoming strings here -- and anywhere else that's supposed to take exactly UTF-8 input (although in general we want to throw the C++ type system at such places). I'll upload it here once I've posted this review.

```
128     const void *get() const { return mData; }
129
130     const char *c_str() const { return mData; }
131
132     operator bool() const { return mData != nullptr; }
```

explicit operator bool() -- otherwise you can do bizarre things like 1 + ConstUTF8CharsZ() and it'll go through this conversion (!).

```
220 /*
221  * Like UTF8CharsToNewTwoByteCharsZ, but for ConstUTF8CharsZ.
222  */
223 extern TwoByteCharsZ
224 UTF8CharsToNewTwoByteCharsZ(JSContext *cx, const ConstUTF8CharsZ &utf8, size_t *outlen);
```

Why can't these new methods return TwoByteChars, as a single data structure pairing pointer and length? Done this way users can have pointer and length decohere from each other, which seems bad.

I think we should fully duplicate the comment by the other method here, with appropriate changes. Better to duplicate, and be absolutely precise about behavior, to help readability, than to cut a corner and save on code. Readability trumps code size here.

```
231 extern TwoByteCharsZ
232 LossyUTF8CharsToNewTwoByteCharsZ(JSContext *cx, const UTF8Chars utf8, size_t *outlen);
233
234 extern TwoByteCharsZ
235 LossyUTF8CharsToNewTwoByteCharsZ(JSContext *cx, const ConstUTF8CharsZ &utf8, size_t *outlen);
```

Same comments as above -- we should probably duplicate comments a bit harder.

js/src/builtin/TestingFunctions.cpp

```
1814
1815     char *buf = JS::FormatStackDump(cx, nullptr, showArgs, showLocals, showThisProps);
1816     RootedString str(cx);
1817     if (!(str = JS_NewStringCopyZUTF8(cx, JS::ConstUTF8CharsZ(buf))))
1818         return false;
```

The presence of |showArgs| there is a tell: if you can include arguments, that probably means you can include strings, and that means you have to deal with the full Unicode gamut. Which means either |buf| is UTF-8 and we're okay, *or* it's Latin1 and we're not. It turns out we're not okay here. (I am shocked, shocked I say!)

js> ((a) => { return getBacktrace({ args: true }); })("\u99BA\u99DB\u99DB\u99EE\u99F1")
"0 anonymous(a = \"\xBA\xDB\xDB\xEE\xF1\") [\"typein\":15]\n1 <TOP LEVEL> [\"typein\":15]\n"

Notice how our careful, pretty, shiny (...ish) 99s there disappeared to leave us with BADBEEF1. Heck, come to think of it, you can give functions non-Latin1 names, too, to produce the same effect, or just mess with people:

js> (function \u1242\u1245\u1245\u1246() { return getBacktrace(); })()
"0 BEEF() [\"typein\":7]\n1 <TOP LEVEL> [\"typein\":7]\n"

But. Later comments on FormatStackDump in this review suggest that you "changed" FormatStackDump to be UTF-8 when file names were inserted in it. So the change here is okay -- but, you need to audit and fix the rest of FormatStackDump to produce UTF-8, for this bit of this to be correct. Please do so.

js/src/jit-test/tests/basic/errorToExceptionEncoding.js

```
1 // Test that encoding of file name is preserved by thisFilename.
2
3 // Anything outside of ASCII will do.
4 let name = "file:///whee\u2708.js";
```

Same multiplicity of things comment as I made elsewhere about BMP/non-BMP and combining characters and all that.

js/src/jit-test/tests/basic/thisFilenameEncoding.js

```
1 // Test that encoding of file name is preserved by thisFilename.
2
3 // Anything outside of ASCII will do.
4 let name = "file:///whee\u2708.js";
```

While true, it seems like it'd be good to have both BMP and non-BMP versions here. Also we should consider combining characters, with BMP and with non-BMP characters (are there composed forms that are non-BMP?) -- check that composed forms, decomposed forms, all that jazz are all left (I would assume, correct me if I'm wrong!) as-is by all our code.

js/src/jit-test/tests/debug/Object-evalInGlobal-encoding.js

```
5 var dbg = new Debugger;
6 var gw = dbg.addDebuggee(g);
7
8 // Anything outside of ASCII will do.
9 let name = "file:///whee\u2708.js";
```

Same non-BMP, combining characters gamut.

js/src/jit-test/tests/debug/Source-displayURL-encoding.js

```
4 let dbg = new Debugger();
5 let gw = dbg.addDebuggee(g);
6
7 // Anything outside of ASCII will do.
8 let name = "file:///whee\u2708.js";
```

Same non-BMP, combining characters gamut.

js/src/jit-test/tests/saved-stacks/display-url-encoding.js

```
1 // Test that saved stacks don't mangle the encoding of the source.
2
3 // Anything outside of ASCII will do.
4 let name = "file:///whee\u2708.js";
```

Same non-BMP, combining characters gamut.

js/src/jit/BaselineBailouts.cpp

```
1145               cx->runtime()->spsProfiler.updatePC(script, script->code());
1146           }
1147
1148           // Register bailout with profiler.
1149           const char *filename = script->filename().c_str();
```

It's far from clear that it's okay for this UTF-8 string to flow into the uses below. Please add comments by SPSProfiler::setEventMarker to clarify that the provided function is invoked with a UTF-8 string, and by js::RegisterRuntimeProfilingEventMarker that calls it.

As for whether all the different markers are fine.

It looks like &ProfilerJSEventMarker from toolkit/profiler/PseudoStack.h is okay -- that spews through a JSStreamWriter that treats the string as UTF-8.

And it looks like &PrintProfilerEvents_Callback from the JS shell is also fine, as it just fprintfs.

So I think this is just a doc fix, no more.

js/src/jit/C1Spewer.cpp

```
34
35     fprintf(spewout_, "begin_compilation\n");
36     if (script) {
37         fprintf(spewout_, "  name \"%s:%d\"\n", script->filename().c_str(), (int)script->lineno());
38         fprintf(spewout_, "  method \"%s:%d\"\n", script->filename().c_str(), (int)script->lineno());
```

While you're touching this, mind switching the linenos to PRIuSIZE in mozilla/SizePrintfMacros.h and removing casts?

js/src/jit/CodeGenerator.cpp

```
3770 #ifdef DEBUG
3771         const char *filename = nullptr;
3772         unsigned lineNumber = 0, columnNumber = 0;
3773         if (current->mir()->info().script()) {
3774             filename = current->mir()->info().script()->filename().c_str();
```

So I went to search for filename() in this file to double-check context, and I discovered at least one, probably more, calls to filename() that are then passed as arguments to JitSpew, then JS_snprintf, and maybe others -- ellipsis functiosn. This doesn't really work, as non-POD objects can't be passed to ellipsis notation per C++11 (a DR adjusted this to allow compiler-dependent behavior, but extracting out a const char* for an argument not of that type probably is UB as well).

clang at least has a warning/error if you do this -- I suggest compiling a debug build with clang and seeing where you hit errors, then fixing them in a *separate* patch atop this one. (I'm already well into this patch, would rather not discard my work.)

```
7187         code->setHasBytecodeMap();
7188     }
7189
7190     if (cx->runtime()->spsProfiler.enabled()) {
7191         const char *filename = script->filename().c_str();
```

This flows into JitSpew which seems, laboriously, to flow into fprintf, so is okay. Right?

js/src/jit/JitFrames.cpp

```
2492         fprintf(stderr, "  global frame, no callee\n");
2493     }
2494
2495     fprintf(stderr, "  file %s line %u\n",
2496             script()->filename().c_str(), (unsigned) script()->lineno());
```

Again use PRIuSIZE here, since you're touching it.

```
2541         fprintf(stderr, "  global frame, no callee\n");
2542     }
2543
2544     fprintf(stderr, "  file %s line %u\n",
2545             script()->filename().c_str(), (unsigned) script()->lineno());
```

And here.

js/src/jit/RematerializedFrame.cpp

```
166         fprintf(stderr, "  global frame, no callee\n");
167     }
168
169     fprintf(stderr, "  file %s line %u offset %zu\n",
170             script()->filename().c_str(), (unsigned) script()->lineno(),
```

Change "%zu" to "%" PRIuSIZE with SizePrintfMacros.h, same as mentioned elsewhere. Whoever wrote this must not have realized that MSVC doesn't support any %z specifiers. :-(

js/src/jsapi-tests/testArrayBufferView.cpp

```
93     static const char code[] = "new DataView(new ArrayBuffer(8))";
94
95     JS::RootedObject global(cx);
96     JS::CompileOptions opts(cx);
97     if (!JS::Evaluate(cx, global, opts.setFileAndLine(JS::ConstUTF8CharsZ(__FILE__), __LINE__),
```

Hmm. Nothing guarantees a particular charset for __FILE__. But I'm not sure there's really anything we can do about that in any event. :-\

js/src/jsapi.cpp

```
3940 JS::OwningCompileOptions::setFile(JSContext *cx, const ConstUTF8CharsZ &f)
3941 {
3942     char *copy = nullptr;
3943     if (f) {
3944         copy = JS_strdup(cx, f.c_str());
```

I can't remember my C++, we can make that |const char *copy| (and assign T* into const T*) to make clear we're never writing into its contents, right? Please do so if it compiles.

```
3982
3983 bool
3984 JS::OwningCompileOptions::setIntroducerFilename(JSContext *cx, const ConstUTF8CharsZ &s)
3985 {
3986     char *copy = nullptr;
```

Again const char* if you can.

```
4716 {
4717     AssertHeapIsIdle(cx);
4718     CHECK_REQUEST(cx);
4719     if (!s || !*(s.c_str()))
4720         return cx->runtime()->emptyString;
```

MOZ_ASSERT(s.get(), "null chars passed to JS_NewStringCopyUTF8Z");

And for the other,

```
if (s.c_str()[0] == '\0')
  return cx->runtime()->emptyString;
```

seems a little more readable to me.

I see the convert-null-to-empty-string behavior is what JS_NewStringCopyZ does, but that seems like a bad API. We should get away from nullptr having that meaning in new APIs, rather complaining early and often about what seems frequently likely to be a mistake.

js/src/jsapi.h

```
3613 extern JS_PUBLIC_API(JSObject *)
3614 JS_GetGlobalFromScript(JSScript *script);
3615
3616 extern JS_PUBLIC_API(JS::ConstUTF8CharsZ)
3617 JS_GetScriptFilename(JSScript *script);
```

I guess this is no less clear about ownership than the existing method is. Alas, but okay.

```
3695    // purpose.
3696    ReadOnlyCompileOptions()
3697      : mutedErrors_(false),
3698        filename_(),
3699        introducerFilename_(),
```

Remove these, it's implied by C++ and seems nicer to not say anything when null-initializing is the obvious expected behavior for a smart-ish pointer.

```
3805    /* These setters make copies of their string arguments, and are fallible. */
3806    bool setFile(JSContext *cx, const ConstUTF8CharsZ &f);
3807    bool setFileAndLine(JSContext *cx, const ConstUTF8CharsZ &f, unsigned l);
3808    bool setSourceMapURL(JSContext *cx, const char16_t *s);
3809    bool setIntroducerFilename(JSContext *cx, const ConstUTF8CharsZ &s);
```

We should add UniquePtr-style versions of these at some point so that callers that have just created a string don't have to have it copied again. I wonder if that'd need bug 1035966 (which is doable now that we have actual nullptr on all supported compilers), or if we would want to use some mechanism not *exactly* UniquePtr for that...

Or there's that "make this all JSString*" approach that seems most ideal to me, and doesn't UniquePtr anything, but requires making this stuff GC-correct, which is non-trivial enough to certainly be deferrable past fixing the issue here. ψ.

```
3963        JS::MutableHandleScript script);
3964
3965 extern JS_PUBLIC_API(bool)
3966 Compile(JSContext *cx, JS::HandleObject obj, const ReadOnlyCompileOptions &options, const JS::ConstUTF8CharsZ &filename,
3967         JS::MutableHandleScript script);
```

Line-wrapping point is at 99ch, so this is far over-long and needs rewrapping.

```
4240 extern JS_PUBLIC_API(JSString *)
4241 JS_NewStringCopyZ(JSContext *cx, const char *s);
4242
4243 extern JS_PUBLIC_API(JSString *)
4244 JS_NewStringCopyZUTF8(JSContext *cx, const JS::ConstUTF8CharsZ &s);
```

I think JS_NewStringCopyUTF8Z seems more consistent with the type name. The Z is a terminator -- it belongs at end of the name. And logically, we're accepting UTF-8 data, that's null-terminated. That lines up best with *UTF8Z for the name.

js/src/jsexn.cpp

```
296                     return nullptr;
297             }
298             /* Second, try the actual filename. */
299             else if (const char *filename = i.scriptFilename().c_str()) {
300                 if (!sb.append(filename, strlen(filename)))
```

This treats the filename as Latin1, not as Unicode. Should be visible in new Error().stack, very clearly. This should have a test for it, assuming (as seems likely) that it has none now.

Let's add (in a separate bug/patch, blocking this bug) StringBuffer::appendUTF8(const char*) and StringBuffer::appendUTF8(JS::ConstUTF8CharsZ) methods. For starters just have them use InflateUTF8StringToBuffer in the stupidest way possible, into a buffer of 16-bit characters, then append that. We can have another bug after that to do something cleaner, that doesn't require extra temporary storage.

Also please file a bug to rename StringBuffer::append(const char*) to StringBuffer::appendLatin1(const char*). That rename can (and *should*) happen after all this bug and patchwork wraps up.

```
721
722             if (reportp && reportp->filename) {
723                 filename = strrchr(reportp->filename.c_str(), '/');
724                 if (filename)
725                     filename++;
```

I guess we're going to just hope that addons don't have too many Unicode-named files, because we're changing the histogram key we're using here. Had to happen eventually, I guess. And maybe this is purely for our own edification, and we can suck up the change on our own.

js/src/jsfriendapi.cpp

```
701
702     RootedObject scopeChain(cx, iter.scopeChain(cx));
703     JSAutoCompartment ac(cx, scopeChain);
704
705     const char *filename = script->filename().c_str();
```

Oh blah, this is why you changed that one use of this in TestingFunctions.cpp. Bleargh. Congratulations on getting to audit all the code used by FormatStackDump to make it all consistently UTF-8! :-\ We can't have it be half-Latin1, half-UTF-8, because then it's neither.

js/src/jsfriendapi.h

```
415      * Set |*src| and |*length| to refer to the source code for |filename|.
416      * On success, the caller owns the buffer to which |*src| points, and
417      * should use JS_free to free it.
418      */
419     virtual bool load(JSContext *cx, const JS::ConstUTF8CharsZ &filename, char16_t **src, size_t *length) = 0;
```

Hmm, at some point we should make this return some sort of TwoByteChars containing a UniquePtr. Or make it return a JSString* if we can get all these filename bits to use that representation instead. Not now, of course.

js/src/jsinfer.cpp

```
219 unsigned JSScript::id() {
220     if (!id_) {
221         id_ = ++compartment()->types.scriptCount;
222         InferSpew(ISpewOps, "script #%u: %p %s:%d",
223                   id_, this, filename() ? filename() : JS::ConstUTF8CharsZ("<null>"), lineno());
```

PRIuSIZE, and fix the ridiculously non-standard formatting while you're here:

unsigned
JSScript::id()
{
  if (!id_) {
...

Oh, this also looks like a place that's passing a non-POD to varargs.

```
5226     else if (script->isForEval())
5227         fprintf(stderr, "Eval");
5228     else
5229         fprintf(stderr, "Main");
5230     fprintf(stderr, " #%u %s:%d ", script->id(), script->filename().c_str(), (int) script->lineno());
```

PRIuSIZE

js/src/jsobj.cpp

```
3767         }
3768         if (fun->hasScript()) {
3769             JSScript *script = fun->nonLazyScript();
3770             fprintf(stderr, " (%s:%d)",
3771                     script->filename() ? script->filename().c_str() : "", (int) script->lineno());
```

Fix the lineno() bit to PRIuSIZE while you're here?

```
4004         }
4005         fputc('\n', stderr);
4006
4007         fprintf(stderr, "file %s line %u\n",
4008                 i.script()->filename().c_str(), (unsigned) i.script()->lineno());
```

And here.

```
4050                         depth, (i.isJit() ? 0 : i.interpFrame()), filename, line,
4051                         script, script->pcToOffset(i.pc()));
4052     }
4053     fprintf(stdout, "%s", sprinter.string());
4054 #ifdef XP_WIN32
```

Immediately below this is

  OutputDebugStringA(sprinter.string());

which won't handle UTF-8. On the other hand. <https://msdn.microsoft.com/en-us/library/windows

/desktop/aa363362%28v=vs.85%29.aspx> says that the method won't handle all Unicode characters, so we lose no matter what we do. So that's good enough -- but please add a comment saying that even OutputDebugStringW won't handle all Unicode characters, so we're just going to not care that this method drops even more of them.

js/src/jsopcode.cpp

```
292
293             fprintf(stdout, "--- SCRIPT %s:%d ---\n", script->filename().c_str(), (int) script->lineno());
294             js_DumpPCCounts(cx, script, &sprinter);
295             fputs(sprinter.string(), stdout);
296             fprintf(stdout, "--- END SCRIPT %s:%d ---\n", script->filename().c_str(), (int) script->lineno());
```

Make these both PRIuSIZE as well.

```
2143
2144     AppendJSONProperty(buf, "file", NO_COMMA);
2145     JSString *str = NewStringCopyZUTF8<CanGC>(cx, script->filename());
2146     if (!str || !(str = StringToSource(cx, str)))
2147         return nullptr;
```

Make the declaration here a RootedString, please. I'm mildly surprised there's no rooting hazard flagged here.

js/src/jsreflect.cpp

```
3422     RootedString src(cx, ToString<CanGC>(cx, args[0]));
3423     if (!src)
3424         return false;
3425
3426     ScopedJSFreePtr<char> filenameChars;
```

Change this to UniquePtr<char[], JS::FreePolicy> while you're here. You'll have to change the assignment below to a .reset(...), but otherwise it should all be identical.

```
3424         return false;
3425
3426     ScopedJSFreePtr<char> filenameChars;
3427     RootedValue filename(cx);
3428     filename.setNull();
```

Instead of |filename| as RootedValue, make it RootedString. |RootedString filename(cx);| will be initialized to nullptr, then you can make the argument type HandleString, then use |if (str)| to determine whether to atomize and set the member variable.

```
3464                 RootedString str(cx, ToString<CanGC>(cx, prop));
3465                 if (!str)
3466                     return false;
3467
3468                 filename.setString(str);
```

filename as RootedString lets us just ToString directly into it and eliminate the |str| here.

js/src/jsscript.cpp

```
2038         char *formatted = FormatIntroducedFilename(cx, filename, options.introductionLineno,
2039                                                    options.introductionType);
2040         if (!formatted)
2041             return false;
2042         filename_.reset(formatted);
```

Add a comment by filename_ indicating that its contents are UTF-8, please.

js/src/shell/js.cpp

```
849             return false;
850         }
851         JSAutoByteString filename;
852         filename.encodeUtf8(cx, str);
853         if (!filename)
```

```
if (!filename.encodeUtf8(cx, str))
    return false;
```

```
913             return false;
914         char *fileName = fileNameBytes.encodeUtf8(cx, s);
915         if (!fileName)
916             return false;
917         options.setFile(JS::ConstUTF8CharsZ(fileName));
```

```
if (!fileNameBytes.encodeUtf8(cx, s))
    return false;
options.setFile(JS::ConstUTF8CharsZ(fileNameBytes.ptr()));
```

```
1459         return false;
1460     args[0].setString(str);
1461     JSAutoByteString filename;
1462     filename.encodeUtf8(cx, str);
1463     if (!filename)
```

Same

```
JSAutoByteString filename;
if (!filename.encodeUtf8(cx, str))
    return false;
```

dance.

```
1478     int64_t startClock = PRMJ_Now();
1479     {
1480         JS::CompileOptions options(cx);
1481         options.setIntroductionType("js shell run")
1482                .setFileAndLine(JS::ConstUTF8CharsZ(filename.ptr()), 1)
```

I don't believe this is right -- above filename is constructed using (JSContext*, HandleString), which calls JS_EncodeString, which does EncodeLatin1. Make that

```
JSAutoByteString filename;
if (!filename.encodeUtf8(cx, str))
    return false;
```

instead, then this will be okay.

```
2357         return false;
2358     JSAutoByteString filename;
2359     filename.encodeUtf8(cx, str);
2360     if (!filename)
2361         return false;
```

Same

```
JSABS filename;
if (!...)
    return false;
```

nit.

js/src/tests/js1_5/Exceptions/error-encoding.js

```
2
3 // Test that Error.fileName is consistently encoded.
4
5 // Anything outside of ASCII will do here.
6 var URL = "file:///whee\u2708.js";
```

Same non-BMP, combining characters stuff.

js/src/tests/js1_8_5/extensions/reflect-parse.js

```
1050
1051 Pattern({ source: "quad.js", start: { line: 1, column: 20 }, end: { line: 1, column: 29 } }).match(fourAC.loc);
1052
1053 // Bug 987069: encoding of the URL.  Anything outside of ASCII will do.
1054 const URL = "file:///whee\u2708.js"
```

Non-BMP, combining characters.

js/src/vm/CharacterEncoding.cpp

```
327
328     return true;
329 }
330
331 template <InflateUTF8Action action>
```

Capitalize Action, as a template parameter name, please. Not universal, but very very close to it.

js/src/vm/Debugger.cpp

```
3390        /* Compute urlCString and displayURLChars, if a url or displayURL was
3391         * given respectively. */
3392        if (url.isString()) {
3393            RootedString str(cx, url.toString());
3394            if (!urlCString.encodeUtf8(cx, str))
```

Please adjust the comment by urlCString, too:

```
/* url as a UTF-8 C string. */
JSAutoByteString urlCString;
```

or something along those lines. This is another place where JSString* would make things oh so much simpler. But for later.

```
4224        if (script->scriptSource()->introducerFilename())
4225            filename = script->scriptSource()->introducerFilename();
4226        else
4227            filename = script->filename();
4228        str = NewStringCopyZUTF8<CanGC>(cx, filename);
```

Move the declaration down to here, and put a blank line above this, for breathing space.

```
6013        if (!v.isUndefined()) {
6014            RootedString url_str(cx, ToString<CanGC>(cx, v));
6015            if (!url_str)
6016                return false;
6017            url = url_bytes.encodeUtf8(cx, url_str);
```

This |url| variable seems to be largely someone's failure to understand that JSAutoByteString has a ptr() method to get the string. Please remove the |url| variable, rename |url_bytes| to |url|, and just use |url.ptr()| further down in this method.

Oh, make it

```
if (!url_bytes.encodeUtf8(cx, url_str))
    return false;
```

as there's no need for an assignment, then null-checking that variable.

js/src/vm/MemoryMetrics.cpp

```
449
450           rtStats->runtime.scriptSourceInfo.add(info);
451
452           if (granularity == FineGrained) {
453               const char* filename = ss->filename().c_str();
```

Can you propagate ConstUTF8CharsZ down into the type of the allScriptSources map here, and NotableStringInfo implicated by it? Everything looks fine in the users as far as encoding goes (nsIMemoryReporterCallback::callback accepts an AUTF8String path), but better to be more explicit about it.

js/src/vm/Probes.cpp

```
46    if (!script)
47        return probes::nullName;
48    if (!script->filename())
49        return probes::anonymousName;
50    return script->filename().c_str();
```

I'm going to pretend that DTrace is fine accepting UTF-8 here, because cursory anecdotal searching suggests DTrace is all-UTF-8 for string arguments. If that's wrong, someone else can determine it at that time, given DTrace is relatively underused last I knew.

js/src/vm/SPSProfiler.cpp

```
272    // Get the function name, if any.
273    JSAtom *atom = maybeFun ? maybeFun->displayAtom() : nullptr;
274
275    // Get the script filename, if any, and its length.
276    const char *filename = script->filename().c_str();
```

Please convert this to a ConstUTF8CharsZ. This use flows onward to a bunch of random code that would be much clearer with a better type.

As for whether all that code is copacetic with this. This is unclear to me. This flows into ProfileEntry structs and a bunch of other places, including Breakpad code -- that largely doesn't examine it for encoding purposes, but I may have missed something. Please ask someone familiar with Breakpad to review all the uses of ProfileEntry::label() to ensure they're all fine accepting UTF-8 data. The use in tools/profiler /BreakpadSampler.cpp is the one that has me most potentially worried about introducing UTF-8 here, where it might possibly have not been understood before.

js/src/vm/SavedStacks.cpp

```
679       if (const char16_t *displayURL = iter.scriptDisplayURL()) {
680           locationp->source = AtomizeChars(cx, displayURL, js_strlen(displayURL));
681       } else {
682           JS::ConstUTF8CharsZ filename = iter.scriptFilename();
683           if (filename) {
```

if (JS::ConstUTF8CharsZ filename = iter.scriptFilename()) {

```
687                   return false;
688               locationp->source = AtomizeChars(cx, chars.get(), len);
689               js_free(chars.get());
690           } else {
691               locationp->source = Atomize(cx, "", 0);
```

You can just assign cx->names().empty at this point.

```
709       if (const char16_t *displayURL = iter.scriptDisplayURL()) {
710           source = AtomizeChars(cx, displayURL, js_strlen(displayURL));
711       } else {
712           JS::ConstUTF8CharsZ filename = script->filename();
713           if (filename) {
```

Same if (...) { as above.

```
713           if (filename) {
714               size_t len;
715               // We use the lossy variant here because it cannot GC.
716               // This avoids a rooting hazard for 'key'.
717               JS::TwoByteCharsZ chars = LossyUTF8CharsToNewTwoByteCharsZ(cx, filename, &len);
```

I don't particularly understand this. Why is it okay to be lossy here, versus sucking it up to always be correct, even if extra work is required to keep |key| in acceptable shape about GC hazards? I don't understand this well enough to propose the solution, myself -- but we should have one. Pretty sure it's not okay to randomly lose data here.

```
719                   return false;
720               source = AtomizeChars(cx, chars.get(), len);
721               js_free(chars.get());
722           } else {
723               source = Atomize(cx, "", 0);
```

And cx->names().empty again.

js/src/vm/SelfHosting.cpp

```
1111    char *filename = getenv("MOZ_SELFHOSTEDJS");
1112    if (filename) {
1113        RootedScript script(cx);
1114        if (Compile(cx, shg, options, JS::ConstUTF8CharsZ(filename), &script))
1115            ok = Execute(cx, script, *shg.get(), rv.address());
```

This is an environment variable, so it could have unknown encoding. But it's also a debugging facility, so this is probably good enough. Add a comment here noting that we're okay with it for this reason.

js/src/vm/TraceLogging.cpp

```
459 TraceLoggerEventPayload *
460 TraceLoggerThread::getOrCreateEventPayload(TraceLoggerTextId type,
461                                            const JS::ReadOnlyCompileOptions &script)
462 {
463     return getOrCreateEventPayload(type, script.filename().c_str(), script.lineno, script.column, &script);
```

I don't think this works.

The filename here is inserted into a TraceLoggerEventPayload, that's inserted into an extraTextId hash. That's consulted by TraceLoggerThread::eventText. And that's used by TraceLoggerThread::extractScriptDetails which exposes that text, which is used at least by

Debugger::drainTraceLoggerScriptCalls that passes the exposed const char* to JSAPI as if it were Latin-1 strings. (I haven't tracked any other uses of this bit of use of the string.)

And later in gOCEP, the string created here is interpolated into |str| passed in |graph->addTextId(textId, str);|. That method does |js::FileEscapedString(dictFile, text, strlen(text), '"')| with |text == str|. And FileEscapedString doesn't understand UTF-8 input.

As for how to deal with this. Um. I guess track harder into users/callers? And probably someone who understands TraceLogger code should review this portion of changes, to be sure the utomst extents have been correctly changed. And probably ConstUTF8CharsZ should be used as a type in at least some of these data structures here.

js/xpconnect/src/XPCShellImpl.cpp

```
321             return false;
322         }
323         JS::CompileOptions options(cx);
324         options.setUTF8(true)
325                 .setFileAndLine(JS::ConstUTF8CharsZ(filename.ptr()), 1)
```

JSABS filename;
if (!filename.encodeUtf8(cx, str))
  return false;

above this, else this is wrong.

```
824         JS_BeginRequest(cx);
825
826         JS::CompileOptions options(cx);
827         options.setUTF8(true)
828                 .setFileAndLine(JS::ConstUTF8CharsZ(filename), 1)
```

...so I guess we're also assuming that commandline arguments (or at least paths) are UTF-8 as well. Given this is mostly for in-tree paths, or paths to files in trees, or test harness stuff, okay. But put a comment by this to clarify that assumption.

js/xpconnect/tests/chrome/test_xrayLogEncoding.xul

```
 45  };
 46
 47  theconsole.registerListener(listener);
 48
 49  var myURL = "file:///whee+.js";
```

As usual it'd be really really really nice to test a larger variety of URLs with non-BMP, combining characters, etc. here.

js/xpconnect/tests/mochitest/test_bug987069.html

```
 12
 13  /** Test for Bug 987069 **/
 14
 15 var theurl = "http://sub1.ält.example.org:8000/tests/js/xpconnect/tests/mochitest/file_bug987069.js";
 16 SimpleTest.waitForExplicitFinish();
```

Again fine, just test a few more domains, please. If you need more -- I don't think we have any that are non-BMP yet, or that contain combining characters -- you can add them to build/pgo/server-locations.txt, as subdomains underneath example.com.

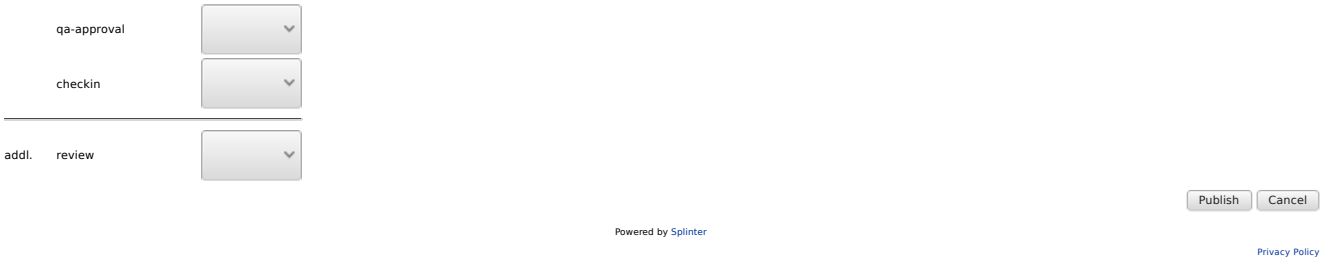
xpcom/threads/ThreadStackHelper.cpp

```
584  MOZ_ASSERT(aEntry->script());
585
586  const char* label;
587  if (IsChromeJSScript(aEntry->script())) {
588    const char* filename = JS_GetScriptFilename(aEntry->script()).c_str();
```

It looks to me like we append this to the stack vector at the end of this function. As far as I can tell, that vector gets used -- at an absolute minimum -- by toolkit/components/telemetry/Telemetry.cpp:CreateJSHangStack, which interprets each element of the vector as a Latin1 string. So I think this is okay as far as this *immediate* code is concerned, but I think you need to track down all the consumers of |mStackToFill| and ensure they all are comfortable with UTF-8 input and won't mangle it into mojibake.

Please get a peer from this code, and probably from telemetry as well, to review the relevant portions of the additional changes you'll need to make to address this. I'd like to look, but ultimately this particular UTF-8 string flows into a very wide-open set of uses that I can't easily identify and, therefore, that I'm not truly competent to review.

Flags:	Requestee:
ttromey: review	–
superreview	
a11y-review	
ui-review	
feedback	
approval-mozilla-aurora	
approval-mozilla-beta	
approval-mozilla-release	
approval-mozilla-esr31	
approval-mozilla-b2g30	
approval-mozilla-b2g32	
approval-mozilla-b2g34	
approval-mozilla-b2g37	
sec-approval	


qa-approval
checkin
addl.
review
Publish
Cancel
Powered by Splinter
Privacy Policy