```
/* Copyright 2016 The Chromium
Authors. All Rights Reserved.
 *
 * Use of this source code is
governed by a BSD-style
 * license that can be found
in the LICENSE file or at
 *
https://developers.google.com/
open-source/licenses/bsd
 */

// ----------------------------
------------------------------
---------------
// This file contains common
utilities and basic javascript
infrastructure.
//
// Notes:
// * Press 'D' to toggle debug
mode.
//
// Functions:
//
// - Assertions
// DEPRECATED: Use assert.js
// AssertTrue(): assert an
expression. Throws an
exception if false.
// Fail(): Throws an
exception. (Mark block of code
that should be unreachable)
// AssertEquals(): assert that
two values are equal.
// AssertType(): assert that a
value has a particular type
//
// - Cookies
// SetCookie(): Sets a cookie.
// ExpireCookie(): Expires a
cookie.
// GetCookie(): Gets a cookie
value.
//
// - Dynamic HTML/DOM
utilities
```

```
// MaybeGetElement(): get an
element by its id
// GetElement(): get an
element by its id
// GetParentNode(): Get the
parent of an element
// GetAttribute(): Get
attribute value of a DOM node
// GetInnerHTML(): get the
inner HTML of a node
// SetCssStyle(): Sets a CSS
property of a node.
// GetStyleProperty(): Get CSS
property from a style
attribute string
// GetCellIndex(): Get the
index of a table cell in a
table row
// ShowElement(): Show/hide
element by setting the
"display" css property.
// ShowBlockElement():
Show/hide block element
// SetButtonText(): Set the
text of a button element.
// AppendNewElement(): Create
and append a html element to a
parent node.
// CreateDIV(): Create a DIV
element and append to the
document.
// HasClass(): check if
element has a given class
// AddClass(): add a class to
an element
// RemoveClass(): remove a
class from an element
//
// - Window/Screen utiltiies
// GetPageOffsetLeft(): get
the X page offset of an
element
// GetPageOffsetTop(): get the
Y page offset of an element
// GetPageOffset(): get the X
and Y page offsets of an
element
```

```
// GetPageOffsetRight() : get
X page offset of the right
side of an element
// GetPageOffsetRight() : get
Y page offset of the bottom of
an element
// GetScrollTop(): get the
vertical scrolling pos of a
window.
// GetScrollLeft(): get the
horizontal scrolling pos of a
window
// IsScrollAtEnd():  check if
window scrollbar has reached
its maximum offset
// ScrollTo(): scroll window
to a position
// ScrollIntoView(): scroll
window so that an element is
in view.
// GetWindowWidth(): get width
of a window.
// GetWindowHeight(): get
height of a window
// GetAvailScreenWidth(): get
available screen width
// GetAvailScreenHeight(): get
available screen height
// GetNiceWindowHeight(): get
a nice height for a new
browser window.
//
Open{External/Internal}Window(
): open a separate window
// CloseWindow(): close a
window
//
// - DOM walking utilities
// AnnotateTerms(): find terms
in a node and decorate them
with some tag
// AnnotateText(): find terms
in a text node and decorate
them with some tag
//
// - String utilties
// HtmlEscape(): html escapes
```

```
a string
// HtmlUnescape(): remove
html-escaping.
// QuoteEscape(): escape "
quotes.
// CollapseWhitespace():
collapse multiple whitespace
into one whitespace.
// Trim(): trim whitespace on
ends of string
// IsEmpty(): check if
CollapseWhiteSpace(String) ==
""
// IsLetterOrDigit(): check if
a character is a letter or a
digit
// ConvertEOLToLF(): normalize
the new-lines of a string.
// HtmlEscapeInsertWbrs():
HtmlEscapes and inserts <wbr>s
(word break tags)
//   after every n non-space
chars and/or after or before
certain special chars
//
// - TextArea utilities
// GetCursorPos(): finds the
cursor position of a textfield
// SetCursorPos(): sets the
cursor position in a textfield
//
// - Array utilities
// FindInArray(): do a linear
search to find an element
value.
// DeleteArrayElement():
return a new array with a
specific value removed.
// CloneObject(): clone an
object, copying its values
recursively.
// CloneEvent(): clone an
event; cannot use CloneObject
because it
//              suffers from
infinite recursion
//
```

```
// - Formatting utilities
// PrintArray(): used to
print/generate HTML by
combining static text
// and dynamic strings.
// ImageHtml(): create html
for an img tag
// FormatJSLink(): formats a
link that invokes js code when
clicked.
// MakeId3(): formats an id
that has two id numbers, eg,
foo_3_7
//
// - Timeouts
// SafeTimeout(): sets a
timeout with protection
against ugly JS-errors
// CancelTimeout(): cancels a
timeout with a given ID
// CancelAllTimeouts():
cancels all timeouts on a
given window
//
// - Miscellaneous
// IsDefined(): returns true
if argument is not undefined
// ------------------------------
------------------------------
---------------

// browser detection
function
BR_AgentContains_(str) {
  if (str in
BR_AgentContains_cache_) {
    return
BR_AgentContains_cache_[str];
  }

  return
BR_AgentContains_cache_[str] =

(navigator.userAgent.toLowerCa
se().indexOf(str) != -1);
}
// We cache the results of the
```

```
indexOf operation. This gets
us a 10x benefit in
// Gecko, 8x in Safari and 4x
in MSIE for all of the browser
checks
var BR_AgentContains_cache_ =
{};

function BR_IsIE() {
  return
(BR_AgentContains_('msie') ||
BR_AgentContains_('trident'))
&&
         !window.opera;
}

function BR_IsKonqueror() {
  return
BR_AgentContains_('konqueror')
;
}

function BR_IsSafari() {
  return
BR_AgentContains_('safari') ||
BR_IsKonqueror();
}

function BR_IsNav() {
  return !BR_IsIE() &&
         !BR_IsSafari() &&

BR_AgentContains_('mozilla');
}

var BACKSPACE_KEYNAME =
'Backspace';
var COMMA_KEYNAME = ',';
var DELETE_KEYNAME = 'Delete';
var UP_KEYNAME = 'ArrowUp';
var DOWN_KEYNAME =
'ArrowDown';
var LEFT_KEYNAME =
'ArrowLeft';
var RIGHT_KEYNAME =
'ArrowRight';
var ENTER_KEYNAME = 'Enter';
var ESC_KEYNAME = 'Escape';
var SPACE_KEYNAME = ' ';
var TAB_KEYNAME = 'Tab';
var SHIFT_KEYNAME = 'Shift';
var PAGE_DOWN_KEYNAME =
'PageDown';
var PAGE_UP_KEYNAME =
'PageUp';

var MAX_EMAIL_ADDRESS_LENGTH =
320; // 64 + '@' + 255
var MAX_SIGNATURE_LENGTH =
1000; // 1000 chars of maximum
signature

// ----------------------------
------------------------------
---------------
// Assertions
// DEPRECATED: Use assert.js
// ----------------------------
------------------------------
---------------
/**
 * DEPRECATED: Use assert.js
 */
function raise(msg) {
  if (typeof Error !=
'undefined') {
    throw new Error(msg ||
'Assertion Failed');
  } else {
    throw (msg);
  }
}

/**
 * DEPRECATED: Use assert.js
 *
 * Fail() is useful for
marking logic paths that
should
 * not be reached. For
example, if you have a class
that uses
 * ints for enums:
 *
 * MyClass.ENUM_FOO = 1;
 * MyClass.ENUM_BAR = 2;
 * MyClass.ENUM_BAZ = 3;
 *
 * And a switch statement
elsewhere in your code that
 * has cases for each of these
enums, then you can
 * "protect" your code as
follows:
 *
 * switch(type) {
 *   case MyClass.ENUM_FOO:
doFooThing(); break;
 *   case MyClass.ENUM_BAR:
doBarThing(); break;
 *   case MyClass.ENUM_BAZ:
doBazThing(); break;
 *   default:
 *     Fail("No enum in
MyClass with value: " + type);
 * }
 *
 * This way, if someone
introduces a new value for
this enum
 * without noticing this
switch statement, then the
code will
 * fail if the logic allows it
to reach the switch with the
 * new value, alerting the
developer that they should add
a
 * case to the switch to
handle the new value they have
introduced.
 *
 * @param {string} opt_msg to
display for failure
 *                  DEFAULT:
"Assertion failed"
 */
function Fail(opt_msg) {
  opt_msg = opt_msg ||
'Assertion failed';
  if (IsDefined(DumpError))
```

```
DumpError(opt_msg + '\n');
  raise(opt_msg);
}

/**
 * DEPRECATED: Use assert.js
 *
 * Asserts that an expression
is true (non-zero and non-
null).
 *
 * Note that it is critical
not to pass logic
 * with side-effects as the
expression for AssertTrue
 * because if the assertions
are removed by the
 * JSCompiler, then the
expression will be removed
 * as well, in which case the
side-effects will
 * be lost. So instead of
this:
 *
 *  AssertTrue(
criticalComputation() );
 *
 * Do this:
 *
 *  var result =
criticalComputation();
 *  AssertTrue(result);
 *
 * @param expression to
evaluate
 * @param {string} opt_msg to
display if the assertion fails
 *
 */
function
AssertTrue(expression,
opt_msg) {
  if (!expression) {
    opt_msg = opt_msg ||
'Assertion failed';
    Fail(opt_msg);
  }
```

```
}

/**
 * DEPRECATED: Use assert.js
 *
 * Asserts that a value is of
the provided type.
 *
 *   AssertType(6, Number);
 *   AssertType("ijk",
String);
 *   AssertType([], Array);
 *   AssertType({}, Object);
 *
AssertType(ICAL_Date.now(),
ICAL_Date);
 *
 * @param value
 * @param type A constructor
function
 * @param {string} opt_msg to
display if the assertion fails
 */
function AssertType(value,
type, opt_msg) {
  // for backwards
compatability only
  if (typeof value == type)
return;

  if (value || value == '') {
    try {
      if (type ==
AssertTypeMap[typeof value] ||
value instanceof type) return;
    } catch (e) {/* failure,
type was an illegal argument
to instanceof */}
  }
  let makeMsg = opt_msg ===
undefined;
  if (makeMsg) {
    if (typeof type ==
'function') {
      let match =
type.toString().match(/^\s*fun
ction\s+([^\s\{]+)/);
```

```
      if (match) type =
match[1];
    }
    opt_msg = 'AssertType
failed: <' + value + '> not
typeof '+ type;
  }
  Fail(opt_msg);
}

var AssertTypeMap = {
  'string': String,
  'number': Number,
  'boolean': Boolean,
};

var EXPIRED_COOKIE_VALUE =
'EXPIRED';


// -----------------------------
------------------------------
---------------
// Window/screen utilities
// TODO: these should be
renamed (e.g. GetWindowWidth
to GetWindowInnerWidth
// and moved to geom.js)
// -----------------------------
------------------------------
---------------
// Get page offset of an
element
function GetPageOffsetLeft(el)
{
  let x = el.offsetLeft;
  if (el.offsetParent != null)
{
    x +=
GetPageOffsetLeft(el.offsetPar
ent);
  }
  return x;
}

// Get page offset of an
element
```

```
function GetPageOffsetTop(el)
{
  let y = el.offsetTop;
  if (el.offsetParent != null)
{
    y +=
GetPageOffsetTop(el.offsetPare
nt);
  }
  return y;
}

// Get page offset of an
element
function GetPageOffset(el) {
  let x = el.offsetLeft;
  let y = el.offsetTop;
  if (el.offsetParent != null)
{
    let pos =
GetPageOffset(el.offsetParent)
;
    x += pos.x;
    y += pos.y;
  }
  return {x: x, y: y};
}

// Get the y position scroll
offset.
function GetScrollTop(win) {
  return
GetWindowPropertyByBrowser_(wi
n, getScrollTopGetters_);
}

var getScrollTopGetters_ = {
  ieQuirks_: function(win) {
    return
win.document.body.scrollTop;
  },
  ieStandards_: function(win)
{
    return
win.document.documentElement.s
crollTop;
  },
  dom_: function(win) {
    return win.pageYOffset;
  },
};

// Get the x position scroll
offset.
function GetScrollLeft(win) {
  return
GetWindowPropertyByBrowser_(wi
n, getScrollLeftGetters_);
}

var getScrollLeftGetters_ = {
  ieQuirks_: function(win) {
    return
win.document.body.scrollLeft;
  },
  ieStandards_: function(win)
{
    return
win.document.documentElement.s
crollLeft;
  },
  dom_: function(win) {
    return win.pageXOffset;
  },
};

// Scroll so that as far as
possible the entire element is
in view.
var ALIGN_BOTTOM = 'b';
var ALIGN_MIDDLE = 'm';
var ALIGN_TOP = 't';

var getWindowWidthGetters_ = {
  ieQuirks_: function(win) {
    return
win.document.body.clientWidth;
  },
  ieStandards_: function(win)
{
    return
win.document.documentElement.c
lientWidth;
  },
  dom_: function(win) {
    return win.innerWidth;
  },
};

function GetWindowHeight(win)
{
  return
GetWindowPropertyByBrowser_(wi
n, getWindowHeightGetters_);
}

var getWindowHeightGetters_ =
{
  ieQuirks_: function(win) {
    return
win.document.body.clientHeight
;
  },
  ieStandards_: function(win)
{
    return
win.document.documentElement.c
lientHeight;
  },
  dom_: function(win) {
    return win.innerHeight;
  },
};

/**
 * Allows the easy use of
different getters for IE
quirks mode, IE standards
 * mode and fully DOM-
compliant browers.
 *
 * @param win window to get
the property for
 * @param getters object with
various getters. Invoked with
the passed window.
 * There are three properties:
 * - ieStandards_: IE 6.0
standards mode
 * - ieQuirks_: IE 6.0 quirks
mode and IE 5.5 and older
```

```
 * - dom_: Mozilla, Safari and
other fully DOM compliant
browsers
 *
 * @private
 */
function
GetWindowPropertyByBrowser_(wi
n, getters) {
  try {
    if (BR_IsSafari()) {
      return
getters.dom_(win);
    } else if (!window.opera
&&
               'compatMode' in
win.document &&

win.document.compatMode ==
'CSS1Compat') {
      return
getters.ieStandards_(win);
    } else if (BR_IsIE()) {
      return
getters.ieQuirks_(win);
    }
  } catch (e) {
    // Ignore for now and fall
back to DOM method
  }

  return getters.dom_(win);
}

function
GetAvailScreenWidth(win) {
  return
win.screen.availWidth;
}

// Used for horizontally
centering a new window of the
given width in the
// available screen. Set the
new window's distance from the
left of the screen
// equal to this function's
```

```
return value.
// Params: width: the width of
the new window
// Returns: the distance from
the left edge of the screen
for the new window to
//   be horizontally centered
function GetCenteringLeft(win,
width) {
  return
(win.screen.availWidth -
width) >> 1;
}

// Used for vertically
centering a new window of the
given height in the
// available screen. Set the
new window's distance from the
top of the screen
// equal to this function's
return value.
// Params: height: the height
of the new window
// Returns: the distance from
the top edge of the screen for
the new window to
//   be vertically aligned.
function GetCenteringTop(win,
height) {
  return
(win.screen.availHeight -
height) >> 1;
}

/**
 * Opens a child popup window
that has no browser
toolbar/decorations.
 * (Copied from caribou's
common.js library with small
modifications.)
 *
 * @param url the URL for the
new window (Note: this will be
unique-ified)
 * @param opt_name the name of
```

```
the new window
 * @param opt_width the width
of the new window
 * @param opt_height the
height of the new window
 * @param opt_center if true,
the new window is centered in
the available screen
 * @param opt_hide_scrollbars
if true, the window hides the
scrollbars
 * @param opt_noresize if
true, makes window unresizable
 * @param opt_blocked_msg
message warning that the popup
has been blocked
 * @return {Window} a
reference to the new child
window
 */
function Popup(url, opt_name,
opt_width, opt_height,
opt_center,
  opt_hide_scrollbars,
opt_noresize, opt_blocked_msg)
{
  if (!opt_height) {
    opt_height =
Math.floor(GetWindowHeight(win
dow.top) * 0.8);
  }
  if (!opt_width) {
    opt_width =
Math.min(GetAvailScreenWidth(w
indow), opt_height);
  }

  let features = 'resizable='
+ (opt_noresize ? 'no' :
'yes') + ',' +
                 'scrollbars='
+ (opt_hide_scrollbars ? 'no'
: 'yes') + ',' +
                 'width=' +
opt_width + ',height=' +
opt_height;
  if (opt_center) {
    features += ',left=' +
GetCenteringLeft(window,
opt_width) + ',' +
                'top=' +
GetCenteringTop(window,
opt_height);
  }
  return OpenWindow(window,
url, opt_name, features,
opt_blocked_msg);
}

/**
 * Opens a new window. Returns
the new window handle. Tries
to open the new
 * window using top.open()
first. If that doesn't work,
then tries win.open().
 * If that still doesn't work,
prints an alert.
 * (Copied from caribou's
common.js library with small
modifications.)
 *
 * @param win the parent
window from which to open the
new child window
 * @param url the URL for the
new window (Note: this will be
unique-ified)
 * @param opt_name the name of
the new window
 * @param opt_features the
properties of the new window
 * @param opt_blocked_msg
message warning that the popup
has been blocked
 * @return {Window} a
reference to the new child
window
 */
function OpenWindow(win, url,
opt_name, opt_features,
opt_blocked_msg) {
  let newwin =
OpenWindowHelper(top, url,
opt_name, opt_features);
  if (!newwin || newwin.closed
|| !newwin.focus) {
    newwin =
OpenWindowHelper(win, url,
opt_name, opt_features);
  }
  if (!newwin || newwin.closed
|| !newwin.focus) {
    if (opt_blocked_msg)
alert(opt_blocked_msg);
  } else {
    // Make sure that the
window has the focus
    newwin.focus();
  }
  return newwin;
}

/*
 * Helper for OpenWindow().
 * (Copied from caribou's
common.js library with small
modifications.)
 */
function OpenWindowHelper(win,
url, name, features) {
  let newwin;
  if (features) {
    newwin = win.open(url,
name, features);
  } else if (name) {
    newwin = win.open(url,
name);
  } else {
    newwin = win.open(url);
  }
  return newwin;
}

// ----------------------------
------------------------------
---------------
// String utilities
// ----------------------------
------------------------------
---------------
```

```
// Do html escaping
var amp_re_ = /&/g;
var lt_re_ = /</g;
var gt_re_ = />/g;

// converts multiple ws chars
to a single space, and strips
// leading and trailing ws
var spc_re_ = /\s+/g;
var beg_spc_re_ = /^ /;
var end_spc_re_ = / $/;

var newline_re_ = /\r?\n/g;
var spctab_re_ = /[ \t]+/g;
var nbsp_re_ = /\xa0/g;

// URL-decodes the string. We
need to specially handle '+'s
because
// the javascript library
doesn't properly convert them
to spaces
var plus_re_ = /\+/g;

// Converts any instances of
"\r" or "\r\n" style EOLs into
"\n" (Line Feed),
// and also trim the extra
newlines and whitespaces at
the end.
var eol_re_ = /\r\n?/g;
var trailingspc_re_ = /[\n\t
]+$/;

// Converts a string to its
canonicalized label form.
var illegal_chars_re_ = /[
\/(){}&|\\\"\000]/g;

// ---------------------------
------------------------------
---------------
// TextArea utilities
// ---------------------------
------------------------------
---------------
```

```
// Gets the cursor pos in a
text area. Returns -1 if the
cursor pos cannot
// be determined or if the
cursor out of the textfield.
function GetCursorPos(win,
textfield) {
  try {
    if
(IsDefined(textfield.selection
End)) {
      // Mozilla directly
supports this
      return
textfield.selectionEnd;
    } else if
(win.document.selection &&
win.document.selection.createR
ange) {
      // IE doesn't export an
accessor for the endpoints of
a selection.
      // Instead, it uses the
TextRange object, which has an
extremely obtuse
      // API. Here's what
seems to work:

      // (1) Obtain a
textfield from the current
selection (cursor)
      let tr =
win.document.selection.createR
ange();

      // Check if the current
selection is in the textfield
      if (tr.parentElement()
!= textfield) {
        return -1;
      }

      // (2) Make a text range
encompassing the textfield
      let tr2 =
tr.duplicate();
tr2.moveToElementText(textfiel
d);

      // (3) Move the end of
the copy to the beginning of
the selection

tr2.setEndPoint('EndToStart',
tr);

      // (4) The span of the
textrange copy is equivalent
to the cursor pos
      let cursor =
tr2.text.length;

      // Finally, perform a
sanity check to make sure the
cursor is in the
      // textfield. IE
sometimes screws this up when
the window is activated
      if (cursor >
textfield.value.length) {
        return -1;
      }
      return cursor;
    } else {
      Debug('Unable to get
cursor position for: ' +
navigator.userAgent);

      // Just return the size
of the textfield
      // TODO: Investigate how
to get cursor pos in Safari!
      return
textfield.value.length;
    }
  } catch (e) {
    DumpException(e, 'Cannot
get cursor pos');
  }

  return -1;
}
```

```
function SetCursorPos(win,
textfield, pos) {
  if
(IsDefined(textfield.selection
End) &&

IsDefined(textfield.selectionS
tart)) {
    // Mozilla directly
supports this
    textfield.selectionStart =
pos;
    textfield.selectionEnd =
pos;
  } else if
(win.document.selection &&
textfield.createTextRange) {
    // IE has textranges. A
textfield's textrange
encompasses the
    // entire textfield's text
by default
    let sel =
textfield.createTextRange();

    sel.collapse(true);
    sel.move('character',
pos);
    sel.select();
  }
}

// ---------------------------
------------------------------
---------------
// Array utilities
// ---------------------------
------------------------------
---------------
// Find an item in an array,
returns the key, or -1 if not
found
function FindInArray(array, x)
{
  for (let i = 0; i <
array.length; i++) {
    if (array[i] == x) {
      return i;
    }
  }
  return -1;
}

// Delete an element from an
array
function
DeleteArrayElement(array, x) {
  let i = 0;
  while (i < array.length &&
array[i] != x) {
    i++;
  }
  array.splice(i, 1);
}

// Clean up email address:
// - remove extra spaces
// - Surround name with quotes
if it contains special
characters
// to check if we need "
quotes
// Note: do not use /g in the
regular expression, otherwise
the
// regular expression cannot
be reusable.
var specialchars_re_ = /[()
<>@,;:\\\".\[\]]/;

// ------------------------------
------------------------------
---------------
// Timeouts
//
// It is easy to forget to put
a try/catch block around a
timeout function,
// and the result is an ugly
user visible javascript error.
// Also, it would be nice if a
timeout associated with a
window is
// automatically cancelled
```

```
when the user navigates away
from that window.
//
// When storing timeouts in a
window, we can't let that
variable be renamed
// since the window could be
top.js, and renaming such a
property could
// clash with any of the
variables/functions defined in
top.js.
// ------------------------------
------------------------------
---------------
/**
 * Sets a timeout safely.
 * @param win the window
object. If null is passed in,
then a timeout if set
 *   on the js frame. If the
window is closed, or freed,
the timeout is
 *   automaticaaly cancelled
 * @param fn the callback
function: fn(win) will be
called.
 * @param ms number of ms the
callback should be called
later
 */
function SafeTimeout(win, fn,
ms) {
  if (!win) win = window;
  if (!win._tm) {
    win._tm = [];
  }
  let timeoutfn =
SafeTimeoutFunction_(win, fn);
  let id =
win.setTimeout(timeoutfn, ms);

  // Save the id so that it
can be removed from the _tm
array
  timeoutfn.id = id;
  // Safe the timeout in the
_tm array
  win._tm[id] = 1;

  return id;
}

/** Creates a callback
function for a timeout*/
function
SafeTimeoutFunction_(win, fn)
{
  var timeoutfn = function() {
    try {
      fn(win);

      let t = win._tm;
      if (t) {
        delete
t[timeoutfn.id];
      }
    } catch (e) {
      DumpException(e);
    }
  };
  return timeoutfn;
}

// ---------------------------
------------------------------
---------------
// Misc
// ---------------------------
------------------------------
---------------
// Check if a value is defined
function IsDefined(value) {
  return (typeof value) !=
'undefined';
}

/* Copyright 2016 The Chromium
Authors. All Rights Reserved.
 *
 * Use of this source code is
governed by a BSD-style
 * license that can be found
in the LICENSE file or at
```

```
 *
https://developers.google.com/
open-source/licenses/bsd
 */

var listen;
var unlisten;
var unlistenByKey;

(function() {
  let listeners = {};
  let nextId = 0;

  function getHashCode_(obj) {
    if (obj.listen_hc_ ==
null) {
      obj.listen_hc_ =
++nextId;
    }
    return obj.listen_hc_;
  }

  /**
   * Takes a node, event,
listener, and capture flag to
create a key
   * to identify the tuple in
the listeners hash.
   *
   * @param {Element} node The
node to listen to events on.
   * @param {string} event The
name of the event without the
"on" prefix.
   * @param {Function}
listener A function to call
when the event occurs.
   * @param {boolean}
opt_useCapture In DOM-
compliant browsers, this
determines
   *
whether the listener is fired
during the
   *
capture or bubble phase of the
event.
```

```
   * @return {string} key to
identify this tuple in the
listeners hash.
   */
  function createKey_(node,
event, listener,
opt_useCapture) {
    let nodeHc =
getHashCode_(node);
    let listenerHc =
getHashCode_(listener);
    opt_useCapture =
!!opt_useCapture;
    let key = nodeHc + '_' +
event + '_' + listenerHc + '_'
+ opt_useCapture;
    return key;
  }

  /**
   * Adds an event listener to
a DOM node for a specific
event.
   *
   * Listen() and unlisten()
use an indirect lookup of
listener functions
   * to avoid circular
references between DOM (in IE)
or XPCOM (in Mozilla)
   * objects which leak
memory. This makes it easier
to write OO
   * Javascript/DOM code.
   *
   * Examples:
   * listen(myButton, 'click',
myHandler, true);
   * listen(myButton, 'click',
this.myHandler.bind(this),
true);
   *
   * @param {Element} node The
node to listen to events on.
   * @param {string} event The
name of the event without the
"on" prefix.
```

```
   * @param {Function}
listener A function to call
when the event occurs.
   * @param {boolean}
opt_useCapture In DOM-
compliant browsers, this
determines
   *
whether the listener is fired
during the
   *
capture or bubble phase of the
event.
   * @return {string} a unique
key to indentify this
listener.
   */
  listen = function(node,
event, listener,
opt_useCapture) {
    let key = createKey_(node,
event, listener,
opt_useCapture);

    // addEventListener does
not allow multiple listeners
    if (key in listeners) {
      return key;
    }

    let proxy =
handleEvent.bind(null, key);
    listeners[key] = {
      listener: listener,
      proxy: proxy,
      event: event,
      node: node,
      useCapture:
opt_useCapture,
    };

    if (node.addEventListener)
{

node.addEventListener(event,
proxy, opt_useCapture);
    } else if
```

```
(node.attachEvent) {
      node.attachEvent('on' +
event, proxy);
    } else {
      throw new Error('Node {'
+ node + '} does not support
event listeners.');
    }

    return key;
  };

  /**
   * Removes an event listener
which was added with listen().
   *
   * @param {Element} node The
node to stop listening to
events on.
   * @param {string} event The
name of the event without the
"on" prefix.
   * @param {Function}
listener The listener function
to remove.
   * @param {boolean}
opt_useCapture In DOM-
compliant browsers, this
determines
   *
whether the listener is fired
during the
   *
capture or bubble phase of the
event.
   * @return {boolean}
indicating whether the
listener was there to remove.
   */
  unlisten = function(node,
event, listener,
opt_useCapture) {
    let key = createKey_(node,
event, listener,
opt_useCapture);

    return unlistenByKey(key);
  };

  /**
   * Variant of {@link
unlisten} that takes a key
that was returned by
   * {@link listen} and
removes that listener.
   *
   * @param {string} key Key
of event to be unlistened.
   * @return {boolean}
indicating whether it was
there to be removed.
   */
  unlistenByKey =
function(key) {
    if (!(key in listeners)) {
      return false;
    }
    let listener =
listeners[key];
    let proxy =
listener.proxy;
    let event =
listener.event;
    let node = listener.node;
    let useCapture =
listener.useCapture;

    if
(node.removeEventListener) {

node.removeEventListener(event
, proxy, useCapture);
    } else if
(node.detachEvent) {
      node.detachEvent('on' +
event, proxy);
    }

    delete listeners[key];
    return true;
  };

  /**
   * The function which is
```

```
actually called when the DOM
event occurs. This
   * function is a proxy for
the real listener the user
specified.
   */
  function handleEvent(key) {
    // pass all arguments
which were sent to this
function except listenerID
    // on to the actual
listener.
    let args =
Array.prototype.splice.call(ar
guments, 1, arguments.length);
    return
listeners[key].listener.apply(
null, args);
  }
})();
```

```
/* Copyright 2016 The Chromium
Authors. All Rights Reserved.
 *
 * Use of this source code is
governed by a BSD-style
 * license that can be found
in the LICENSE file or at
 *
https://developers.google.com/
open-source/licenses/bsd
 */

/**
 * @fileoverview A bunch of
XML HTTP recipes used to do
RPC from JavaScript
 */


/**
 * The active x identifier
used for ie.
 * @type String
 * @private
 */
var XH_ieProgId_;


// Domain for XMLHttpRequest
readyState
var
XML_READY_STATE_UNINITIALIZED
= 0;
var XML_READY_STATE_LOADING =
1;
var XML_READY_STATE_LOADED =
2;
var
XML_READY_STATE_INTERACTIVE =
3;
var XML_READY_STATE_COMPLETED
= 4;


/**
 * Initialize the private
state used by other functions.
 * @private
 */
function XH_XmlHttpInit_() {
  // The following blog post
describes what PROG IDs to use
to create the
  // XMLHTTP object in
Internet Explorer:
  //
http://blogs.msdn.com/xmlteam/
archive/2006/10/23/using-the-
right-version-of-msxml-in-
internet-explorer.aspx
  // However we do not (yet)
fully trust that this will be
OK for old versions
  // of IE on Win9x so we
therefore keep the last 2.
  // Versions 4 and 5 have
been removed because 3.0 is
the preferred "fallback"
  // per the article above.
  // - Version 5 was built for
```

```
Office applications and is not
recommended for
  //   web applications.
  // - Version 4 has been
superseded by 6 and is only
intended for legacy apps.
  // - Version 3 has a wide
install base and is serviced
regularly with the OS.

  /**
   * Candidate Active X types.
   * @type Array.<String>
   * @private
   */
  let XH_ACTIVE_X_IDENTS =
['MSXML2.XMLHTTP.6.0',
'MSXML2.XMLHTTP.3.0',
    'MSXML2.XMLHTTP',
'Microsoft.XMLHTTP'];

  if (typeof XMLHttpRequest ==
'undefined' &&
      typeof ActiveXObject !=
'undefined') {
    for (let i = 0; i <
XH_ACTIVE_X_IDENTS.length;
i++) {
      let candidate =
XH_ACTIVE_X_IDENTS[i];

      try {
        new
ActiveXObject(candidate);
        XH_ieProgId_ =
candidate;
        break;
      } catch (e) {
        // do nothing; try
next choice
      }
    }

    // couldn't find any
matches
    if (!XH_ieProgId_) {
      throw Error('Could not
create ActiveXObject. ActiveX
might be disabled,' +
                  ' or MSXML
might not be installed.');
    }
  }
}


XH_XmlHttpInit_();


/**
 * Create and return an xml
http request object that can
be passed to
 * {@link #XH_XmlHttpGET} or
{@link #XH_XmlHttpPOST}.
 */
function XH_XmlHttpCreate() {
  if (XH_ieProgId_) {
    return new
ActiveXObject(XH_ieProgId_);
  } else {
    return new
XMLHttpRequest();
  }
}


/**
 * Send a get request.
 * @param {XMLHttpRequest}
xmlHttp as from {@link
XH_XmlHttpCreate}.
 * @param {string} url the
service to contact
 * @param {Function} handler
function called when the
response is received.
 */
function
XH_XmlHttpGET(xmlHttp, url,
handler) {
  xmlHttp.open('GET', url,
true);
  xmlHttp.onreadystatechange =
handler;
  XH_XmlHttpSend(xmlHttp,
null);
}

/**
 * Send a post request.
 * @param {XMLHttpRequest}
xmlHttp as from {@link
XH_XmlHttpCreate}.
 * @param {string} url the
service to contact
 * @param {string} data the
request content.
 * @param {Function} handler
function called when the
response is received.
 */
function
XH_XmlHttpPOST(xmlHttp, url,
data, handler) {
  xmlHttp.open('POST', url,
true);
  xmlHttp.onreadystatechange =
handler;

xmlHttp.setRequestHeader('Cont
ent-Type', 'application/x-www-
form-urlencoded');
  XH_XmlHttpSend(xmlHttp,
data);
}

/**
 * Calls 'send' on the
XMLHttpRequest object and
calls a function called 'log'
 * if any error occured.
 *
 * @deprecated This dependes
on a function called 'log'.
You are better off
 * handling your errors on
application level.
 *
 * @param {XMLHttpRequest}
xmlHttp as from {@link
```

```
XH_XmlHttpCreate}.
 * @param {string|null} data
the request content.
 */
function
XH_XmlHttpSend(xmlHttp, data)
{
  try {
    xmlHttp.send(data);
  } catch (e) {
    // You may want to
log/debug this error one that
you should be aware of is
    // e.number ==
-2146697208, which occurs when
the 'Languages...' setting in
    // IE is empty.
    // This is not entirely
true. The same error code is
used when the user is
    // off line.
    console.log('XMLHttpSend
failed ' + e.toString() +
'<br>' + e.stack);
    throw e;
  }
}/* Copyright 2016 The Chromium
Authors. All Rights Reserved.

 *
 * Use of this source code is
governed by a BSD-style
 * license that can be found
in the LICENSE file or at
 *
https://developers.google.com/
open-source/licenses/bsd
 */

// shape related classes

/** a point in 2 cartesian
dimensions.
  * @constructor
  * @param x x-coord.
  * @param y y-coord.
  * @param opt_coordinateFrame
```

```
a key that can be passed to a
translation function to
  *   convert from one
coordinate frame to another.
  *   Coordinate frames might
correspond to things like
windows, iframes, or
  *   any element with a
position style attribute.
  */
function Point(x, y,
opt_coordinateFrame) {
  /** a numeric x coordinate.
*/
  this.x = x;
  /** a numeric y coordinate.
*/
  this.y = y;
  /** a key that can be passed
to a translation function to
    * convert from one
coordinate frame to another.
    * Coordinate frames might
correspond to things like
windows, iframes, or
    * any element with a
position style attribute.
    */
  this.coordinateFrame =
opt_coordinateFrame || null;
}
Point.prototype.toString =
function() {
  return '[P ' + this.x + ','
+ this.y + ']';
};
Point.prototype.clone =
function() {
  return new Point(this.x,
this.y, this.coordinateFrame);
};

/** a distance between two
points in 2-space in cartesian
form.
  * A delta doesn't have a
coordinate frame associated
```

```
since all the coordinate
  * frames used in the HTML
dom are convertible without
rotation/scaling.
  * If a delta is not being
used in pixel-space then it
may be annotated with
  * a coordinate frame, and
the undefined coordinate frame
can be assumed
  * to represent pixel space.
  * @constructor
  * @param dx distance along x
axis
  * @param dy distance along y
axis
  */
function Delta(dx, dy) {
  /** a numeric distance along
the x dimension. */
  this.dx = dx;
  /** a numeric distance along
the y dimension. */
  this.dy = dy;
}
Delta.prototype.toString =
function() {
  return '[D ' + this.dx + ','
+ this.dy + ']';
};

/** a rectangle or bounding
region.
  * @constructor
  * @param x x-coord of the
left edge.
  * @param y y-coord of the
top edge.
  * @param w width.
  * @param h height.
  * @param opt_coordinateFrame
a key that can be passed to a
translation function to
  *   convert from one
coordinate frame to another.
  *   Coordinate frames might
correspond to things like
```

```
windows, iframes, or
  *   any element with a
position style attribute.
  */
function Rect(x, y, w, h,
opt_coordinateFrame) {
  /** the numeric x coordinate
of the left edge. */
  this.x = x;
  /** the numeric y coordinate
of the top edge. */
  this.y = y;
  /** the numeric distance
between the right edge and the
left. */
  this.w = w;
  /** the numeric distance
between the top edge and the
bottom. */
  this.h = h;
  /** a key that can be passed
to a translation function to
    * convert from one
coordinate frame to another.
    * Coordinate frames might
correspond to things like
windows, iframes, or
    * any element with a
position style attribute.
    */
  this.coordinateFrame =
opt_coordinateFrame || null;
}

/**
 * Determines whether the
Rectangle contains the Point.
 * The Point is considered
"contained" if it lies
 * on the boundary of, or in
the interior of, the
Rectangle.
 *
 * @param {Point} p
 * @return boolean indicating
if this Rect contains p
 */
```

```
Rect.prototype.contains =
function(p) {
  return this.x <= p.x && p.x
< (this.x + this.w) &&
           this.y <= p.y &&
p.y < (this.y + this.h);
};

/**
 * Determines whether the
given rectangle intersects
this rectangle.
 *
 * @param {Rect} r
 * @return boolean indicating
if this the two rectangles
intersect
 */
Rect.prototype.intersects =
function(r) {
  let p = function(x, y) {
    return new Point(x, y,
null);
  };

  return this.contains(p(r.x,
r.y)) ||
         this.contains(p(r.x +
r.w, r.y)) ||
         this.contains(p(r.x +
r.w, r.y + r.h)) ||
         this.contains(p(r.x,
r.y + r.h)) ||
         r.contains(p(this.x,
this.y)) ||
         r.contains(p(this.x +
this.w, this.y)) ||
         r.contains(p(this.x +
this.w, this.y + this.h)) ||
         r.contains(p(this.x,
this.y + this.h));
};

Rect.prototype.toString =
function() {
  return '[R ' + this.w + 'x'
+ this.h + '+' + this.x + '+'
+ this.y + ']';
};

Rect.prototype.clone =
function() {
  return new Rect(this.x,
this.y, this.w, this.h,
this.coordinateFrame);
};/* Copyright 2016 The Chromium
Authors. All Rights Reserved.

 *
 * Use of this source code is
governed by a BSD-style
 * license that can be found
in the LICENSE file or at
 *
https://developers.google.com/
open-source/licenses/bsd
 */

// functions for dealing with
layout and geometry of page
elements.
// Requires shapes.js

/** returns the bounding box
of the given DOM node in
document space.
  *
  * @param {Element?} obj a
DOM node.
  * @return {Rect?}
  */
function nodeBounds(obj) {
  if (!obj) return null;

  function
fixRectForScrolling(r) {
    // Need to take into
account scrolling offset of
ancestors (IE already does
    // this)
    for (let o =
obj.offsetParent;
      o && o.offsetParent;
      o = o.offsetParent) {
```

```
      if (o.scrollLeft) {
        r.x -= o.scrollLeft;
      }
      if (o.scrollTop) {
        r.y -= o.scrollTop;
      }
    }
  }

  let refWindow;
  if (obj.ownerDocument &&
obj.ownerDocument.parentWindow
) {
    refWindow =
obj.ownerDocument.parentWindow
;
  } else if (obj.ownerDocument
&&
obj.ownerDocument.defaultView)
{
    refWindow =
obj.ownerDocument.defaultView;
  } else {
    refWindow = window;
  }

  // IE, Mozilla 3+
  if
(obj.getBoundingClientRect) {
    let rect =
obj.getBoundingClientRect();

    return new Rect(rect.left
+ GetScrollLeft(refWindow),
      rect.top +
GetScrollTop(refWindow),
      rect.right - rect.left,
      rect.bottom - rect.top,
      refWindow);
  }

  // Mozilla < 3
  if (obj.ownerDocument &&
obj.ownerDocument.getBoxObject
For) {
    let box =
obj.ownerDocument.getBoxObject
```

```
For(obj);
    var r = new Rect(box.x,
box.y, box.width, box.height,
refWindow);
    fixRectForScrolling(r);
    return r;
  }

  // Fallback to recursively
computing this
  let left = 0;
  let top = 0;
  for (let o = obj;
o.offsetParent; o =
o.offsetParent) {
    left += o.offsetLeft;
    top += o.offsetTop;
  }

  var r = new Rect(left, top,
obj.offsetWidth,
obj.offsetHeight, refWindow);
  fixRectForScrolling(r);
  return r;
}

function GetMousePosition(e) {
  // copied from
http://www.quirksmode.org/js/e
vents_compinfo.html
  let posx = 0;
  let posy = 0;
  if (e.pageX || e.pageY) {
    posx = e.pageX;
    posy = e.pageY;
  } else if (e.clientX ||
e.clientY) {
    let obj = (e.target ?
e.target : e.srcElement);
    let refWindow;
    if (obj.ownerDocument &&
obj.ownerDocument.parentWindow
) {
      refWindow =
obj.ownerDocument.parentWindow
;
    } else {
      refWindow = window;
    }
    posx = e.clientX +
GetScrollLeft(refWindow);
    posy = e.clientY +
GetScrollTop(refWindow);
  }
  return new Point(posx, posy,
window);
}

/* Copyright 2016 The Chromium
Authors. All Rights Reserved.
 *
 * Use of this source code is
governed by a BSD-style
 * license that can be found
in the LICENSE file or at
 *
https://developers.google.com/
open-source/licenses/bsd
 */

/**
 * It is common to make a DIV
temporarily visible to
simulate
 * a popup window. Often, this
is done by adding an onClick
 * handler to the element that
can be clicked on to show the
 * popup.
 *
 * Unfortunately, closing the
popup is not as simple.
 * The popup creator often
wants to let the user close
 * the popup by clicking
elsewhere on the window;
however,
 * the popup only receives
mouse events that occur
 * on the popup itself. Thus,
popups need a mechanism
 * that notifies them that the
user has clicked elsewhere
 * to try to get rid of them.
 *
 * PopupController is such a
mechanism --
 * it monitors all mousedown
events that
 * occur in the window so that
it can notify registered
 * popups of the mousedown,
and the popups can choose
 * to deactivate themselves.
 *
 * For an object to qualify as
a popup, it must have a
 * function called
"deactivate" that takes a
mousedown event
 * and returns a boolean
indicating that it has
deactivated
 * itself as a result of that
event.
 *
 * EXAMPLE:
 *
 * // popup that attaches
itself to the supplied div
 * function MyPopup(div) {
 *   this._div = div;
 *   this._isVisible = false;
 *   this._innerHTML = ...
 * }
 *
 * MyPopup.prototype.show =
function() {
 *   this._div.display = '';
 *   this._isVisible = true;
 *   PC_addPopup(this);
 * }
 *
 * MyPopup.prototype.hide =
function() {
 *   this._div.display =
'none';
 *   this._isVisible = false;
 * }
 *
 *
MyPopup.prototype.deactivate =
function(e) {
 *   if (this._isVisible) {
 *     var p =
GetMousePosition(e);
 *     if
(nodeBounds(this._div).contain
s(p)) {
 *       return false; // use
clicked on popup, remain
visible
 *     } else {
 *       this.hide();
 *       return true; //
clicked outside popup, make
invisible
 *     }
 *   } else {
 *     return true; // already
deactivated, not visible
 *   }
 * }
 *
 * DEPENDENCIES (from this
directory):
 *   bind.js
 *   listen.js
 *   common.js
 *   shapes.js
 *   geom.js
 *
 * USAGE:
 *  _PC_Install() must be
called after the body is
loaded
 */

/**
 * PopupController
constructor.
 * @constructor
 */
function PopupController() {
  this.activePopups_ = [];
}

/**
 * @param {Document} opt_doc
```

```
document to add
PopupController to
 *                   DEFAULT:
"document" variable that is
currently in scope
 * @return {boolean}
indicating if PopupController
installed for the document;
 *                   returns
false if document already had
PopupController
 */
function _PC_Install(opt_doc)
{
  if
(gPopupControllerInstalled)
return false;
  gPopupControllerInstalled =
true;
  let doc = (opt_doc) ?
opt_doc : document;

  // insert _notifyPopups in
BODY's onmousedown chain
  listen(doc.body,
'mousedown', PC_notifyPopups);
  return true;
}

/**
 * Notifies each popup of a
mousedown event, giving
 * each popup the chance to
deactivate itself.
 *
 * @throws Error if a popup
does not have a deactivate
function
 *
 * @private
 */
function PC_notifyPopups(e) {
  if
(gPopupController.activePopups
_.length == 0) return false;
  e = e || window.event;
  for (let i =
gPopupController.activePopups_
.length - 1; i >= 0; --i) {
    let popup =
gPopupController.activePopups_
[i];
    PC_assertIsPopup(popup);
    if (popup.deactivate(e)) {

gPopupController.activePopups_
.splice(i, 1);
    }
  }
  return true;
}

/**
 * Adds the popup to the list
of popups to be
 * notified of a mousedown
event.
 *
 * @return boolean indicating
if added popup; false if
already contained
 * @throws Error if popup does
not have a deactivate function
 */
function PC_addPopup(popup) {
  PC_assertIsPopup(popup);
  for (let i = 0; i <
gPopupController.activePopups_
.length; ++i) {
    if (popup ===
gPopupController.activePopups_
[i]) return false;
  }

gPopupController.activePopups_
.push(popup);
  return true;
}

/** asserts that popup has a
deactivate function */
function
PC_assertIsPopup(popup) {
  AssertType(popup.deactivate,
Function, 'popup missing
deactivate function');
}

var gPopupController = new
PopupController();
var gPopupControllerInstalled
= false;
```

```
/* Copyright 2016 The Chromium
Authors. All Rights Reserved.
 *
 * Use of this source code is
governed by a BSD-style
 * license that can be found
in the LICENSE file or at
 *
https://developers.google.com/
open-source/licenses/bsd
 */

/**
 * An autocomplete library for
javascript.
 * Public API
 * - _ac_install() install
global handlers required for
everything else to
 *   function.
 * - _ac_register(SC) register
a store constructor (see
below)
 * - _ac_isCompleting() true
iff focus is in an auto
complete box and the user
 *   has triggered completion
with a keystroke, and
completion has not been
 *   cancelled
(programatically or
otherwise).
 * -
```

```
_ac_isCompleteListShowing()
true if _as_isCompleting and
the complete list
 *   is visible to the user.
 * - _ac_cancel() if
completing, stop it, otherwise
a no-op.
 *
 *
 * A quick example
 *     // an auto complete
store
 *     var
myFavoritestAutoCompleteStore
= new _AC_SimpleStore(
 *       ['some', 'strings',
'to', 'complete']);
 *
 *     // a store constructor
 *     _ac_register(function
(inputNode, keyEvent) {
 *         if (inputNode.id ==
'my-auto-completing-check-
box') {
 *           return
myFavoritestAutoCompleteStore;
 *         }
 *         return null;
 *       });
 *
 *     <html>
 *       <head>
 *         <script
type=text/javascript
src=ac.js></script>
 *       </head>
 *       <body
onload=_ac_install()>
 *         <!-- the
constructor above looks at the
id.  It could as easily
 *           - look at the
class, name, or value.
 *           - The
autocomplete=off stops browser
autocomplete from
 *           - interfering
```

```
with our autocomplete
 *           -->
 *         <input type=text
id="my-auto-completing-check-
box"
 *          autocomplete=off>
 *       </body>
 *     </html>
 *
 *
 * Concepts
 * - Store Constructor
function
 *   A store constructor is a
policy function with the
signature
 *     _AC_Store
myStoreConstructor(
 *
HtmlInputElement|HtmlTextAreaE
lement inputNode, Event
keyEvent)
 *   When a key event is
received on a text input or
text area, the autocomplete
 *   library will try each of
the store constructors in turn
until it finds one
 *   that returns an AC_Store
which will be used for auto-
completion of that
 *   text box until focus is
lost.
 *
 * - interface _AC_Store
 *   An autocomplete store
encapsulates all operations
that affect how a
 *   particular text node is
autocompleted.  It has the
following operations:
 *   - String
completable(String inputValue,
int caret)
 *     This method returns
null if not completable or the
section of inputValue
```

```
 *     that is subject to
completion.  If autocomplete
works on items in a
 *     comma separated list,
then the input value "foo, ba"
might yield "ba"
 *     as the completable
chunk since it is separated
from its predecessor by
 *     a comma.
 *     caret is the position
of the text cursor (caret) in
the text input.
 *   - _AC_Completion[]
completions(String
completable,
 *
_AC_Completion[] toFilter)
 *     This method returns
null if there are no
completions.  If toFilter is
 *     not null or undefined,
then this method may assume
that toFilter was
 *     returned as a set of
completions that contain
completable.
 *   - String
substitute(String inputValue,
int caret,
 *
String completable,
_AC_Completion completion)
 *     returns the inputValue
with the given completion
substituted for the
 *     given completable.
caret has the same meaning as
in the
 *     completable operation.
 *   - String
oncomplete(boolean completed,
String key,
 *
HTMLElement element, String
text)
 *     This method is called
```

```
when the user hits a
completion key. The default
 *     value is to do nothing,
but you can override it if you
want. Note that
 *     key will be null if the
user clicked on it to select
 *   - Boolean
autoselectFirstRow()
 *     This method returns
True by default, but
subclasses can override it
 *     to make autocomplete
fields that require the user
to press the down
 *     arrow or do a mouseover
once before any completion
option is considered
 *     to be selected.
 *
 * - class _AC_SimpleStore
 *   An implementation of
_AC_Store that completes a set
of strings given at
 *   construct time in a text
field with a comma separated
value.
 *
 * - struct _AC_Completion
 *   a struct with two fields
 *   - String value : the
plain text completion value
 *   - String html : the
value, as html, with the
completable in bold.
 *
 * Key Handling
 * Several keys affect
completion in an autocompleted
input.
 * ESC - the escape key
cancels autocompleting.  The
autocompletion will have
 *   no effect on the focused
textbox until it loses focus,
regains it, and
 *   a key is pressed.
 * ENTER - completes using the
currently selected completion,
or if there is
 *   only one, uses that
completion.
 * UP ARROW - selects the
completion above the current
selection.
 * DOWN ARROW - selects the
completion below the current
selection.
 *
 *
 * CSS styles
 * The following CSS selector
rules can be used to change
the completion list
 * look:
 * #ac-list
style of the auto-complete
list
 * #ac-list .selected
style of the selected item
 * #ac-list b
style of the matching text in
a candidate completion
 *
 * Dependencies
 * The library depends on the
following libraries:
 * javascript:base for
definition of key constants
and SetCursorPos
 * javascript:shapes for
nodeBounds()
 */

/**
 * install global handlers
required for the rest of the
module to function.
 */
function _ac_install() {

ac_addHandler_(document.body,
'onkeydown', ac_keyevent_);
ac_addHandler_(document.body,
'onkeypress', ac_keyevent_);
}

/**
 * register a store
constructor
 * @param storeConstructor a
function like
 *   _AC_Store
myStoreConstructor(HtmlInputEl
ement|HtmlTextArea, Event)
 */
function
_ac_register(storeConstructor)
{
  // check that not already
registered
  for (let i =
ac_storeConstructors.length; -
-i >= 0;) {
    if
(ac_storeConstructors[i] ===
storeConstructor) {
      return;
    }
  }

ac_storeConstructors.push(stor
eConstructor);
}

/**
 * may be attached as an
onfocus handler to a text
input to popup autocomplete
 * immediately on the box
gaining focus.
 */
function _ac_onfocus(event) {
  ac_keyevent_(event);
}

/**
 * true iff the autocomplete
widget is currently active.
 */
```

```
function _ac_isCompleting() {
  return !!ac_store &&
!ac_suppressCompletions;
}

/**
 * true iff the completion
list is displayed.
 */
function
_ac_isCompleteListShowing() {
  return !!ac_store &&
!ac_suppressCompletions &&
ac_completions &&
    ac_completions.length;
}

/**
 * cancel any autocomplete in
progress.
 */
function _ac_cancel() {
  ac_suppressCompletions =
true;

ac_updateCompletionList(false)
;
}

/** add a handler without
whacking any existing handler.
@private */
function ac_addHandler_(node,
handlerName, handler) {
  const oldHandler =
node[handlerName];
  if (!oldHandler) {
    node[handlerName] =
handler;
  } else {
    node[handlerName] =
ac_fnchain_(node[handlerName],
handler);
  }
  return oldHandler;
}
```

```
/** cancel the event. @private
*/
function
ac_cancelEvent_(event) {
  if ('stopPropagation' in
event) {
    event.stopPropagation();
  } else {
    event.cancelBubble = true;
  }

  // This is handled in IE by
returning false from the
handler
  if ('preventDefault' in
event) {
    event.preventDefault();
  }
}

/** Call two functions, a and
b, and return false if either
one returns
    false.  This is used as a
primitive way to attach
multiple event
    handlers to an element
without using
addEventListener().   This
    library predates the
availablity of
addEventListener().
    @private
*/
function ac_fnchain_(a, b) {
  return function() {
    const ar = a.apply(this,
arguments);
    const br = b.apply(this,
arguments);

    // NOTE 1: (undefined &&
false) -> undefined
    // NOTE 2: returning FALSE
from a onkeypressed cancels
it,
    //          returning
```

```
UNDEFINED does not.
    // As such, we
specifically look for falses
here
    if (ar === false || br ===
false) {
      return false;
    } else {
      return true;
    }
  };
}

/** key press handler.
@private */
function ac_keyevent_(event) {
  event = event ||
window.event;

  const source =
getTargetFromEvent(event);
  const isInput = 'INPUT' ==
source.tagName &&

source.type.match(/^text|email
$/i);
  const isTextarea =
'TEXTAREA' == source.tagName;
  if (!isInput && !isTextarea)
return true;

  const key = event.key;
  const isDown = event.type ==
'keydown';
  const isShiftKey =
event.shiftKey;
  let storeFound = true;

  if ((source !==
ac_focusedInput) || (ac_store
=== null)) {
    ac_focusedInput = source;
    storeFound = false;
    if (ENTER_KEYNAME !== key
&& ESC_KEYNAME !== key) {
      for (let i = 0; i <
ac_storeConstructors.length;
++i) {
        const store =
(ac_storeConstructors[i])
(source, event);
        if (store) {
          ac_store = store;

ac_store.setAvoid(event);
          ac_oldBlurHandler =
ac_addHandler_(
              ac_focusedInput,
'onblur', _ac_ob);
          storeFound = true;
          break;
        }
      }

      // There exists an odd
condition where an edit box
with autocomplete
      // attached can be
removed from the DOM without
blur being called
      // In which case we are
left with a store around that
will try to
      // autocomplete the next
edit box to receive focus. We
need to clean
      // this up

      // If we can't find a
store, force a blur
      if (!storeFound) {
        _ac_ob(null);
      }
    }
    // ac-table rows need to
be removed when switching to
another input.

ac_updateCompletionList(false)
;
  }
  // If the user typed Esc
when the auto-complete menu
was not shown,
  // then blur the input text
field so that the user can use
keyboard
  // shortcuts.
  const acList =
document.getElementById('ac-
list');
  if (ESC_KEYNAME == key &&
      (!acList ||
acList.style.display ==
'none')) {
    ac_focusedInput.blur();
  }

  if (!storeFound) return
true;

  const isCompletion =
ac_store.isCompletionKey(key,
isDown, isShiftKey);
  const hasResults =
ac_completions &&
(ac_completions.length > 0);
  let cancelEvent = false;

  if (isCompletion &&
hasResults) {
    // Cancel any enter
keystrokes if something is
selected so that the
    // browser doesn't go
submitting the form.
    cancelEvent =
(!ac_suppressCompletions &&
!!ac_completions &&

(ac_selected != -1));

window.setTimeout(function() {
      if (ac_store) {
        ac_handleKey_(key,
isDown, isShiftKey);
      }
    }, 0);
  } else if (!isCompletion) {
    // Don't want to also blur
the field. Up and down move
the cursor (in
    // Firefox) to the
start/end of the field. We
also don't want that while
    // the list is showing.
    cancelEvent = (key ==
ESC_KEYNAME ||
                  key ==
DOWN_KEYNAME ||
                  key ==
UP_KEYNAME);


window.setTimeout(function() {
      if (ac_store) {
        ac_handleKey_(key,
isDown, isShiftKey);
      }
    }, 0);
  } else { // implicit if
(isCompletion && !hasResults)
    if (ac_store.oncomplete) {

ac_store.oncomplete(false,
key, ac_focusedInput,
undefined);
    }
  }

  if (cancelEvent) {
    ac_cancelEvent_(event);
  }

  return !cancelEvent;
}

/** Autocomplete onblur
handler. */
function _ac_ob(event) {
  if (ac_focusedInput) {
    ac_focusedInput.onblur =
ac_oldBlurHandler;
  }
  ac_store = null;
  ac_focusedInput = null;
  ac_everTyped = false;
  ac_oldBlurHandler = null;
  ac_suppressCompletions =
false;

ac_updateCompletionList(false)
;
}

/** @constructor */
function _AC_Store() {
}
/** returns the chunk of the
input to treat as completable.
*/
_AC_Store.prototype.completabl
e = function(inputValue,
caret) {
  console.log('UNIMPLEMENTED
completable');
};
/** returns the chunk of the
input to treat as completable.
*/
_AC_Store.prototype.completion
s = function(prefix, tofilter)
{
  console.log('UNIMPLEMENTED
completions');
};
/** returns the chunk of the
input to treat as completable.
*/
_AC_Store.prototype.oncomplete
= function(completed, key,
element, text) {
  // Call the onkeyup handler
so that choosing an
autocomplete option has
  // the same side-effect as
typing.  E.g., exposing the
next row of input
  // fields.
  element.dispatchEvent(new
Event('keyup'));
  _ac_ob();
};
/** substitutes a completion
for a completable in a text
```

```
input's value. */
_AC_Store.prototype.substitute
=
  function(inputValue, caret,
completable, completion) {
    console.log('UNIMPLEMENTED
substitute');
  };
/** true iff hitting a comma
key should complete. */
_AC_Store.prototype.commaCompl
etes = true;
/**
 * true iff the given
keystroke should cause a
completion (and be consumed in
 * the process.
 */
_AC_Store.prototype.isCompleti
onKey = function(key, isDown,
isShiftKey) {
  if (!isDown &&
(ENTER_KEYNAME === key ||

(COMMA_KEYNAME == key &&
this.commaCompletes))) {
    return true;
  }
  if (TAB_KEYNAME === key &&
!isShiftKey) {
    // IE doesn't fire an
event for tab on click in a
text field, and firefox
    // requires that the
onkeypress event for tab be
consumed or it navigates
    // to next field.
    return false;
    // JER: return isDown ==
BR_IsIE();
  }
  return false;
};

_AC_Store.prototype.setAvoid =
function(event) {
  if (event &&
event.avoidValues) {
    ac_avoidValues =
event.avoidValues;
  } else {
    ac_avoidValues =
this.computeAvoid();
  }
  ac_avoidValues =
ac_avoidValues.map((val) =>
val.toLowerCase());
};

/* Subclasses may implement
this to compute values to
avoid
   offering in the current
input field, i.e., because
those
   values are already used. */
_AC_Store.prototype.computeAvo
id = function() {
  return [];
};


function
_AC_AddItemToFirstCharMap(firs
tCharMap, ch, s) {
  let l = firstCharMap[ch];
  if (!l) {
    l = firstCharMap[ch] = [];
  } else if (l[l.length -
1].value == s) {
    return;
  }
  l.push(new _AC_Completion(s,
null, ''));
}

/**
 * an _AC_Store implementation
suitable for completing lists
of email
 * addresses.
 * @constructor
 */
function
```

```
_AC_SimpleStore(strings,
opt_docStrings) {
  this.firstCharMap_ = {};

  for (let i = 0; i <
strings.length; ++i) {
    let s = strings[i];
    if (!s) {
      continue;
    }
    if (opt_docStrings &&
opt_docStrings[s]) {
      s = s + ' ' +
opt_docStrings[s];
    }

    const parts =
s.split(/\W+/);
    for (let j = 0; j <
parts.length; ++j) {
      if (parts[j]) {

_AC_AddItemToFirstCharMap(

this.firstCharMap_,
parts[j].charAt(0).toLowerCase
(), strings[i]);
      }
    }
  }

  // The maximimum number of
results that we are willing to
show
  this.countThreshold = 2500;
  this.docstrings =
opt_docStrings || {};
}
_AC_SimpleStore.prototype =
new _AC_Store();
_AC_SimpleStore.prototype.cons
tructor = _AC_SimpleStore;

_AC_SimpleStore.prototype.comp
letable =
  function(inputValue, caret)
{
  // complete after the last
comma not inside ""s
    let start = 0;
    let state = 0;
    for (let i = 0; i < caret;
++i) {
      const ch =
inputValue.charAt(i);
      switch (state) {
        case 0:
          if ('"' == ch) {
            state = 1;
          } else if (',' == ch
|| ' ' == ch) {
            start = i + 1;
          }
          break;
        case 1:
          if ('"' == ch) {
            state = 0;
          }
          break;
      }
    }
    while (start < caret &&
         '
\t\r\n'.indexOf(inputValue.cha
rAt(start)) >= 0) {
      ++start;
    }
    return
inputValue.substring(start,
caret);
  };


/** Simple function to create
a <span> with matching text in
bold.
 */
function
_AC_CreateSpanWithMatchHighlig
hted(match) {
  const span =
document.createElement('span')
;
```

```
span.appendChild(document.crea
teTextNode(match[1] || ''));
  const bold =
document.createElement('b');
  span.appendChild(bold);

bold.appendChild(document.crea
teTextNode(match[2]));

span.appendChild(document.crea
teTextNode(match[3] || ''));
  return span;
};


/**
 * Get all completions
matching the given prefix.
 * @param {string} prefix The
prefix of the text to
autocomplete on.
 * @param {List.<string>?}
toFilter Optional list to
filter on. Otherwise will
 *     use this.firstCharMap_
using the prefix's first
character.
 * @return {List.
<_AC_Completion>} The computed
list of completions.
 */
_AC_SimpleStore.prototype.comp
letions = function(prefix) {
  if (!prefix) {
    return [];
  }
  toFilter =
this.firstCharMap_[prefix.char
At(0).toLowerCase()];

  // Since we use prefix to
build a regular expression, we
need to escape RE
  // characters. We match '-',
'{', '$' and others in the
prefix and convert
  // them into "\-", "\{",
"\$".
  const
regexForRegexCharacters =
/([\^*+\-\$\\\{\}\(\)\[\]\#?
\.])/g;
  const modifiedPrefix =
prefix.replace(regexForRegexCh
aracters, '\\$1');

  // Match the modifiedPrefix
anywhere as long as it is
either at the very
  // beginning "Th" -> "The
Hobbit", or comes immediately
after a word separator
  // such as "Ga" -> "The-
Great-Gatsby".
  const patternRegex =
'^(.*\\W)?(' + modifiedPrefix
+ ')(.*)';
  const pattern = new
RegExp(patternRegex, 'i' /*
ignore case */);

  // We keep separate lists of
possible completions that were
generated
  // by matching a value or
generated by matching a
docstring.  We return
  // a concatenated list so
that value matches all come
before docstring
  // matches.
  const completions = [];
  const docCompletions = [];

  if (toFilter) {
    const toFilterLength =
toFilter.length;
    for (let i = 0; i <
toFilterLength; ++i) {
      const docStr =
this.docstrings[toFilter[i].va
lue];
      let compSpan = null;
      let docSpan = null;
      const matches =
toFilter[i].value.match(patter
n);
      const docMatches =
docStr &&
docStr.match(pattern);
      if (matches) {
        compSpan =
_AC_CreateSpanWithMatchHighlig
hted(matches);
        if (docStr) docSpan =
document.createTextNode(docStr
);
      } else if (docMatches) {
        compSpan =
document.createTextNode(toFilt
er[i].value);
        docSpan =
_AC_CreateSpanWithMatchHighlig
hted(docMatches);
      }

      if (compSpan) {
        const newCompletion =
new _AC_Completion(
            toFilter[i].value,
compSpan, docSpan);

        if (matches) {

completions.push(newCompletion
);
        } else {

docCompletions.push(newComplet
ion);
        }
        if (completions.length
+ docCompletions.length >
this.countThreshold) {
          break;
        }
      }
    }
  }

  return
```

```
completions.concat(docCompleti
ons);
};

// Normally, when the user
types a few characters, we
aggressively
// select the first possible
completion (if any).  When the
user
// hits ENTER, that first
completion is substituted.
When that
// behavior is not desired,
override this to return false.
_AC_SimpleStore.prototype.auto
selectFirstRow = function() {
  return true;
};

// Comparison function for
_AC_Completion
function
_AC_CompareACCompletion(a, b)
{
  // convert it to lower case
and remove all leading junk
  const aval =
a.value.toLowerCase().replace(
/^\W*/, '');
  const bval =
b.value.toLowerCase().replace(
/^\W*/, '');

  if (a.value === b.value) {
    return 0;
  } else if (aval < bval) {
    return -1;
  } else {
    return 1;
  }
}

_AC_SimpleStore.prototype.subs
titute =
function(inputValue, caret,
completable, completion) {
  return
inputValue.substring(0, caret
- completable.length) +
    completion.value + ', ' +
inputValue.substring(caret);
};

/**
 * a possible completion.
 * @constructor
 */
function _AC_Completion(value,
compSpan, docSpan) {
  /** plain text. */
  this.value = value;
  if (typeof compSpan ==
'string') compSpan =
document.createTextNode(compSp
an);
  this.compSpan = compSpan;
  if (typeof docSpan ==
'string') docSpan =
document.createTextNode(docSpa
n);
  this.docSpan = docSpan;
}
_AC_Completion.prototype.toStr
ing = function() {
  return '(AC_Completion: ' +
this.value + ')';
};

/** registered store
constructors.  @private */
var ac_storeConstructors = [];
/**
 * the focused text input or
textarea whether store is null
or not.
 * A text input may have focus
and this may be null iff no
key has been typed in
 * the text input.
 */
var ac_focusedInput = null;
/**
 * null or the autocomplete
```

```
store used to complete
ac_focusedInput.
 * @private
 */
var ac_store = null;
/** store handler from
ac_focusedInput. @private */
var ac_oldBlurHandler = null;
/**
 * true iff user has indicated
completions are unwanted (via
ESC key)
 * @private
 */
var ac_suppressCompletions =
false;
/**
 * chunk of completable text
seen last keystroke.
 * Used to generate
ac_completions.
 * @private
 */
let ac_lastCompletable = null;
/** an array of
_AC_Completions.  @private */
var ac_completions = null;
/** -1 or in [0,
_AC_Completions.length).
@private */
var ac_selected = -1;

/** Maximum number of options
displayed in menu. @private */
const ac_max_options = 100;

/** Don't offer these values
because they are already used.
@private */
let ac_avoidValues = [];

/**
 * handles all the key
strokes, updating the
completion list, tracking
selected
 * element, performing
```

```
substitutions, etc.
 * @private
 */
function ac_handleKey_(key,
isDown, isShiftKey) {
  // check completions
  ac_checkCompletions();
  let show = true;
  const numCompletions =
ac_completions ?
ac_completions.length : 0;
  // handle enter and tab on
key press and the rest on key
down
  if
(ac_store.isCompletionKey(key,
isDown, isShiftKey)) {
    if (ac_selected < 0 &&
numCompletions >= 1 &&

ac_store.autoselectFirstRow())
{
      ac_selected = 0;
    }
    if (ac_selected >= 0) {
      const backupInput =
ac_focusedInput;
      const completeValue =
ac_completions[ac_selected].va
lue;
      ac_complete();
      if (ac_store.oncomplete)
{

ac_store.oncomplete(true, key,
backupInput, completeValue);
      }
    }
  } else {
    switch (key) {
      case ESC_KEYNAME: //
escape
      // JER??
ac_suppressCompletions = true;
        ac_selected = -1;
        show = false;
        break;
      case UP_KEYNAME: // up
        if (isDown) {
        // firefox fires arrow
events on both down and press,
but IE only fires
        // then on press.
          ac_selected =
Math.max(numCompletions >= 0 ?
0 : -1, ac_selected - 1);
        }
        break;
      case DOWN_KEYNAME: //
down
        if (isDown) {
          ac_selected =
Math.min(
              ac_max_options -
1, Math.min(numCompletions -
1, ac_selected + 1));
        }
        break;
    }

    if (isDown) {
      switch (key) {
        case ESC_KEYNAME:
        case ENTER_KEYNAME:
        case UP_KEYNAME:
        case DOWN_KEYNAME:
        case RIGHT_KEYNAME:
        case LEFT_KEYNAME:
        case TAB_KEYNAME:
        case SHIFT_KEYNAME:
        case
BACKSPACE_KEYNAME:
        case DELETE_KEYNAME:
          break;
        default: // User typed
some new characters.
          ac_everTyped = true;
      }
    }
  }

  if (ac_focusedInput) {

ac_updateCompletionList(show);
```

```
  }
}

/**
 * called when an option is
clicked on to select that
option.
 */
function
_ac_select(optionIndex) {
  ac_selected = optionIndex;
  ac_complete();
  if (ac_store.oncomplete) {
    ac_store.oncomplete(true,
null, ac_focusedInput,
ac_focusedInput.value);
  }

  // check completions
  ac_checkCompletions();

ac_updateCompletionList(true);
}

function
_ac_mouseover(optionIndex) {
  ac_selected = optionIndex;

ac_updateCompletionList(true);
}

/** perform the substitution
of the currently selected
item. */
function ac_complete() {
  const caret =
ac_getCaretPosition_(ac_focuse
dInput);
  const completion =
ac_completions[ac_selected];

  ac_focusedInput.value =
ac_store.substitute(
      ac_focusedInput.value,
caret,
      ac_lastCompletable,
completion);
```

```
  // When the prefix starts
with '*' we want to return the
complete set of all
  // possible completions. We
treat the ac_lastCompletable
value as empty so
  // that the caret is
correctly calculated (i.e. the
caret should not consider
  // placeholder values like
'*member').
  let new_caret = caret +
completion.value.length;
  if
(!ac_lastCompletable.startsWit
h('*')) {
    // Only consider the
ac_lastCompletable length if
it does not start with '*'
    new_caret = new_caret -
ac_lastCompletable.length;
  }
  // If we inserted something
ending in two quotation marks,
position
  // the cursor between the
quotation marks. If we
inserted a complete term,
  // skip over the trailing
space so that the user is
ready to enter the next
  // term.  If we inserted
just a search operator, leave
the cursor immediately
  // after the colon or equals
and don't skip over the space.
  if
(completion.value.substring(co
mpletion.value.length - 2) ==
'""') {
    new_caret--;
  } else if
(completion.value.substring(co
mpletion.value.length - 1) !=
':' &&

completion.value.substring(com
pletion.value.length - 1) !=
'=') {
    new_caret++; // To account
for the comma.
    new_caret++; // To account
for the space after the comma.
  }
  ac_selected = -1;
  ac_completions = null;
  ac_lastCompletable = null;
  ac_everTyped = false;
  SetCursorPos(window,
ac_focusedInput, new_caret);
}

/**
 * True if the user has ever
typed any actual characters in
the currently
 * focused text field.  False
if they have only clicked,
backspaced, and
 * used the arrow keys.
 */
var ac_everTyped = false;

/**
 * maintains ac_completions,
ac_selected,
ac_lastCompletable.
 * @private
 */
function ac_checkCompletions()
{
  if (ac_focusedInput &&
!ac_suppressCompletions) {
    const caret =
ac_getCaretPosition_(ac_focuse
dInput);
    const completable =
ac_store.completable(ac_focuse
dInput.value, caret);

    // If we already have
completed, then our work here
is done.
    if (completable ==
ac_lastCompletable) {
      return;
    }

    ac_completions = null;
    ac_selected = -1;

    const oldSelected =
      ((ac_selected >= 0 &&
ac_selected <
ac_completions.length) ?

ac_completions[ac_selected].va
lue : null);
    ac_completions =
ac_store.completions(completab
le);
    // Don't offer options for
values that the user has
already used
    // in another part of the
current form.
    ac_completions =
ac_completions.filter((comp)
=>

FindInArray(ac_avoidValues,
comp.value.toLowerCase()) ===
-1);

    ac_selected = oldSelected
? 0 : -1;
    ac_lastCompletable =
completable;
    return;
  }
  ac_lastCompletable = null;
  ac_completions = null;
  ac_selected = -1;
}

/**
 * maintains the completion
list GUI.
 * @private
 */
function
```

```
ac_updateCompletionList(show)
{
  let clist =
document.getElementById('ac-
list');
  const input =
ac_focusedInput;
  if (input) {
    input.setAttribute('aria-
activedescendant', 'ac-status-
row-none');
  }
  let tableEl;
  let tableBody;
  if (show && ac_completions
&& ac_completions.length) {
    if (!clist) {
      clist =
document.createElement('DIV');
      clist.id = 'ac-list';
      clist.style.position =
'absolute';
      clist.style.display =
'none';
      // with 'listbox' and
'option' roles, screenreader
narrates total
      // number of options eg.
'New = issue has not .... 1 of
9'

document.body.appendChild(clis
t);
      tableEl =
document.createElement('table'
);

tableEl.setAttribute('cellpadd
ing', 0);

tableEl.setAttribute('cellspac
ing', 0);
      tableEl.id = 'ac-table';

tableEl.setAttribute('role',
'presentation');
      tableBody =
document.createElement('tbody'
);
      tableBody.id = 'ac-
table-body';

tableEl.appendChild(tableBody)
;

tableBody.setAttribute('role',
'listbox');

clist.appendChild(tableEl);

input.setAttribute('aria-
controls', 'ac-table');

input.setAttribute('aria-
haspopup', 'grid');
    } else {
      tableEl =
document.getElementById('ac-
table');
      tableBody =
document.getElementById('ac-
table-body');
      while
(tableBody.childNodes.length)
{

tableBody.removeChild(tableBod
y.childNodes[0]);
      }
    }

    // If no choice is
selected, then select the
first item, if desired.
    if (ac_selected < 0 &&
ac_store &&
ac_store.autoselectFirstRow())
{
      ac_selected = 0;
    }

    let headerCount= 0;
    for (let i = 0; i <
Math.min(ac_max_options,
ac_completions.length); ++i) {
      if
(ac_completions[i].heading) {
        var rowEl =
document.createElement('tr');

tableBody.appendChild(rowEl);
        const cellEl =
document.createElement('th');

rowEl.appendChild(cellEl);

cellEl.setAttribute('colspan',
2);
        if (headerCount) {

cellEl.appendChild(document.cr
eateElement('br'));
        }
        cellEl.appendChild(

document.createTextNode(ac_com
pletions[i].heading));
        headerCount++;
      } else {
        var rowEl =
document.createElement('tr');

tableBody.appendChild(rowEl);
        if (i == ac_selected)
{
          rowEl.className =
'selected';
        }
        rowEl.id = `ac-status-
row-${i}`;

rowEl.setAttribute('data-
index', i);

rowEl.setAttribute('role',
'option');

rowEl.addEventListener('moused
own', function(event) {

event.preventDefault();
```

```
        });

rowEl.addEventListener('mouseu
p', function(event) {
          let target =
event.target;
          while (target &&
target.tagName != 'TR') {
            target =
target.parentNode;
          }
          const idx =
Number(target.getAttribute('da
ta-index'));
          try {
            _ac_select(idx);
          } finally {
            return false;
          }
        });

rowEl.addEventListener('mouseo
ver', function(event) {
          let target =
event.target;
          while (target &&
target.tagName != 'TR') {
            target =
target.parentNode;
          }
          const idx =
Number(target.getAttribute('da
ta-index'));
          _ac_mouseover(idx);
        });
        const valCellEl =
document.createElement('td');

rowEl.appendChild(valCellEl);
        if
(ac_completions[i].compSpan) {

valCellEl.appendChild(ac_compl
etions[i].compSpan);
        }
        const docCellEl =
document.createElement('td');
```

```
rowEl.appendChild(docCellEl);
        if
(ac_completions[i].docSpan &&

ac_completions[i].docSpan.text
Content) {

docCellEl.appendChild(document
.createTextNode(' = '));

docCellEl.appendChild(ac_compl
etions[i].docSpan);
        }
      }
    }

    // position
    const inputBounds =
nodeBounds(ac_focusedInput);
    clist.style.left =
inputBounds.x + 'px';
    clist.style.top =
(inputBounds.y +
inputBounds.h) + 'px';


window.setTimeout(ac_autoscrol
l, 100);
    input.setAttribute('aria-
activedescendant', `ac-status-
row-${ac_selected}`);
    // Note - we use ''
instead of 'block', since
'block' has odd effects on
    // the screen in IE, and
causes scrollbars to resize
    clist.style.display = '';
  } else {
    tableBody =
document.getElementById('ac-
table-body');
    if (clist && tableBody) {
      clist.style.display =
'none';
      while
(tableBody.childNodes.length)
```

```
{

tableBody.removeChild(tableBod
y.childNodes[0]);
      }
    }
  }
}

// TODO(jrobbins): make arrow
keys and mouse not conflict if
they are
// used at the same time.


/** Scroll the autocomplete
menu to show the currently
selected row. */
function ac_autoscroll() {
  const acList =
document.getElementById('ac-
list');
  const acSelRow =
acList.getElementsByClassName(
'selected')[0];
  const acSelRowTop = acSelRow
? acSelRow.offsetTop : 0;
  const acSelRowHeight =
acSelRow ?
acSelRow.offsetHeight : 0;


  const EXTRA = 8; // Go an
extra few pixels so the next
row is partly exposed.

  if (!acList || !acSelRow)
return;

  // Autoscroll upward if the
selected item is above the
visible area,
  // else autoscroll downward
if the selected item is below
the visible area.
  if (acSelRowTop <
acList.scrollTop) {
    acList.scrollTop =
acSelRowTop - EXTRA;
  } else if (acSelRowTop +
acSelRowHeight + EXTRA >
             acList.scrollTop
+ acList.offsetHeight) {
    acList.scrollTop =
(acSelRowTop + acSelRowHeight
-

acList.offsetHeight + EXTRA);
  }
}


/** the position of the text
caret in the given text field.
 *
 * @param textField an INPUT
node with type=text or a
TEXTAREA node
 * @return an index in [0,
textField.value.length]
 */
function
ac_getCaretPosition_(textField
) {
  if ('INPUT' ==
textField.tagName) {
    let caret =
textField.value.length;

    // chrome/firefox
    if (undefined !=
textField.selectionStart) {
      caret =
textField.selectionEnd;

      // JER: Special
treatment for issue status
field that makes all
      // options show up more
often
      if
(textField.id.startsWith('stat
us')) {
        caret =
textField.selectionStart;
      }
      // ie
    } else if
(document.selection) {
      // get an empty
selection range
      const range =
document.selection.createRange
();
      const
origSelectionLength =
range.text.length;
      // Force selection start
to 0 position

range.moveStart('character', -
caret);
      // the caret end
position is the new selection
length
      caret =
range.text.length;

      // JER: Special
treatment for issue status
field that makes all
      // options show up more
often
      if
(textField.id.startsWith('stat
us')) {
        // The amount that the
selection grew when we forced
start to
        // position 0 is ==
the original start position.
        caret =
range.text.length -
origSelectionLength;
      }
    }

    return caret;
  } else {
    // a textarea
    return
GetCursorPos(window,
textField);
  }
}

function
getTargetFromEvent(event) {
  let targ = event.target ||
event.srcElement;
  if (targ.shadowRoot) {
    // Find the element within
the shadowDOM.
    const path = event.path ||
event.composedPath();
    targ = path[0];
  }
  return targ;
}




/* Copyright 2016 The Chromium
Authors. All Rights Reserved.
 *
 * Use of this source code is
governed by a BSD-style
 * license that can be found
in the LICENSE file or at
 *
https://developers.google.com/
open-source/licenses/bsd
 */
/* eslint-disable camelcase */
/* eslint-disable no-unused-
vars */

/**
 * This file contains the
autocomplete configuration
logic that is
 * specific to the issue
fields of Monorail.  It
depends on ac.js, our
 * modified version of the
autocomplete library.
 */
```

```
/**
 * This is an autocomplete
store that holds the hotlists
of the current user.
 */
let TKR_hotlistsStore;

/**
 * This is an autocomplete
store that holds well-known
issue label
 * values for the current
project.
 */
let TKR_labelStore;

/**
 * Like TKR_labelStore but
stores only label prefixes.
 */
let TKR_labelPrefixStore;

/**
 * Like TKR_labelStore but
adds a trailing comma instead
of replacing.
 */
let TKR_labelMultiStore;

/**
 * This is an autocomplete
store that holds issue
components.
 */
let TKR_componentStore;

/**
 * Like TKR_componentStore but
adds a trailing comma instead
of replacing.
 */
let TKR_componentListStore;

/**
 * This is an autocomplete
store that holds many
```

```
different kinds of
 * items that can be shown in
the artifact search
autocomplete.
 */
let TKR_searchStore;

/**
 * This is similar to
TKR_searchStore, but does not
include any suggestions
 * to use the "me" keyword.
Using "me" is not a good idea
for project canned
 * queries and filter rules.
 */
let TKR_projectQueryStore;

/**
 * This is an autocomplete
store that holds items for the
quick edit
 * autocomplete.
 */
// TODO(jrobbins): add options
for fields and components.
let TKR_quickEditStore;

/**
 * This is a list of label
prefixes that each issue
should only use once.
 * E.g., each issue should
only have one Priority-*
label.  We do not prevent
 * the user from using
multiple such labels, we just
warn the user before
 * they submit.
 */
let TKR_exclPrefixes = [];

/**
 * This is an autocomplete
store that holds custom
permission names that
 * have already been used in
```

```
this project.
 */
let
TKR_customPermissionsStore;


/**
 * This is an autocomplete
store that holds well-known
issue status
 * values for the current
project.
 */
let TKR_statusStore;


/**
 * This is an autocomplete
store that holds the usernames
of all the
 * members of the current
project.  This is used for
autocomplete in
 * the cc-list of an issue,
where many user names can
entered with
 * commas between them.
 */
let TKR_memberListStore;


/**
 * This is an autocomplete
store that holds the projects
that the current
 * user is
contributor/member/owner of.
 */
let TKR_projectStore;

/**
 * This is an autocomplete
store that holds the usernames
of possible
 * issue owners in the current
project.  The list of possible
issue
```

```
 * owners is the same as the
list of project members, but
the behavior
 * of this autocompete store
is different because the issue
owner text
 * field can only accept one
value.
 */
let TKR_ownerStore;


/**
 * This is an autocomplete
store that holds any list of
string for choices.
 */
let TKR_autoCompleteStore;


/**
 * An array of autocomplete
stores used for user-type
custom fields.
 */
const
TKR_userAutocompleteStores =
[];


/**
 * This boolean controls
whether odd-ball status and
labels are treated as
 * a warning or an error.
Normally, it is False.
 */
// TODO(jrobbins): split this
into one option for statuses
and one for labels.
let TKR_restrict_to_known;

/**
 * This substitute function
should be used for multi-
valued autocomplete fields
 * that are delimited by
```

```
commas. When we insert an
autocomplete value, replace
 * an entire search term. Add
a comma and a space after it
if it is a complete
 * search term.
 */
function
TKR_acSubstituteWithComma(inpu
tValue, caret, completable,
completion) {
  let nextTerm = caret;

  // Subtract one in case the
cursor is at the end of the
input, before a comma.
  let prevTerm = caret - 1;
  while (nextTerm <
inputValue.length - 1 &&
inputValue.charAt(nextTerm)
!== ',') {
    nextTerm++;
  }
  // Set this at the position
after the found comma.
  nextTerm++;

  while (prevTerm > 0 && !
[',', '
'].includes(inputValue.charAt(
prevTerm))) {
    prevTerm--;
  }
  if (prevTerm > 0) {
    // Set this boundary after
the found space/comma if it's
not the beginning
    // of the field.
    prevTerm++;
  }

  return
inputValue.substring(0,
prevTerm) +
         completion.value + ',
' +
inputValue.substring(nextTerm)
;
}

/**
 * When the prefix starts with
'*', return the complete set
of all
 * possible completions.
 * @param {string} prefix If
this starts with '*', return
all possible
 * completions.  Otherwise
return null.
 * @param {Array} labelDefs
The array of label names and
docstrings.
 * @return Array of new
_AC_Completions for each
possible completion, or null.
 */
function
TKR_fullComplete(prefix,
labelDefs) {
  if (!prefix.startsWith('*'))
return null;
  const out = [];
  for (let i = 0; i <
labelDefs.length; i++) {
    out.push(new
_AC_Completion(labelDefs[i].na
me,
        labelDefs[i].name,
        labelDefs[i].doc));
  }
  return out;
}


/**
 * Constucts a list of all
completions for both open and
closed
 * statuses, with a header for
each group.
 * @param {string} prefix If
starts with '*', return all
possible completions,
 * else return null.
 * @param {Array}
openStatusDefs The array of
open status values and
 * docstrings.
 * @param {Array}
closedStatusDefs The array of
closed status values
 * and docstrings.
 * @return Array of new
_AC_Completions for each
possible completion, or null.
 */
function
TKR_openClosedComplete(prefix,
openStatusDefs,
closedStatusDefs) {
  if (!prefix.startsWith('*'))
return null;
  const out = [];
  out.push({heading: 'Open
Statuses:'}); // TODO: i18n
  for (var i = 0; i <
openStatusDefs.length; i++) {
    out.push(new
_AC_Completion(openStatusDefs[
i].name,

openStatusDefs[i].name,

openStatusDefs[i].doc));
  }
  out.push({heading: 'Closed
Statuses:'}); // TODO: i18n
  for (var i = 0; i <
closedStatusDefs.length; i++)
{
    out.push(new
_AC_Completion(closedStatusDef
s[i].name,

closedStatusDefs[i].name,

closedStatusDefs[i].doc));
  }
  return out;
}
```

```
function
TKR_setUpHotlistsStore(hotlist
s) {
  const docdict = {};
  const ref_strs = [];

  for (let i = 0; i <
hotlists.length; i++) {
    ref_strs.push(hotlists[i]
['ref_str']);
    docdict[hotlists[i]
['ref_str']] = hotlists[i]
['summary'];
  }

  TKR_hotlistsStore = new
_AC_SimpleStore(ref_strs,
docdict);
  TKR_hotlistsStore.substitute
= TKR_acSubstituteWithComma;
}


/**
 * An array of definitions of
all well-known issue statuses.
Each
 * definition has the name of
the status value, and a
docstring that
 * describes its meaning.
 */
let TKR_statusWords = [];


/**
 * Constuct a new autocomplete
store with all the well-known
issue
 * status values.  The store
has some DIT-specific methods.
 * TODO(jrobbins): would it be
easier to define my own class
to use
 * instead of
```

```
_AC_Simple_Store?
 * @param {Array}
openStatusDefs An array of
definitions of the
 * well-known open status
values.  Each definition has a
name and
 * docstring.
 * @param {Array}
closedStatusDefs An array of
definitions of the
 * well-known closed status
values.  Each definition has a
name and
 * docstring.
 */
function
TKR_setUpStatusStore(openStatu
sDefs, closedStatusDefs) {
  const docdict = {};
  TKR_statusWords = [];
  for (var i = 0; i <
openStatusDefs.length; i++) {
    var status =
openStatusDefs[i];

TKR_statusWords.push(status.na
me);
    docdict[status.name] =
status.doc;
  }
  for (var i = 0; i <
closedStatusDefs.length; i++)
{
    var status =
closedStatusDefs[i];

TKR_statusWords.push(status.na
me);
    docdict[status.name] =
status.doc;
  }

  TKR_statusStore = new
_AC_SimpleStore(TKR_statusWord
s, docdict);
TKR_statusStore.commaCompletes
= false;

  TKR_statusStore.substitute =
  function(inputValue, cursor,
completable, completion) {
    return completion.value;
  };

  TKR_statusStore.completable
= function(inputValue, cursor)
{
    if (!ac_everTyped) return
'*status';
    return inputValue;
  };

  TKR_statusStore.completions
= function(prefix, tofilter) {
    const fullList =
TKR_openClosedComplete(prefix,
        openStatusDefs,
        closedStatusDefs);
    if (fullList) return
fullList;
    return
_AC_SimpleStore.prototype.comp
letions.call(this, prefix,
tofilter);
  };
}


/**
 * Simple function to add a
given item to the list of
items used to construct
 * an "autocomplete store",
and also update the docstring
that describes
 * that item.  They are stored
separately for backward
compatability with
 * autocomplete store logic
that preceeded the
introduction of descriptions.
```

```
 */
function TKR_addACItem(items,
docDict, item, docStr) {
  items.push(item);
  docDict[item] = docStr;
}

/**
 * Adds a group of three items
related to a date field.
 */
function
TKR_addACDateItems(items,
docDict, fieldName,
humanReadable) {
  const today = new Date();
  const todayStr =
(today.getFullYear() + '-' +
(today.getMonth() + 1) + '-' +
    today.getDate());
  TKR_addACItem(items,
docDict, fieldName + '>today-
1',
      humanReadable + ' within
the last N days');
  TKR_addACItem(items,
docDict, fieldName + '>' +
todayStr,
      humanReadable + ' after
the specified date');
  TKR_addACItem(items,
docDict, fieldName + '<today-
1',
      humanReadable + ' more
than N days ago');
}

/**
 * Add several autocomplete
items to a word list that will
be used to construct
 * an autocomplete store.
Also, keep track of
description strings for each
 * item.  A search operator is
prepended to the name of each
item.  The opt_old
```

```
 * and opt_new parameters are
used to transform Key-Value
labels into Key=Value
 * search terms.
 */
function TKR_addACItemList(
    items, docDict, searchOp,
acDefs, opt_old, opt_new) {
  let item;
  for (let i = 0; i <
acDefs.length; i++) {
    const nameAndDoc =
acDefs[i];
    item = searchOp +
nameAndDoc.name;
    if (opt_old) {
      // Preserve any leading
minus-sign.
      item = item.slice(0, 1)
+
item.slice(1).replace(opt_old,
opt_new);
    }
    TKR_addACItem(items,
docDict, item,
nameAndDoc.doc);
  }
}


/**
 * Use information from an
options feed to populate the
artifact search
 * autocomplete menu.  The
order of sections is: custom
fields, labels,
 * components, people, status,
special, dates.  Within each
section,
 * options are ordered
semantically where possible,
or alphabetically
 * if there is no semantic
ordering.  Negated options all
come after
 * all normal options.
 */
function TKR_setUpSearchStore(
    labelDefs, memberDefs,
openDefs, closedDefs,
componentDefs, fieldDefs,
    indMemberDefs) {
  let searchWords = [];
  const searchWordsNeg = [];
  const docDict = {};

  // Treat Key-Value and
OneWord labels separately.
  const keyValueLabelDefs =
[];
  const oneWordLabelDefs = [];
  for (var i = 0; i <
labelDefs.length; i++) {
    const nameAndDoc =
labelDefs[i];
    if
(nameAndDoc.name.indexOf('-')
== -1) {

oneWordLabelDefs.push(nameAndD
oc);
    } else {

keyValueLabelDefs.push(nameAnd
Doc);
    }
  }

  // Autocomplete for custom
fields.
  for (i = 0; i <
fieldDefs.length; i++) {
    const fieldName =
fieldDefs[i]['field_name'];
    const fieldType =
fieldDefs[i]['field_type'];
    if (fieldType ==
'ENUM_TYPE') {
      const choices =
fieldDefs[i]['choices'];

TKR_addACItemList(searchWords,
docDict, fieldName + '=',
choices);

TKR_addACItemList(searchWordsN
eg, docDict, '-' + fieldName +
'=', choices);
    } else if (fieldType ==
'STR_TYPE') {

TKR_addACItem(searchWords,
docDict, fieldName + ':',
          fieldDefs[i]
['docstring']);
    } else if (fieldType ==
'DATE_TYPE') {

TKR_addACItem(searchWords,
docDict, fieldName + ':',
          fieldDefs[i]
['docstring']);

TKR_addACDateItems(searchWords
, docDict, fieldName,
fieldName);
    } else {

TKR_addACItem(searchWords,
docDict, fieldName + '=',
          fieldDefs[i]
['docstring']);
    }
    TKR_addACItem(searchWords,
docDict, 'has:' + fieldName,
        'Issues with any ' +
fieldName + ' value');

TKR_addACItem(searchWordsNeg,
docDict, '-has:' + fieldName,
        'Issues with no ' +
fieldName + ' value');
  }

  // Add suggestions with "me"
first, because otherwise they
may be impossible
  // to reach in a project
that has a lot of members with
emails starting with
```

```
  // "me".
  if
(CS_env['loggedInUserEmail'])
{
    TKR_addACItem(searchWords,
docDict, 'owner:me', 'Issues
owned by me');

TKR_addACItem(searchWordsNeg,
docDict, '-owner:me', 'Issues
not owned by me');
    TKR_addACItem(searchWords,
docDict, 'cc:me', 'Issues that
CC me');

TKR_addACItem(searchWordsNeg,
docDict, '-cc:me', 'Issues
that don\'t CC me');
    TKR_addACItem(searchWords,
docDict, 'reporter:me',
'Issues I reported');

TKR_addACItem(searchWordsNeg,
docDict, '-reporter:me',
'Issues reported by others');
    TKR_addACItem(searchWords,
docDict, 'commentby:me',
        'Issues that I
commented on');

TKR_addACItem(searchWordsNeg,
docDict, '-commentby:me',
        'Issues that I didn\'t
comment on');
  }


TKR_addACItemList(searchWords,
docDict, '',
keyValueLabelDefs, '-', '=');

TKR_addACItemList(searchWordsN
eg, docDict, '-',
keyValueLabelDefs, '-', '=');

TKR_addACItemList(searchWords,
docDict, 'label:',
oneWordLabelDefs);

TKR_addACItemList(searchWordsN
eg, docDict, '-label:',
oneWordLabelDefs);


TKR_addACItemList(searchWords,
docDict, 'component:',
componentDefs);

TKR_addACItemList(searchWordsN
eg, docDict, '-component:',
componentDefs);
  TKR_addACItem(searchWords,
docDict, 'has:component',
      'Issues with any
components specified');

TKR_addACItem(searchWordsNeg,
docDict, '-has:component',
      'Issues with no
components specified');


TKR_addACItemList(searchWords,
docDict, 'owner:',
indMemberDefs);

TKR_addACItemList(searchWordsN
eg, docDict, '-owner:',
indMemberDefs);

TKR_addACItemList(searchWords,
docDict, 'cc:', memberDefs);

TKR_addACItemList(searchWordsN
eg, docDict, '-cc:',
memberDefs);
  TKR_addACItem(searchWords,
docDict, 'has:cc',
      'Issues with any cc\'d
users');

TKR_addACItem(searchWordsNeg,
docDict, '-has:cc',
      'Issues with no cc\'d
```

```
users');

TKR_addACItemList(searchWords,
docDict, 'reporter:',
memberDefs);

TKR_addACItemList(searchWordsN
eg, docDict, '-reporter:',
memberDefs);

TKR_addACItemList(searchWords,
docDict, 'status:', openDefs);

TKR_addACItemList(searchWordsN
eg, docDict, '-status:',
openDefs);

TKR_addACItemList(searchWords,
docDict, 'status:',
closedDefs);

TKR_addACItemList(searchWordsN
eg, docDict, '-status:',
closedDefs);
  TKR_addACItem(searchWords,
docDict, 'has:status',
      'Issues with any
status');

TKR_addACItem(searchWordsNeg,
docDict, '-has:status',
      'Issues with no
status');

  TKR_addACItem(searchWords,
docDict, 'is:blocked',
      'Issues that are
blocked');

TKR_addACItem(searchWordsNeg,
docDict, '-is:blocked',
      'Issues that are not
blocked');
  TKR_addACItem(searchWords,
docDict, 'has:blockedon',
      'Issues that are
blocked');
```

```
TKR_addACItem(searchWordsNeg,
docDict, '-has:blockedon',
      'Issues that are not
blocked');
  TKR_addACItem(searchWords,
docDict, 'has:blocking',
      'Issues that are
blocking other issues');

TKR_addACItem(searchWordsNeg,
docDict, '-has:blocking',
      'Issues that are not
blocking other issues');
  TKR_addACItem(searchWords,
docDict, 'has:mergedinto',
      'Issues that were merged
into other issues');

TKR_addACItem(searchWordsNeg,
docDict, '-has:mergedinto',
      'Issues that were not
merged into other issues');

  TKR_addACItem(searchWords,
docDict, 'is:starred',
      'Starred by me');

TKR_addACItem(searchWordsNeg,
docDict, '-is:starred',
      'Not starred by me');
  TKR_addACItem(searchWords,
docDict, 'stars>10',
      'More than 10 stars');
  TKR_addACItem(searchWords,
docDict, 'stars>100',
      'More than 100 stars');
  TKR_addACItem(searchWords,
docDict, 'summary:',
      'Search within the
summary field');


TKR_addACItemList(searchWords,
docDict, 'commentby:',
memberDefs);
  TKR_addACItem(searchWords,
docDict, 'attachment:',
      'Search within
attachment names');
  TKR_addACItem(searchWords,
docDict, 'attachments>5',
      'Has more than 5
attachments');
  TKR_addACItem(searchWords,
docDict, 'is:open', 'Issues
that are open');

TKR_addACItem(searchWordsNeg,
docDict, '-is:open', 'Issues
that are closed');
  TKR_addACItem(searchWords,
docDict, 'has:owner',
      'Issues with some
owner');

TKR_addACItem(searchWordsNeg,
docDict, '-has:owner',
      'Issues with no owner');
  TKR_addACItem(searchWords,
docDict, 'has:attachments',
      'Issues with some
attachments');
  TKR_addACItem(searchWords,
docDict, 'id:1,2,3',
      'Match only the
specified issues');
  TKR_addACItem(searchWords,
docDict, 'id<100000',
      'Issues with IDs under
100,000');
  TKR_addACItem(searchWords,
docDict, 'blockedon:1',
      'Blocked on the
specified issues');
  TKR_addACItem(searchWords,
docDict, 'blocking:1',
      'Blocking the specified
issues');
  TKR_addACItem(searchWords,
docDict, 'mergedinto:1',
      'Merged into the
specified issues');
  TKR_addACItem(searchWords,
docDict, 'is:ownerbouncing',
      'Issues with owners we
cannot contact');
  TKR_addACItem(searchWords,
docDict, 'is:spam', 'Issues
classified as spam');
  // We do not suggest -
is:spam because it is
implicit.


TKR_addACDateItems(searchWords
, docDict, 'opened',
'Opened');

TKR_addACDateItems(searchWords
, docDict, 'modified',
'Modified');

TKR_addACDateItems(searchWords
, docDict, 'closed',
'Closed');

TKR_addACDateItems(searchWords
, docDict, 'ownermodified',
'Owner field modified');

TKR_addACDateItems(searchWords
, docDict, 'ownerlastvisit',
'Owner last visit');

TKR_addACDateItems(searchWords
, docDict, 'statusmodified',
'Status field modified');
  TKR_addACDateItems(
      searchWords, docDict,
'componentmodified',
'Component field modified');

  TKR_projectQueryStore = new
_AC_SimpleStore(searchWords,
docDict);

  searchWords =
searchWords.concat(searchWords
Neg);
```

```
  TKR_searchStore = new
_AC_SimpleStore(searchWords,
docDict);

  // When we insert an
autocomplete value, replace an
entire search term.
  // Add just a space after it
(not a comma) if it is a
complete search term,
  // or leave the caret
immediately after the
completion if we are just
helping
  // the user with the search
operator.
  TKR_searchStore.substitute =
      function(inputValue,
caret, completable,
completion) {
        let nextTerm = caret;
        while
(inputValue.charAt(nextTerm)
!= ' ' &&
               nextTerm <
inputValue.length) {
          nextTerm++;
        }
        while
(inputValue.charAt(nextTerm)
== ' ' &&
               nextTerm <
inputValue.length) {
          nextTerm++;
        }
        return
inputValue.substring(0, caret
- completable.length) +

completion.value + ' ' +
inputValue.substring(nextTerm)
;
      };

TKR_searchStore.autoselectFirs
tRow =
      function() {
        return false;
      };


TKR_projectQueryStore.substitu
te =
TKR_searchStore.substitute;

TKR_projectQueryStore.autosele
ctFirstRow =
TKR_searchStore.autoselectFirs
tRow;
}


/**
 * Use information from an
options feed to populate the
issue quick edit
 * autocomplete menu.
 */
function
TKR_setUpQuickEditStore(
    labelDefs, memberDefs,
openDefs, closedDefs,
indMemberDefs) {
  const qeWords = [];
  const docDict = {};

  // Treat Key-Value and
OneWord labels separately.
  const keyValueLabelDefs =
[];
  const oneWordLabelDefs = [];
  for (let i = 0; i <
labelDefs.length; i++) {
    const nameAndDoc =
labelDefs[i];
    if
(nameAndDoc.name.indexOf('-')
== -1) {

oneWordLabelDefs.push(nameAndD
oc);
    } else {

keyValueLabelDefs.push(nameAnd
Doc);
    }
  }
  TKR_addACItemList(qeWords,
docDict, '',
keyValueLabelDefs, '-', '=');
  TKR_addACItemList(qeWords,
docDict, '-',
keyValueLabelDefs, '-', '=');
  TKR_addACItemList(qeWords,
docDict, '',
oneWordLabelDefs);
  TKR_addACItemList(qeWords,
docDict, '-',
oneWordLabelDefs);

  TKR_addACItem(qeWords,
docDict, 'owner=me', 'Make me
the owner');
  TKR_addACItem(qeWords,
docDict, 'owner=----', 'Clear
the owner field');
  TKR_addACItem(qeWords,
docDict, 'cc=me', 'CC me on
this issue');
  TKR_addACItem(qeWords,
docDict, 'cc=-me', 'Remove me
from CC list');
  TKR_addACItemList(qeWords,
docDict, 'owner=',
indMemberDefs);
  TKR_addACItemList(qeWords,
docDict, 'cc=', memberDefs);
  TKR_addACItemList(qeWords,
docDict, 'cc=-', memberDefs);
  TKR_addACItemList(qeWords,
docDict, 'status=', openDefs);
  TKR_addACItemList(qeWords,
docDict, 'status=',
closedDefs);
  TKR_addACItem(qeWords,
docDict, 'summary=""', 'Set
the summary field');

  TKR_quickEditStore = new
_AC_SimpleStore(qeWords,
docDict);
```

```
  // When we insert an
autocomplete value, replace an
entire command part.
  // Add just a space after it
(not a comma) if it is a
complete part,
  // or leave the caret
immediately after the
completion if we are just
helping
  // the user with the command
operator.

TKR_quickEditStore.substitute
=
      function(inputValue,
caret, completable,
completion) {
        let nextTerm = caret;
        while
(inputValue.charAt(nextTerm)
!= ' ' &&
               nextTerm <
inputValue.length) {
          nextTerm++;
        }
        while
(inputValue.charAt(nextTerm)
== ' ' &&
               nextTerm <
inputValue.length) {
          nextTerm++;
        }
        return
inputValue.substring(0, caret
- completable.length) +

completion.value + ' ' +
inputValue.substring(nextTerm)
;
      };
}


/**
 * Constuct a new autocomplete
```

```
store with all the project
 * custom permissions.
 * @param {Array}
customPermissions An array of
custom permission names.
 */
function
TKR_setUpCustomPermissionsStor
e(customPermissions) {
  customPermissions =
customPermissions || [];
  const permWords = ['View',
'EditIssue',
'AddIssueComment',
'DeleteIssue'];
  const docdict = {
    'View': '', 'EditIssue':
'', 'AddIssueComment': '',
'DeleteIssue': ''};
  for (let i = 0; i <
customPermissions.length; i++)
{

permWords.push(customPermissio
ns[i]);

docdict[customPermissions[i]]
= '';
  }

  TKR_customPermissionsStore =
new _AC_SimpleStore(permWords,
docdict);


TKR_customPermissionsStore.com
maCompletes = false;


TKR_customPermissionsStore.sub
stitute =
  function(inputValue, cursor,
completable, completion) {
    return completion.value;
  };
}
```

```
/**
 * Constuct a new autocomplete
store with all the well-known
project
 * member user names and real
names.  The store has some
 * monorail-specific methods.
 * TODO(jrobbins): would it be
easier to define my own class
to use
 * instead of
_AC_Simple_Store?
 * @param {Array} memberDefs
an array of member objects.
 * @param {Array}
nonGroupMemberDefs an array of
member objects who are not
groups.
 */
function
TKR_setUpMemberStore(memberDef
s, nonGroupMemberDefs) {
  const memberWords = [];
  const indMemberWords = [];
  const docdict = {};


memberDefs.forEach((memberDef)
=> {

memberWords.push(memberDef.nam
e);
    docdict[memberDef.name] =
null;
  });

nonGroupMemberDefs.forEach((me
mberDef) => {

indMemberWords.push(memberDef.
name);
  });

  TKR_memberListStore = new
_AC_SimpleStore(memberWords,
docdict);
```

```
TKR_memberListStore.completion
s = function(prefix, tofilter)
{
    const fullList =
TKR_fullComplete(prefix,
memberDefs);
    if (fullList) return
fullList;
    return
_AC_SimpleStore.prototype.comp
letions.call(this, prefix,
tofilter);
  };


TKR_memberListStore.completabl
e = function(inputValue,
cursor) {
    if (inputValue == '')
return '*member';
    return
_AC_SimpleStore.prototype.comp
letable.call(this, inputValue,
cursor);
  };


TKR_memberListStore.substitute
= TKR_acSubstituteWithComma;

  TKR_ownerStore = new
_AC_SimpleStore(indMemberWords
, docdict);


TKR_ownerStore.commaCompletes
= false;

  TKR_ownerStore.substitute =
  function(inputValue, cursor,
completable, completion) {
    return completion.value;
  };

  TKR_ownerStore.completions =
function(prefix, tofilter) {
    const fullList =
TKR_fullComplete(prefix,
nonGroupMemberDefs);
    if (fullList) return
fullList;
    return
_AC_SimpleStore.prototype.comp
letions.call(this, prefix,
tofilter);
  };

  TKR_ownerStore.completable =
function(inputValue, cursor) {
    if (!ac_everTyped) return
'*owner';
    return inputValue;
  };
}


/**
 * Constuct one new
autocomplete store for each
user-valued custom
 * field that has a needs_perm
validation requirement, and
thus a
 * list of allowed user
indexes.
 * TODO(jrobbins): would it be
easier to define my own class
to use
 * instead of
_AC_Simple_Store?
 * @param {Array} fieldDefs An
array of field definitions,
only some
 * of which have a
'user_indexes' entry.
 */
function
TKR_setUpUserAutocompleteStore
s(fieldDefs) {
  fieldDefs.forEach((fieldDef)
=> {
    if
```

```
(fieldDef.qualifiedMembers) {
      const us =
makeOneUserAutocompleteStore(f
ieldDef);

TKR_userAutocompleteStores['cu
stom_' + fieldDef['field_id']]
= us;
    }
  });
}

function
makeOneUserAutocompleteStore(f
ieldDef) {
  const memberWords = [];
  const docdict = {};
  for (const member of
fieldDef.qualifiedMembers) {

memberWords.push(member.name);
    docdict[member.name] =
member.doc;
  }

  const userStore = new
_AC_SimpleStore(memberWords,
docdict);
  userStore.commaCompletes =
false;

  userStore.substitute =
  function(inputValue, cursor,
completable, completion) {
    return completion.value;
  };

  userStore.completions =
function(prefix, tofilter) {
    const fullList =
TKR_fullComplete(prefix,
fieldDef.qualifiedMembers);
    if (fullList) return
fullList;
    return
_AC_SimpleStore.prototype.comp
letions.call(this, prefix,
tofilter);
  };

  userStore.completable =
function(inputValue, cursor) {
    if (!ac_everTyped) return
'*custom';
    return inputValue;
  };

  return userStore;
}


/**
 * Constuct a new autocomplete
store with all the components.
 * The store has some
monorail-specific methods.
 * @param {Array}
componentDefs An array of
definitions of components.
 */
function
TKR_setUpComponentStore(compon
entDefs) {
  const componentWords = [];
  const docdict = {};
  for (let i = 0; i <
componentDefs.length; i++) {
    const component =
componentDefs[i];

componentWords.push(component.
name);
    docdict[component.name] =
component.doc;
  }

  const completions =
function(prefix, tofilter) {
    const fullList =
TKR_fullComplete(prefix,
componentDefs);
    if (fullList) return
fullList;
    return
```

```
_AC_SimpleStore.prototype.comp
letions.call(this, prefix,
tofilter);
  };
  const completable =
function(inputValue, cursor) {
    if (inputValue == '')
return '*component';
    return
_AC_SimpleStore.prototype.comp
letable.call(this, inputValue,
cursor);
  };

  TKR_componentStore = new
_AC_SimpleStore(componentWords
, docdict);

TKR_componentStore.commaComple
tes = false;

TKR_componentStore.substitute
=
  function(inputValue, cursor,
completable, completion) {
    return completion.value;
  };

TKR_componentStore.completions
= completions;

TKR_componentStore.completable
= completable;

  TKR_componentListStore = new
_AC_SimpleStore(componentWords
, docdict);

TKR_componentListStore.commaCo
mpletes = false;

TKR_componentListStore.substit
ute =
TKR_acSubstituteWithComma;

TKR_componentListStore.complet
ions = completions;
```

```
TKR_componentListStore.complet
able = completable;
}


/**
 * An array of definitions of
all well-known issue labels.
Each
 * definition has the name of
the label, and a docstring
that
 * describes its meaning.
 */
let TKR_labelWords = [];


/**
 * Constuct a new autocomplete
store with all the well-known
issue
 * labels for the current
project.  The store has some
DIT-specific methods.
 * TODO(jrobbins): would it be
easier to define my own class
to use
 * instead of
_AC_Simple_Store?
 * @param {Array} labelDefs An
array of definitions of the
project
 * members.  Each definition
has a name and docstring.
 */
function
TKR_setUpLabelStore(labelDefs)
{
  TKR_labelWords = [];
  const TKR_labelPrefixes =
[];
  const labelPrefs = new
Set();
  const docdict = {};
  for (let i = 0; i <
labelDefs.length; i++) {
    const label =
labelDefs[i];

TKR_labelWords.push(label.name
);

TKR_labelPrefixes.push(label.n
ame.split('-')[0]);
    docdict[label.name] =
label.doc;

labelPrefs.add(label.name.spli
t('-')[0]);
  }
  const labelPrefArray =
Array.from(labelPrefs);
  const labelPrefDefs =
labelPrefArray.map((s) =>
({name: s, doc: ''}));

  TKR_labelStore = new
_AC_SimpleStore(TKR_labelWords
, docdict);


TKR_labelStore.commaCompletes
= false;
  TKR_labelStore.substitute =
  function(inputValue, cursor,
completable, completion) {
    return completion.value;
  };

  TKR_labelPrefixStore = new
_AC_SimpleStore(TKR_labelPrefi
xes);


TKR_labelPrefixStore.commaComp
letes = false;

TKR_labelPrefixStore.substitut
e =
  function(inputValue, cursor,
completable, completion) {
    return completion.value;
  };
  TKR_labelMultiStore = new
_AC_SimpleStore(TKR_labelWords
, docdict);


TKR_labelMultiStore.substitute
= TKR_acSubstituteWithComma;

  const completable =
function(inputValue, cursor) {
    if (cursor === 0) {
      return '*label'; // Show
every well-known label that is
not redundant.
    }
    let start = 0;
    for (let i = cursor; --i
>= 0;) {
      const c =
inputValue.charAt(i);
      if (c === ' ' || c ===
',') {
        start = i + 1;
        break;
      }
    }
    const questionPos =
inputValue.indexOf('?');
    if (questionPos >= 0) {
      // Ignore any "?"
character and anything after
it.
      inputValue =
inputValue.substring(start,
questionPos);
    }
    let result =
inputValue.substring(start,
cursor);
    if
(inputValue.lastIndexOf('-') >
0 && !ac_everTyped) {
      // Act like a menu:
offer all alternative values
for the same prefix.
      result =
inputValue.substring(
          start,
Math.min(cursor,
inputValue.lastIndexOf('-')));
    }
    if
(inputValue.startsWith('Restri
ct-') && !ac_everTyped) {
      // If user is in the
middle of 2nd part, use that
to narrow the choices.
      result = inputValue;
      // If they completed 2nd
part, give all choices
matching 2-part prefix.
      if
(inputValue.lastIndexOf('-') >
8) {
        result =
inputValue.substring(
            start,
Math.min(cursor,
inputValue.lastIndexOf('-') +
1));
      }
    }

    return result;
  };

  const computeAvoid =
function() {
    const labelTextFields =
Array.from(

document.querySelectorAll('.la
belinput'));
    const otherTextFields =
labelTextFields.filter(
        (tf) => (tf !==
ac_focusedInput && tf.value));
    return
otherTextFields.map((tf) =>
tf.value);
  };
  const completions =
function(labeldic) {
    return function(prefix,
tofilter) {
      let comps =
TKR_fullComplete(prefix,
labeldic);
      if (comps === null) {
        comps =
_AC_SimpleStore.prototype.comp
letions.call(
            this, prefix,
tofilter);
      }

      const filteredComps =
[];
      for (const completion of
comps) {
        const completionLower
=
completion.value.toLowerCase()
;
        const labelPrefix =
completionLower.split('-')[0];
        let alreadyUsed =
false;
        const isExclusive =
FindInArray(TKR_exclPrefixes,
labelPrefix) !== -1;
        if (isExclusive) {
          for (const usedLabel
of ac_avoidValues) {
            if
(usedLabel.startsWith(labelPre
fix + '-')) {
              alreadyUsed =
true;
              break;
            }
          }
        }
        if (!alreadyUsed) {

filteredComps.push(completion)
;
        }
      }

      return filteredComps;
    };
  };

  TKR_labelStore.computeAvoid
= computeAvoid;
  TKR_labelStore.completable =
completable;
  TKR_labelStore.completions =
completions(labelDefs);


TKR_labelPrefixStore.completab
le = completable;

TKR_labelPrefixStore.completio
ns =
completions(labelPrefDefs);


TKR_labelMultiStore.completabl
e = completable;

TKR_labelMultiStore.completion
s = completions(labelDefs);
}


/**
 * Constuct a new autocomplete
store with the given strings
as choices.
 * @param {Array} choices An
array of autocomplete choices.
 */
function
TKR_setUpAutoCompleteStore(cho
ices) {
  TKR_autoCompleteStore = new
_AC_SimpleStore(choices);
  const choicesDefs = [];
  for (let i = 0; i <
choices.length; ++i) {
    choicesDefs.push({'name':
choices[i], 'doc': ''});
```

```
  }

  /**
   * Override the default
completions() function to
return a list of
   * available choices.  It
proactively shows all choices
when the user has
   * not yet typed anything.
It stops offering choices if
the text field
   * has a pretty long string
in it already.  It does not
offer choices that
   * have already been chosen.
   */

TKR_autoCompleteStore.completi
ons = function(prefix,
tofilter) {
    if (prefix.length > 18) {
      return [];
    }
    let comps =
TKR_fullComplete(prefix,
choicesDefs);
    if (comps == null) {
      comps =
_AC_SimpleStore.prototype.comp
letions.call(
          this, prefix,
tofilter);
    }

    const usedComps = {};
    const textFields =
document.getElementsByTagName(
'input');
    for (var i = 0; i <
textFields.length; ++i) {
      if
(textFields[i].classList.conta
ins('autocomplete')) {

usedComps[textFields[i].value]
= true;
```

```
      }
    }
    const unusedComps = [];
    for (i = 0; i <
comps.length; ++i) {
      if
(!usedComps[comps[i].value]) {

unusedComps.push(comps[i]);
      }
    }

    return unusedComps;
  };

  /**
   * Override the default
completable() function with
one that gives a
   * special value when the
user has not yet typed
anything.  This
   * causes TKR_fullComplete()
to show all choices.  Also,
always consider
   * the whole textfield value
as an input to completion
matching.  Otherwise,
   * it would only consider
the part after the last comma
(which makes sense
   * for gmail To: and Cc:
address fields).
   */

TKR_autoCompleteStore.completa
ble = function(inputValue,
cursor) {
    if (inputValue == '') {
      return '*ac';
    }
    return inputValue;
  };

  /**
   * Override the default
substitute() function to
```

```
completely replace the
   * contents of the text
field when the user selects a
completion. Otherwise,
   * it would append, much
like the Gmail To: and Cc:
fields append autocomplete
   * selections.
   */

TKR_autoCompleteStore.substitu
te =
  function(inputValue, cursor,
completable, completion) {
    return completion.value;
  };

  /**
   * We consider the whole
textfield to be one value, not
a comma separated
   * list.  So, typing a ','
should not trigger an
autocomplete selection.
   */

TKR_autoCompleteStore.commaCom
pletes = false;
}


/**
 * XMLHTTP object used to
fetch autocomplete options
from the server.
 */
const TKR_optionsXmlHttp =
undefined;

/**
 * Contact the server to fetch
the set of autocomplete
options for the
 * projects the user is
contributor/member/owner of.
 * @param {multiValue} boolean
If set to true, the
```

```
projectStore is configured to
 * have support for multi-
values (useful for example for
saved queries where
 * a query can apply to
multiple projects).
 */
function
TKR_fetchUserProjects(multiVal
ue) {
  // Set a request token to
prevent XSRF leaking of user
project lists.
  const userRefs =
[{displayName:
window.CS_env.loggedInUserEmai
l}];
  const userProjectsPromise =
window.prpcClient.call(
      'monorail.Users',
'GetUsersProjects',
{userRefs});

userProjectsPromise.then((resp
onse) => {
    const userProjects =
response.usersProjects[0];
    const projects =
(userProjects.ownerOf || [])

.concat(userProjects.memberOf
|| [])

.concat(userProjects.contribut
orTo || []);
    projects.sort();
    if (projects) {

TKR_setUpProjectStore(projects
, multiValue);
    }
  });
}


/**
 * Constuct a new autocomplete
```

```
store with all the projects
that the
 * current user has visibility
into. The store has some
monorail-specific
 * methods.
 * @param {Array} projects An
array of project names.
 * @param {boolean} multiValue
Determines whether the store
should support
 *                  multiple
values.
 */
function
TKR_setUpProjectStore(projects
, multiValue) {
  const projectsDefs = [];
  const docdict = {};
  for (let i = 0; i <
projects.length; ++i) {
    projectsDefs.push({'name':
projects[i], 'doc': ''});
    docdict[projects[i]] = '';
  }

  TKR_projectStore = new
_AC_SimpleStore(projects,
docdict);

TKR_projectStore.commaComplete
s = !multiValue;

  if (multiValue) {

TKR_projectStore.substitute =
TKR_acSubstituteWithComma;
  } else {

TKR_projectStore.substitute =
      function(inputValue,
cursor, completable,
completion) {
        return
completion.value;
      };
  }
  TKR_projectStore.completions
= function(prefix, tofilter) {
    const fullList =
TKR_fullComplete(prefix,
projectsDefs);
    if (fullList) return
fullList;
    return
_AC_SimpleStore.prototype.comp
letions.call(this, prefix,
tofilter);
  };

  TKR_projectStore.completable
= function(inputValue, cursor)
{
    if (inputValue == '')
return '*project';
    if (multiValue) {
      return
_AC_SimpleStore.prototype.comp
letable.call(
          this, inputValue,
cursor);
    } else {
      return inputValue;
    }
  };
}


/**
 * Convert the object
resulting of a
monorail.Projects ListStatuses
to
 * the format expected by
TKR_populateAutocomplete.
 * @param {object}
statusesResponse A pRPC
ListStatusesResponse object.
 */
function
TKR_convertStatuses(statusesRe
sponse) {
  const statusDefs =
statusesResponse.statusDefs ||
[];
  const jsonData = {};

  // Split statusDefs into
open and closed name-doc
objects.
  jsonData.open = [];
  jsonData.closed = [];
  for (const s of statusDefs)
{
    if (!s.deprecated) {
      const item = {
        name: s.status,
        doc: s.docstring,
      };
      if (s.meansOpen) {

jsonData.open.push(item);
      } else {

jsonData.closed.push(item);
      }
    }
  }

  jsonData.strict =
statusesResponse.restrictToKno
wn;

  return jsonData;
}


/**
 * Convert the object
resulting of a
monorail.Projects
ListComponents to
 * the format expected by
TKR_populateAutocomplete.
 * @param {object}
componentsResponse A pRPC
ListComponentsResponse object.
 */
function
TKR_convertComponents(componen
tsResponse) {
  const componentDefs =
(componentsResponse.componentD
efs || []);
  const jsonData = {};

  // Filter out deprecated
components and normalize to
name-doc object.
  jsonData.components = [];
  for (const c of
componentDefs) {
    if (!c.deprecated) {

jsonData.components.push({
        name: c.path,
        doc: c.docstring,
      });
    }
  }

  return jsonData;
}


/**
 * Convert the object
resulting of a
monorail.Projects
GetLabelOptions
 * call to the format expected
by TKR_populateAutocomplete.
 * @param {object}
labelsResponse A pRPC
GetLabelOptionsResponse.
 * @param {Array<FieldDef>=}
fieldDefs FieldDefs from a
project config, used to
 *   mask labels that are used
to implement custom enum
fields.
 */
function
TKR_convertLabels(labelsRespon
se, fieldDefs = []) {
  const labelDefs =
(labelsResponse.labelDefs ||
[]);
  const exclusiveLabelPrefixes
=
(labelsResponse.exclusiveLabel
Prefixes || []);
  const jsonData = {};

  const maskedLabels = new
Set();
  fieldDefs.forEach((fd) => {
    if (fd.enumChoices) {

fd.enumChoices.forEach(({label
}) => {

maskedLabels.add(`${fd.fieldRe
f.fieldName}-${label}`);
      });
    }
  });

  jsonData.labels =
labelDefs.filter(({label}) =>
!maskedLabels.has(label)).map(
      (label) => ({name:
label.label, doc:
label.docstring}));

  jsonData.excl_prefixes =
exclusiveLabelPrefixes.map(
      (prefix) =>
prefix.toLowerCase());

  return jsonData;
}


/**
 * Convert the object
resulting of a
monorail.Projects
GetVisibleMembers
 * call to the format expected
by TKR_populateAutocomplete.
 * @param {object?}
visibleMembersResponse A pRPC
GetVisibleMembersResponse.
```

```
 * @return {{memberEmails:
{name: string},
nonGroupEmails: {name:
string}}}
 */
function
TKR_convertVisibleMembers(visi
bleMembersResponse) {
  if (!visibleMembersResponse)
{
    visibleMembersResponse =
{};
  }
  const groupRefs =
(visibleMembersResponse.groupR
efs || []);
  const userRefs =
(visibleMembersResponse.userRe
fs || []);
  const jsonData = {};

  const groupEmails = new
Set(groupRefs.map(
      (groupRef) =>
groupRef.displayName));

  jsonData.memberEmails =
userRefs.map(
      (userRef) => ({name:
userRef.displayName}));
  jsonData.nonGroupEmails =
jsonData.memberEmails.filter(
      (memberEmail) =>
!groupEmails.has(memberEmail))
;

  return jsonData;
}


/**
 * Convert the object
resulting of a
monorail.Projects ListFields
to
 * the format expected by
TKR_populateAutocomplete.
```

```
 * @param {object}
fieldsResponse A pRPC
ListFieldsResponse object.
 */
function
TKR_convertFields(fieldsRespon
se) {
  const fieldDefs =
(fieldsResponse.fieldDefs ||
[]);
  const jsonData = {};

  jsonData.fields =
fieldDefs.map((field) =>
    ({
      field_id:
field.fieldRef.fieldId,
      field_name:
field.fieldRef.fieldName,
      field_type:
field.fieldRef.type,
      docstring:
field.docstring,
      choices:
(field.enumChoices || []).map(
          (choice) => ({name:
choice.label, doc:
choice.docstring})),
      qualifiedMembers:
(field.userChoices || []).map(
          (userRef) => ({name:
userRef.displayName})),
    }),
  );

  return jsonData;
}


/**
 * Convert the object
resulting of a
monorail.Features
ListHotlistsByUser
 * call to the format expected
by TKR_populateAutocomplete.
 * @param {Array<HotlistV0>}
```

```
hotlists A lists of hotlists
 * @return {Array<{ref_str:
string, summary: string}>}
 */
function
TKR_convertHotlists(hotlists)
{
  if (hotlists === undefined)
{
    return [];
  }

  const seen = new Set();
  const ambiguousNames = new
Set();

  hotlists.forEach((hotlist)
=> {
    if
(seen.has(hotlist.name)) {

ambiguousNames.add(hotlist.nam
e);
    }
    seen.add(hotlist.name);
  });

  return
hotlists.map((hotlist) => {
    let ref_str =
hotlist.name;
    if
(ambiguousNames.has(hotlist.na
me)) {
      ref_str =
hotlist.owner_ref.display_name
+ ':' + ref_str;
    }
    return {ref_str: ref_str,
summary: hotlist.summary};
  });
}


/**
 * Initializes hotlists in
autocomplete store.
```

```
 * @param {Array<HotlistV0>}
hotlists
 */
function
TKR_populateHotlistAutocomplet
e(hotlists) {

TKR_setUpHotlistsStore(TKR_con
vertHotlists(hotlists));
}


/**
 * Add project config data
that's already been fetched to
the legacy
 * autocomplete.
 * @param {Config}
projectConfig Returned
projectConfig data.
 * @param
{GetVisibleMembersResponse}
visibleMembers
 * @param {Array<string>}
customPermissions
 */
function
TKR_populateAutocomplete(proje
ctConfig, visibleMembers,
    customPermissions = []) {
  const {statusDefs,
componentDefs, labelDefs,
fieldDefs,
    exclusiveLabelPrefixes,
projectName} = projectConfig;

  const {memberEmails,
nonGroupEmails} =

TKR_convertVisibleMembers(visi
bleMembers);

TKR_setUpMemberStore(memberEma
ils, nonGroupEmails);

TKR_prepOwnerField(memberEmail
s);
```

```
  const {open, closed, strict}
=
TKR_convertStatuses({statusDef
s});
  TKR_setUpStatusStore(open,
closed);
  TKR_restrict_to_known =
strict;

  const {components} =
TKR_convertComponents({compone
ntDefs});

TKR_setUpComponentStore(compon
ents);

  const {excl_prefixes,
labels} = TKR_convertLabels(
      {labelDefs,
exclusiveLabelPrefixes},
fieldDefs);
  TKR_exclPrefixes =
excl_prefixes;
  TKR_setUpLabelStore(labels);

  const {fields} =
TKR_convertFields({fieldDefs})
;

TKR_setUpUserAutocompleteStore
s(fields);

  /* QuickEdit is not yet in
Monorail.
crbug.com/monorail/1926
  TKR_setUpQuickEditStore(
      jsonData.labels,
jsonData.memberEmails,
jsonData.open,
jsonData.closed,

jsonData.nonGroupEmails);
  */

  // We need to wait until
both exclusive prefixes (in
configPromise) and
  // labels (in labelsPromise)
have been read.

TKR_prepLabelAC(TKR_labelField
IDPrefix);

  TKR_setUpSearchStore(
      labels, memberEmails,
open, closed,
      components, fields,
nonGroupEmails);


TKR_setUpCustomPermissionsStor
e(customPermissions);
}



/* Copyright 2016 The Chromium
Authors. All Rights Reserved.
 *
 * Use of this source code is
governed by a BSD-style
 * license that can be found
in the LICENSE file or at
 *
https://developers.google.com/
open-source/licenses/bsd
 */
/* eslint-disable camelcase */
/* eslint-disable no-unused-
vars */

/**
  * Sets up the legacy
autocomplete editing widget on
DOM elements that are
  * set to use it.
  */
function TKR_install_ac() {
  _ac_install();

  _ac_register(function(input,
event) {
    if
(input.id.startsWith('hotlists
')) return TKR_hotlistsStore;
    if
(input.id.startsWith('search')
) return TKR_searchStore;
    if
(input.id.startsWith('query_')
||
input.id.startsWith('predicate
_')) {
      return
TKR_projectQueryStore;
    }
    if
(input.id.startsWith('cmd'))
return TKR_quickEditStore;
    if
(input.id.startsWith('labelPre
fix')) return
TKR_labelPrefixStore;
    if
(input.id.startsWith('label')
&& input.id != 'labelsInput')
return TKR_labelStore;
    if (input.dataset.acType
=== 'label' && input.id !=
'labelsInput') return
TKR_labelMultiStore;
    if
((input.id.startsWith('compone
nt') || input.dataset.acType
=== 'component')
      && input.id !=
'componentsInput') return
TKR_componentListStore;
    if
(input.id.startsWith('status')
) return TKR_statusStore;
    if
(input.id.startsWith('member')
|| input.dataset.acType ===
'member') return
TKR_memberListStore;

    if (input.id ==
'admin_names_editor') return
TKR_memberListStore;
    if
```

```
(input.id.startsWith('owner')
&& input.id != 'ownerInput')
return TKR_ownerStore;
    if (input.name ==
'needs_perm' || input.name ==
'grants_perm') {
      return
TKR_customPermissionsStore;
    }
    if (input.id ==
'owner_editor' ||
input.dataset.acType ===
'owner') return
TKR_ownerStore;
    if
(input.className.indexOf('user
autocomplete') != -1) {
      const customFieldIDStr =
input.name;
      const uac =
TKR_userAutocompleteStores[cus
tomFieldIDStr];
      if (uac) return uac;
      return TKR_ownerStore;
    }
    if
(input.className.indexOf('auto
complete') != -1) {
      return
TKR_autoCompleteStore;
    }
    if
(input.id.startsWith('copy_to'
) ||
input.id.startsWith('move_to')
||

input.id.startsWith('new_saved
query_projects') ||

input.id.startsWith('savedquer
y_projects')) {
      return TKR_projectStore;
    }
  });
};
```

```
/* Copyright 2016 The Chromium
Authors. All Rights Reserved.
 *
 * Use of this source code is
governed by a BSD-style
 * license that can be found
in the LICENSE file or at
 *
https://developers.google.com/
open-source/licenses/bsd
 */
/* eslint-disable no-var */
/* eslint-disable prefer-const
*/

/**
 * This file contains JS
functions that support various
issue editing
 * features of Monorail.
These editing features
include: selecting
 * issues on the issue list
page, adding attachments,
expanding and
 * collapsing the issue
editing form, and starring
issues.
 *
 * Browser compatability: IE6,
IE7, FF1.0+, Safari.
 */


/**
 * Here are some string
constants that are used
repeatedly in the code.
 */
let TKR_SELECTED_CLASS =
'selected';
let TKR_UNDEF_CLASS = 'undef';
let TKR_NOVEL_CLASS = 'novel';
let TKR_EXCL_CONFICT_CLASS =
'exclconflict';
let TKR_QUESTION_MARK_CLASS =
'questionmark';
let TKR_ATTACHPROMPT_ID =
'attachprompt';
let TKR_ATTACHAFILE_ID =
'attachafile';
let TKR_ATTACHMAXSIZE_ID =
'attachmaxsize';
let
TKR_CURRENT_TEMPLATE_INDEX_ID
= 'current_template_index';
let
TKR_PROMPT_MEMBERS_ONLY_CHECKB
OX_ID =
'members_only_checkbox';
let
TKR_PROMPT_SUMMARY_EDITOR_ID =
'summary_editor';
let
TKR_PROMPT_SUMMARY_MUST_BE_EDI
TED_CHECKBOX_ID =

'summary_must_be_edited_checkb
ox';
let
TKR_PROMPT_CONTENT_EDITOR_ID =
'content_editor';
let
TKR_PROMPT_STATUS_EDITOR_ID =
'status_editor';
let TKR_PROMPT_OWNER_EDITOR_ID
= 'owner_editor';
let
TKR_PROMPT_ADMIN_NAMES_EDITOR_
ID = 'admin_names_editor';
let
TKR_OWNER_DEFAULTS_TO_MEMBER_C
HECKBOX_ID =

'owner_defaults_to_member_chec
kbox';
let
TKR_OWNER_DEFAULTS_TO_MEMBER_A
REA_ID =

'owner_defaults_to_member_area
';
let
```

```
TKR_COMPONENT_REQUIRED_CHECKBO
X_ID =

'component_required_checkbox';
let
TKR_PROMPT_COMPONENTS_EDITOR_I
D = 'components_editor';
let TKR_FIELD_EDITOR_ID_PREFIX
= 'tmpl_custom_';
let
TKR_PROMPT_LABELS_EDITOR_ID_PR
EFIX = 'label';
let TKR_CONFIRMAREA_ID =
'confirmarea';
let TKR_DISCARD_YOUR_CHANGES =
'Discard your changes?';
// Note, users cannot enter
'<'.
let TKR_DELETED_PROMPT_NAME =
'<DELETED>';
// Display warning if labels
contain the following
prefixes.
// The following list is the
same as
tracker_constants.RESERVED_PRE
FIXES except
// for the 'hotlist' prefix.
'hostlist' will be added when
it comes a full
// feature and when projects
that use 'Hostlist-*' labels
are transitioned off.
let
TKR_LABEL_RESERVED_PREFIXES =
[
  'id', 'project', 'reporter',
'summary', 'status', 'owner',
'cc',
  'attachments', 'attachment',
'component', 'opened',
'closed',
  'modified', 'is', 'has',
'blockedon', 'blocking',
'blocked', 'mergedinto',
  'stars', 'starredby',
'description', 'comment',
'commentby', 'label',
  'rank', 'explicit_status',
'derived_status',
'explicit_owner',
  'derived_owner',
'explicit_cc', 'derived_cc',
'explicit_label',
  'derived_label',
'last_comment_by',
'exact_component',
  'explicit_component',
'derived_component'];


/**
 * Appends a given child
element to the DOM based on
parameters.
 * @param {HTMLElement}
parentEl
 * @param {string} tag
 * @param {string}
optClassName
 * @param {string} optID
 * @param {string} optText
 * @param {string} optStyle
*/
function
TKR_createChild(parentEl, tag,
optClassName, optID, optText,
optStyle) {
  let el =
document.createElement(tag);
  if (optClassName)
el.classList.add(optClassName)
;
  if (optID) el.id = optID;
  if (optText) el.textContent
= optText;
  if (optStyle)
el.setAttribute('style',
optStyle);
  parentEl.appendChild(el);
  return el;
}

/**
```

```
 * Select all the issues on
the issue list page.
 */
function TKR_selectAllIssues()
{
  TKR_selectIssues(true);
}


/**
 * Function to deselect all
the issues on the issue list
page.
 */
function
TKR_selectNoneIssues() {
  TKR_selectIssues(false);
}


/**
 * Function to select or
deselect all the issues on the
issue list page.
 * @param {boolean} checked
True means select issues,
False means deselect.
 */
function
TKR_selectIssues(checked) {
  let table =
$('resultstable');
  for (let r = 0; r <
table.rows.length; ++r) {
    let row = table.rows[r];
    let firstCell =
row.cells[0];
    if (firstCell.tagName ==
'TD') {
      for (let e = 0; e <
firstCell.childNodes.length;
++e) {
        let element =
firstCell.childNodes[e];
        if (element.tagName ==
'INPUT' && element.type ==
'checkbox') {
```

```
          element.checked =
checked ? 'checked' : '';
          if (checked) {

row.classList.add(TKR_SELECTED
_CLASS);
          } else {

row.classList.remove(TKR_SELEC
TED_CLASS);
          }
        }
      }
    }
  }
}


/**
 * The ID number to append to
the next dynamically created
file upload field.
 */
let TKR_nextFileID = 1;


/**
 * Function to dynamically
create a new attachment upload
field add
 * insert it into the page
DOM.
 * @param {string} id The id
of the parent HTML element.
 *
 * TODO(lukasperaza): use
different nextFileID for
separate forms on same page,
 *  e.g. issue update form and
issue description update form
 */
function
TKR_addAttachmentFields(id,
attachprompt_id,
    attachafile_id,
attachmaxsize_id) {
  if (TKR_nextFileID >= 16) {
    return;
  }
  if (typeof attachprompt_id
=== 'undefined') {
    attachprompt_id =
TKR_ATTACHPROMPT_ID;
  }
  if (typeof attachafile_id
=== 'undefined') {
    attachafile_id =
TKR_ATTACHAFILE_ID;
  }
  if (typeof attachmaxsize_id
=== 'undefined') {
    attachmaxsize_id =
TKR_ATTACHMAXSIZE_ID;
  }
  let el = $(id);
  el.style.marginTop = '4px';
  let div =
document.createElement('div');
  var id = 'file' +
TKR_nextFileID;
  let label =
TKR_createChild(div, 'label',
null, null, 'Attach file:');
  label.setAttribute('for',
id);
  let input = TKR_createChild(
      div, 'input', null, id,
null, 'width:auto;margin-
left:17px');
  input.setAttribute('type',
'file');
  input.name = id;
  let removeLink =
TKR_createChild(
      div, 'a', null, null,
'Remove', 'font-size:x-
small');
  removeLink.href = '#';

removeLink.addEventListener('c
lick', function(event) {
    let target = event.target;
    $(attachafile_id).focus();
target.parentNode.parentNode.r
emoveChild(target.parentNode);
    event.preventDefault();
  });
  el.appendChild(div);

el.querySelector('input').focu
s();
  ++TKR_nextFileID;
  if (TKR_nextFileID < 16) {

$(attachafile_id).textContent
= 'Attach another file';
  } else {

$(attachprompt_id).style.displ
ay = 'none';
  }

$(attachmaxsize_id).style.disp
lay = '';
}


/**
 * Function to display the
form so that the user can
update an issue.
 */
function
TKR_openIssueUpdateForm() {

TKR_showHidden($('makechangesa
rea'));

TKR_goToAnchor('makechanges');
  TKR_forceProperTableWidth();
  window.setTimeout(
      function() {

document.getElementById('addCo
mmentTextArea').focus();
      },
      100);
}
```

```
/**
 * The index of the template
that is currently selected for
editing
 * on the administration page
for issues.
 */
let TKR_currentTemplateIndex =
0;


/**
 * Array of field IDs that are
defined in the current
project, set by call to
setFieldIDs().
 */
let TKR_fieldIDs = [];


function
TKR_setFieldIDs(fieldIDs) {
  TKR_fieldIDs = fieldIDs;
}


/**
 * This function displays the
appropriate template text in a
text field.
 * It is called after the user
has selected one template to
view/edit.
 * @param {Element} widget The
list widget containing the
list of templates.
 */
function
TKR_selectTemplate(widget) {

TKR_showHidden($('edit_panel')
);
  TKR_currentTemplateIndex =
widget.value;

$(TKR_CURRENT_TEMPLATE_INDEX_I
D).value =
TKR_currentTemplateIndex;

  let content_editor =
$(TKR_PROMPT_CONTENT_EDITOR_ID
);

TKR_makeDefined(content_editor
);

  let can_edit = $('can_edit_'
+
TKR_currentTemplateIndex).valu
e == 'yes';
  let disabled = can_edit ? ''
: 'disabled';


$(TKR_PROMPT_MEMBERS_ONLY_CHEC
KBOX_ID).disabled = disabled;

$(TKR_PROMPT_MEMBERS_ONLY_CHEC
KBOX_ID).checked = $(
      'members_only_' +
TKR_currentTemplateIndex).valu
e == 'yes';

$(TKR_PROMPT_SUMMARY_EDITOR_ID
).disabled = disabled;

$(TKR_PROMPT_SUMMARY_EDITOR_ID
).value = $(
      'summary_' +
TKR_currentTemplateIndex).valu
e;

$(TKR_PROMPT_SUMMARY_MUST_BE_E
DITED_CHECKBOX_ID).disabled =
disabled;

$(TKR_PROMPT_SUMMARY_MUST_BE_E
DITED_CHECKBOX_ID).checked =
$(

'summary_must_be_edited_' +
TKR_currentTemplateIndex).valu
e == 'yes';
  content_editor.disabled =
disabled;
  content_editor.value =
$('content_' +
TKR_currentTemplateIndex).valu
e;

$(TKR_PROMPT_STATUS_EDITOR_ID)
.disabled = disabled;

$(TKR_PROMPT_STATUS_EDITOR_ID)
.value = $(
      'status_' +
TKR_currentTemplateIndex).valu
e;

$(TKR_PROMPT_OWNER_EDITOR_ID).
disabled = disabled;

$(TKR_PROMPT_OWNER_EDITOR_ID).
value = $(
      'owner_' +
TKR_currentTemplateIndex).valu
e;

$(TKR_OWNER_DEFAULTS_TO_MEMBER
_CHECKBOX_ID).disabled =
disabled;

$(TKR_OWNER_DEFAULTS_TO_MEMBER
_CHECKBOX_ID).checked = $(

'owner_defaults_to_member_' +
TKR_currentTemplateIndex).valu
e == 'yes';

$(TKR_COMPONENT_REQUIRED_CHECK
BOX_ID).disabled = disabled;

$(TKR_COMPONENT_REQUIRED_CHECK
BOX_ID).checked = $(
      'component_required_' +
TKR_currentTemplateIndex).valu
e == 'yes';

$(TKR_OWNER_DEFAULTS_TO_MEMBER
_AREA_ID).disabled = disabled;
```

```
$(TKR_OWNER_DEFAULTS_TO_MEMBER
_AREA_ID).style.display =

$(TKR_PROMPT_OWNER_EDITOR_ID).
value ? 'none' : '';

$(TKR_PROMPT_COMPONENTS_EDITOR
_ID).disabled = disabled;

$(TKR_PROMPT_COMPONENTS_EDITOR
_ID).value = $(
      'components_' +
TKR_currentTemplateIndex).valu
e;

  // Blank out all custom
field editors first, then fill
them in during the next loop.
  for (var i = 0; i <
TKR_fieldIDs.length; i++) {
    let fieldEditor =
$(TKR_FIELD_EDITOR_ID_PREFIX +
TKR_fieldIDs[i]);
    let holder =
$('field_value_' +
TKR_currentTemplateIndex + '_'
+ TKR_fieldIDs[i]);
    if (fieldEditor) {
      fieldEditor.disabled =
disabled;
      fieldEditor.value =
holder ? holder.value : '';
    }
  }

  var i = 0;
  while
($(TKR_PROMPT_LABELS_EDITOR_ID
_PREFIX + i)) {

$(TKR_PROMPT_LABELS_EDITOR_ID_
PREFIX + i).disabled =
disabled;

$(TKR_PROMPT_LABELS_EDITOR_ID_
PREFIX + i).value =
        $('label_' +
TKR_currentTemplateIndex + '_'
+ i).value;
    i++;
  }


$(TKR_PROMPT_ADMIN_NAMES_EDITO
R_ID).disabled = disabled;

$(TKR_PROMPT_ADMIN_NAMES_EDITO
R_ID).value = $(
      'admin_names_' +
TKR_currentTemplateIndex).valu
e;

  let numNonDeletedTemplates =
0;
  for (var i = 0; i <
TKR_templateNames.length; i++)
{
    if (TKR_templateNames[i]
!= TKR_DELETED_PROMPT_NAME) {

numNonDeletedTemplates++;
    }
  }
  if ($('delbtn')) {
    if (numNonDeletedTemplates
> 1) {
      $('delbtn').disabled='';
    } else { // Don't allow
the last template to be
deleted.

$('delbtn').disabled='disabled
';
    }
  }
}


var TKR_templateNames = []; //
Exported in tracker-onload.js


/**
 * Create a new issue template
```

```
and add the needed form fields
to the DOM.
 */
function TKR_newTemplate() {
  let newIndex =
TKR_templateNames.length;
  let templateName =
prompt('Name of new
template?', '');
  templateName =
templateName.replace(
      /[&<>"]/g, '', // " help
emacs highlighting
  );
  if (!templateName) return;

  for (let i = 0; i <
TKR_templateNames.length; i++)
{
    if (templateName ==
TKR_templateNames[i]) {
      alert('Please choose a
unique name.');
      return;
    }
  }


TKR_addTemplateHiddenFields(ne
wIndex, templateName);

TKR_templateNames.push(templat
eName);

  let templateOption =
TKR_createChild(
      $('template_menu'),
'option', null, null,
templateName);
  templateOption.value =
newIndex;
  templateOption.selected =
'selected';

  let developerOption =
TKR_createChild(
$('default_template_for_develo
pers'), 'option', null, null,
templateName);
  developerOption.value =
templateName;

  let userOption =
TKR_createChild(

$('default_template_for_users'
), 'option', null, null,
templateName);
  userOption.value =
templateName;


TKR_selectTemplate($('template
_menu'));
}


/**
 * Private function to append
HTML for new hidden form
fields
 * for a new issue template to
the issue admin form.
 */
function
TKR_addTemplateHiddenFields(te
mplateIndex, templateName) {
  let parentEl =
$('adminTemplates');
  TKR_appendHiddenField(
      parentEl, 'template_id_'
+ templateIndex,
'template_id_' +
templateIndex, '0');

TKR_appendHiddenField(parentEl
, 'name_' + templateIndex,
      'name_' + templateIndex,
templateName);

TKR_appendHiddenField(parentEl
, 'members_only_' +
templateIndex);
```

```
TKR_appendHiddenField(parentEl
, 'summary_' + templateIndex);

TKR_appendHiddenField(parentEl
, 'summary_must_be_edited_' +
templateIndex);

TKR_appendHiddenField(parentEl
, 'content_' + templateIndex);

TKR_appendHiddenField(parentEl
, 'status_' + templateIndex);

TKR_appendHiddenField(parentEl
, 'owner_' + templateIndex);
  TKR_appendHiddenField(
      parentEl,
'owner_defaults_to_member_' +
templateIndex,

'owner_defaults_to_member_' +
templateIndex, 'yes');

TKR_appendHiddenField(parentEl
, 'component_required_' +
templateIndex);

TKR_appendHiddenField(parentEl
, 'components_' +
templateIndex);

  var i = 0;
  while ($('label_0_' + i)) {

TKR_appendHiddenField(parentEl
, 'label_' + templateIndex,
        'label_' +
templateIndex + '_' + i);
    i++;
  }

  for (var i = 0; i <
TKR_fieldIDs.length; i++) {
    let fieldId =
'field_value_' + templateIndex
+ '_' + TKR_fieldIDs[i];
TKR_appendHiddenField(parentEl
, fieldId, fieldId);
  }


TKR_appendHiddenField(parentEl
, 'admin_names_' +
templateIndex);
  TKR_appendHiddenField(
      parentEl, 'can_edit_' +
templateIndex, 'can_edit_' +
templateIndex,
      'yes');
}


/**
 * Utility function to append
string parts for one hidden
field
 * to the given array.
 */
function
TKR_appendHiddenField(parentEl
, name, opt_id, opt_value) {
  let input =
TKR_createChild(parentEl,
'input', null, opt_id ||
name);
  input.setAttribute('type',
'hidden');
  input.name = name;
  input.value = opt_value ||
'';
}


/**
 * Delete the currently
selected issue template, and
mark its hidden
 * form field as deleted so
that they will be ignored when
submitted.
 */
function TKR_deleteTemplate()
```

```
{
  // Mark the current template
name as deleted.
  TKR_templateNames.splice(

TKR_currentTemplateIndex, 1,
TKR_DELETED_PROMPT_NAME);
  $('name_' +
TKR_currentTemplateIndex).valu
e = TKR_DELETED_PROMPT_NAME;

_toggleHidden($('edit_panel'))
;
  $('delbtn').disabled =
'disabled';
  TKR_rebuildTemplateMenu();

TKR_rebuildDefaultTemplateMenu
('default_template_for_develop
ers');

TKR_rebuildDefaultTemplateMenu
('default_template_for_users')
;
}

/**
 * Utility function to rebuild
the template menu on the issue
admin page.
 */
function
TKR_rebuildTemplateMenu() {
  let parentEl =
$('template_menu');
  while
(parentEl.childNodes.length) {

parentEl.removeChild(parentEl.
childNodes[0]);
  }
  for (let i = 0; i <
TKR_templateNames.length; i++)
{
    if (TKR_templateNames[i]
!= TKR_DELETED_PROMPT_NAME) {
      let option =
TKR_createChild(
          parentEl, 'option',
null, null,
TKR_templateNames[i]);
      option.value = i;
    }
  }
}


/**
 * Utility function to rebuild
a default template drop-down.
 */
function
TKR_rebuildDefaultTemplateMenu
(menuID) {
  let defaultTemplateName =
$(menuID).value;
  let parentEl = $(menuID);
  while
(parentEl.childNodes.length) {

parentEl.removeChild(parentEl.
childNodes[0]);
  }
  for (let i = 0; i <
TKR_templateNames.length; i++)
{
    if (TKR_templateNames[i]
!= TKR_DELETED_PROMPT_NAME) {
      let option =
TKR_createChild(
          parentEl, 'option',
null, null,
TKR_templateNames[i]);
      option.values =
TKR_templateNames[i];
      if (defaultTemplateName
== TKR_templateNames[i]) {

option.setAttribute('selected'
, 'selected');
      }
    }
  }
}
```

```
/**
 * Change the issue template
to the specified one.
 * TODO(jrobbins): move to an
AJAX implementation that would
not reload page.
 *
 * @param {string} projectName
The name of the current
project.
 * @param {string}
templateName The name of the
template to switch to.
 */
function
TKR_switchTemplate(projectName
, templateName) {
  let ok = true;
  if (TKR_isDirty()) {
    ok = confirm('Switching to
a different template will lose
the text you entered.');
  }
  if (ok) {
    TKR_initialFormValues =
TKR_currentFormValues();
    window.location = '/p/' +
projectName +
      '/issues/entry?
template=' + templateName;
  }
}

/**
 * Function to remove a CSS
class and initial tip from a
text widget.
 * Some text fields or text
areas display gray textual
tips to help the user
 * make use of those widgets.
When the user focuses on the
field, the tip
 * disappears and is made
ready for user input (in the
normal text color).
 * @param {Element} el The
form field that had the gray
text tip.
 */
function TKR_makeDefined(el) {
  if
(el.classList.contains(TKR_UND
EF_CLASS)) {

el.classList.remove(TKR_UNDEF_
CLASS);
    el.value = '';
  }
}


/**
 * Save the contents of the
visible issue template text
area into a hidden
 * text field for later
submission.
 * Called when the user has
edited the text of a issue
template.
 */
function TKR_saveTemplate() {
  if
(TKR_currentTemplateIndex) {
    $('members_only_' +
TKR_currentTemplateIndex).valu
e =

$(TKR_PROMPT_MEMBERS_ONLY_CHEC
KBOX_ID).checked ? 'yes' : '';
    $('summary_' +
TKR_currentTemplateIndex).valu
e =

$(TKR_PROMPT_SUMMARY_EDITOR_ID
).value;

$('summary_must_be_edited_' +
TKR_currentTemplateIndex).valu
e =
$(TKR_PROMPT_SUMMARY_MUST_BE_E
DITED_CHECKBOX_ID).checked ?
'yes' : '';
    $('content_' +
TKR_currentTemplateIndex).valu
e =

$(TKR_PROMPT_CONTENT_EDITOR_ID
).value;
    $('status_' +
TKR_currentTemplateIndex).valu
e =

$(TKR_PROMPT_STATUS_EDITOR_ID)
.value;
    $('owner_' +
TKR_currentTemplateIndex).valu
e =

$(TKR_PROMPT_OWNER_EDITOR_ID).
value;

$('owner_defaults_to_member_'
+
TKR_currentTemplateIndex).valu
e =

$(TKR_OWNER_DEFAULTS_TO_MEMBER
_CHECKBOX_ID).checked ? 'yes'
: '';
    $('component_required_' +
TKR_currentTemplateIndex).valu
e =

$(TKR_COMPONENT_REQUIRED_CHECK
BOX_ID).checked ? 'yes' : '';
    $('components_' +
TKR_currentTemplateIndex).valu
e =

$(TKR_PROMPT_COMPONENTS_EDITOR
_ID).value;

$(TKR_OWNER_DEFAULTS_TO_MEMBER
_AREA_ID).style.display =

$(TKR_PROMPT_OWNER_EDITOR_ID).
```

```
value ? 'none' : '';

    for (var i = 0; i <
TKR_fieldIDs.length; i++) {
      let fieldID =
TKR_fieldIDs[i];
      let fieldEditor =
$(TKR_FIELD_EDITOR_ID_PREFIX +
fieldID);
      if (fieldEditor) {

_saveFieldValue(fieldID,
fieldEditor.value);
      }
    }

    var i = 0;
    while ($('label_' +
TKR_currentTemplateIndex + '_'
+ i)) {
      $('label_' +
TKR_currentTemplateIndex + '_'
+ i).value =

$(TKR_PROMPT_LABELS_EDITOR_ID_
PREFIX + i).value;
      i++;
    }

    $('admin_names_' +
TKR_currentTemplateIndex).valu
e =

$(TKR_PROMPT_ADMIN_NAMES_EDITO
R_ID).value;
  }
}


function
_saveFieldValue(fieldID, val)
{
  let fieldValId =
'field_value_' +
TKR_currentTemplateIndex + '_'
+ fieldID;
  $(fieldValId).value = val;
```

```
}


/**
 * This is a json string
encoding of an array of form
values after the initial
 * page load. It is used for
comparison on page unload to
prompt the user
 * before abandoning changes.
It is initialized in
TKR_onload().
*/
let TKR_initialFormValues;


/**
 * Returns a json string
encoding of an array of all
the values from user
 * input fields of interest
(omits search box, e.g.)
 */
function
TKR_currentFormValues() {
  let inputs =
document.querySelectorAll('inp
ut, textarea, select,
checkbox');
  let values = [];

  for (i = 0; i <
inputs.length; i++) {
    // Don't include blank
inputs. This prevents a popup
if the user
    // clicks "add a row" for
new labels but doesn't
actually enter any
    // text into them. Also
ignore search box contents.
    if (inputs[i].value &&
!inputs[i].hasAttribute('ignor
e-dirty') &&
        inputs[i].name !=
'token') {
```

```
values.push(inputs[i].value);
    }
  }

  return
JSON.stringify(values);
}


/**
 * This function returns true
if the user has made any edits
to fields of
 * interest.
 */
function TKR_isDirty() {
  return TKR_initialFormValues
!= TKR_currentFormValues();
}


/**
 * The user has clicked the
'Discard' button on the issue
update form.
 * If the form has been
edited, ask if they are sure
about discarding
 * before then navigating to
the given URL.  This can go up
to some
 * other page, or reload the
current page with a fresh
form.
 * @param {string} nextUrl The
page to show after discarding.
 */
function
TKR_confirmDiscardUpdate(nextU
rl) {
  if (!TKR_isDirty() ||
confirm(TKR_DISCARD_YOUR_CHANG
ES)) {
    document.location =
nextUrl;
  }
```

```
}


/**
 * The user has clicked the
'Discard' button on the issue
entry form.
 * If the form has been
edited, this function asks if
they are sure about
 * discarding before doing it.
 * @param {Element}
discardButton The 'Discard'
button.
 */
function
TKR_confirmDiscardEntry(discar
dButton) {
  if (!TKR_isDirty() ||
confirm(TKR_DISCARD_YOUR_CHANG
ES)) {
    TKR_go('list');
  }
}


/**
 * Normally, we show 2 rows of
label editing fields when
updating an issue.
 * However, if the issue has
more than that many labels
already, we make sure to
 * show them all.
 */
function
TKR_exposeExistingLabelFields(
) {
  if ($('label3').value ||
      $('label4').value ||
      $('label5').value) {
    if ($('addrow1')) {
      _showID('LF_row2');
      _hideID('addrow1');
    }
  }
  if ($('label6').value ||
      $('label7').value ||
      $('label8').value) {
    _showID('LF_row3');
    _hideID('addrow2');
  }
  if ($('label9').value ||
      $('label10').value ||
      $('label11').value) {
    _showID('LF_row4');
    _hideID('addrow3');
  }
  if ($('label12').value ||
      $('label13').value ||
      $('label14').value) {
    _showID('LF_row5');
    _hideID('addrow4');
  }
  if ($('label15').value ||
      $('label16').value ||
      $('label17').value) {
    _showID('LF_row6');
    _hideID('addrow5');
  }
  if ($('label18').value ||
      $('label19').value ||
      $('label20').value) {
    _showID('LF_row7');
    _hideID('addrow6');
  }
  if ($('label21').value ||
      $('label22').value ||
      $('label23').value) {
    _showID('LF_row8');
    _hideID('addrow7');
  }
}


/**
 * Flag to indicate when the
user has not yet caused any
input events.
 * We use this to clear the
placeholder in the new issue
summary field
 * exactly once.
 */
```

```
let TKR_firstEvent = true;


/**
 * This is called in response
to almost any user input event
on the
 * issue entry page.  If the
placeholder in the new issue
sumary field has
 * not yet been cleared, then
this function clears it.
 */
function
TKR_clearOnFirstEvent(initialS
ummary) {
  if (TKR_firstEvent &&
$('summary').value ==
initialSummary) {
    TKR_firstEvent = false;
    $('summary').value =
TKR_keepJustSummaryPrefixes($(
'summary').value);
  }
}

/**
 * Clear the summary, except
for any prefixes of the form "
[bracketed text]"
 * or "keyword:".  If there
were any, add a trailing
space.  This is useful
 * to people who like to
encode issue classification
info in the summary line.
 */
function
TKR_keepJustSummaryPrefixes(s)
{
  let matches = s.match(/^(\
[[^\]]+\])+|^(\S+:\s*)+/);
  if (matches == null) {
    return '';
  }

  let prefix = matches[0];
  if
(prefix.substr(prefix.length -
1) != ' ') {
    prefix += ' ';
  }
  return prefix;
}

/**
 * An array of label <input>s
that start with reserved
prefixes.
 */
let
TKR_labelsWithReservedPrefixes
= [];

/**
 * An array of label <input>s
that are equal to reserved
words.
 */
let
TKR_labelsConflictingWithReser
ved = [];

/**
 * An array of novel issue
status values entered by the
user on the
 * current page. 'Novel' means
that they are not well known
and are
 * likely to be typos.  Note
that this list will always
have zero or
 * one element, but a list is
used for consistency with the
list of
 * novel labels.
 */
let TKR_novelStatuses = [];

/**
 * An array of novel issue
label values entered by the
user on the
```

```
 * current page. 'Novel' means
that they are not well known
and are
 * likely to be typos.
 */
let TKR_novelLabels = [];

/**
 * A boolean that indicates
whether the entered owner
value is valid or not.
 */
let TKR_invalidOwner = false;

/**
 * The user has changed the
issue status text field.  This
function
 * checks whether it is a
well-known status value.  If
not, highlight it
 * as a potential typo.
 * @param {Element} textField
The issue status text field.
 * @return Always returns true
to indicate that the browser
should
 * continue to process the
user input event normally.
 */
function
TKR_confirmNovelStatus(textFie
ld) {
  let v =
textField.value.trim().toLower
Case();
  let isNovel = (v !== '');
  let wellKnown =
TKR_statusWords;
  for (let i = 0; i <
wellKnown.length && isNovel;
++i) {
    let wk = wellKnown[i];
    if (v == wk.toLowerCase())
{
      isNovel = false;
    }
  }
  if (isNovel) {
    if
(TKR_novelStatuses.indexOf(tex
tField) == -1) {

TKR_novelStatuses.push(textFie
ld);
    }

textField.classList.add(TKR_NO
VEL_CLASS);
  } else {
    if
(TKR_novelStatuses.indexOf(tex
tField) != -1) {

TKR_novelStatuses.splice(TKR_n
ovelStatuses.indexOf(textField
), 1);
    }

textField.classList.remove(TKR
_NOVEL_CLASS);
  }

TKR_updateConfirmBeforeSubmit(
);
  return true;
}


/**
 * The user has changed a
issue label text field.  This
function checks
 * whether it is a well-known
label value.  If not,
highlight it as a
 * potential typo.
 * @param {Element} textField
An issue label text field.
 * @return Always returns true
to indicate that the browser
should
 * continue to process the
user input event normally.
```

```
 *
 * TODO(jrobbins): code
duplication with function
above.
 */
function
TKR_confirmNovelLabel(textFiel
d) {
  let v =
textField.value.trim().toLower
Case();
  if (v.search('-') == 0) {
    v = v.substr(1);
  }
  let isNovel = (v !== '');
  if (v.indexOf('?') > -1) {
    isNovel = false; // We
don't count labels that the
user must edit anyway.
  }
  let wellKnown =
TKR_labelWords;
  for (var i = 0; i <
wellKnown.length && isNovel;
++i) {
    let wk = wellKnown[i];
    if (v == wk.toLowerCase())
{
      isNovel = false;
    }
  }

  let containsReservedPrefix =
false;
  var
textFieldWarningDisplayed =
TKR_labelsWithReservedPrefixes
.indexOf(textField) != -1;
  for (var i = 0; i <
TKR_LABEL_RESERVED_PREFIXES.le
ngth; ++i) {
    if
(v.startsWith(TKR_LABEL_RESERV
ED_PREFIXES[i] + '-')) {
      if
(!textFieldWarningDisplayed) {
```

```
TKR_labelsWithReservedPrefixes
.push(textField);
      }
      containsReservedPrefix =
true;
      break;
    }
  }
  if (!containsReservedPrefix
&& textFieldWarningDisplayed)
{

TKR_labelsWithReservedPrefixes
.splice(

TKR_labelsWithReservedPrefixes
.indexOf(textField), 1);
  }

  let conflictsWithReserved =
false;
  var
textFieldWarningDisplayed =

TKR_labelsConflictingWithReser
ved.indexOf(textField) != -1;
  for (var i = 0; i <
TKR_LABEL_RESERVED_PREFIXES.le
ngth; ++i) {
    if (v ==
TKR_LABEL_RESERVED_PREFIXES[i]
) {
      if
(!textFieldWarningDisplayed) {

TKR_labelsConflictingWithReser
ved.push(textField);
      }
      conflictsWithReserved =
true;
      break;
    }
  }
  if (!conflictsWithReserved
&& textFieldWarningDisplayed)
{
```

```
TKR_labelsConflictingWithReser
ved.splice(

TKR_labelsConflictingWithReser
ved.indexOf(textField), 1);
  }

  if (isNovel) {
    if
(TKR_novelLabels.indexOf(textF
ield) == -1) {

TKR_novelLabels.push(textField
);
    }

textField.classList.add(TKR_NO
VEL_CLASS);
  } else {
    if
(TKR_novelLabels.indexOf(textF
ield) != -1) {

TKR_novelLabels.splice(TKR_nov
elLabels.indexOf(textField),
1);
    }

textField.classList.remove(TKR
_NOVEL_CLASS);
  }

TKR_updateConfirmBeforeSubmit(
);
  return true;
}

/**
 * Dictionary { prefix:
[textField,...], ...} for all
the prefixes of any
 * text that has been entered
into any label field.  This is
used to find
 * duplicate labels and
multiple labels that share an
single exclusive
```

```
 * prefix (e.g., Priority).
 */
let TKR_usedPrefixes = {};

/**
 * This is a prefix to the
HTML ids of each label editing
field.
 * It varied by page, so it is
set in the HTML page.  Needed
to initialize
 * our validation across label
input text fields.
 */
let TKR_labelFieldIDPrefix =
'';

/**
 * Initialize the set of all
used labels on forms that
allow users to
 * enter issue labels.  Some
labels are supplied in the
HTML page
 * itself, and we do not want
to offer duplicates of those.
 */
function TKR_prepLabelAC() {
  let i = 0;
  while ($('label'+i)) {

TKR_validateLabel($('label'+i)
);
    i++;
  }
}

/**
 * Reads the owner field and
determines if the current
value is a valid member.
 */
function
TKR_prepOwnerField(validOwners
) {
  if ($('owneredit')) {
    currentOwner =
$('owneredit').value;
    if (currentOwner == '') {
      // Empty owner field is
not an invalid owner.
      invalidOwner = false;
      return;
    }
    invalidOwner = true;
    for (let i = 0; i <
validOwners.length; i++) {
      let owner =
validOwners[i].name;
      if (currentOwner ==
owner) {
        invalidOwner = false;
        break;
      }
    }
    TKR_invalidOwner =
invalidOwner;
  }
}

/**
 * Keep track of which label
prefixes have been used so
that
 * we can not offer the same
label twice and so that we can
highlight
 * multiple labels that share
an exclusive prefix.
 */
function
TKR_updateUsedPrefixes(textFie
ld) {
  if (textField.oldPrefix !=
undefined) {

DeleteArrayElement(TKR_usedPre
fixes[textField.oldPrefix],
textField);
  }

  let prefix =
textField.value.split('-')
[0].toLowerCase();
```

```
  if (TKR_usedPrefixes[prefix]
== undefined) {
    TKR_usedPrefixes[prefix] =
[textField];
  } else {

TKR_usedPrefixes[prefix].push(
textField);
  }
  textField.oldPrefix =
prefix;
}

/**
 * Go through all the label
entry fields in our prefix-
oriented
 * data structure and
highlight any that are part of
a conflict
 * (multiple labels with the
same exclusive prefix).
Unhighlight
 * any label text entry fields
that are not in conflict.
And, display
 * a warning message to
encourage the user to correct
the conflict.
 */
function
TKR_highlightExclusiveLabelPre
fixConflicts() {
  let conflicts = [];
  for (let prefix in
TKR_usedPrefixes) {
    let textFields =
TKR_usedPrefixes[prefix];
    if (textFields ==
undefined || textFields.length
== 0) {
      delete
TKR_usedPrefixes[prefix];
    } else if
(textFields.length > 1 &&

FindInArray(TKR_exclPrefixes,
prefix) != -1) {
      conflicts.push(prefix);
      for (var i = 0; i <
textFields.length; i++) {
        var tf =
textFields[i];

tf.classList.add(TKR_EXCL_CONF
ICT_CLASS);
      }
    } else {
      for (var i = 0; i <
textFields.length; i++) {
        var tf =
textFields[i];

tf.classList.remove(TKR_EXCL_C
ONFICT_CLASS);
      }
    }
  }
  if (conflicts.length > 0) {
    let severity =
TKR_restrict_to_known ?
'Error' : 'Warning';
    let confirm_area =
$(TKR_CONFIRMAREA_ID);
    if (confirm_area) {

$('confirmmsg').textContent =
(severity +
          ': Multiple values
for: ' + conflicts.join(',
'));
      confirm_area.className =
TKR_EXCL_CONFICT_CLASS;

confirm_area.style.display =
'';
    }
  }
}

/**
 * Keeps track of any label
text fields that have a value
that
```

```
 * is bad enough to prevent
submission of the form.  When
this
 * list is non-empty, the
submit button gets disabled.
 */
let TKR_labelsBlockingSubmit =
[];

/**
 * Look for any "?" characters
in the label and, if found,
 * make the label text red,
prevent form submission, and
 * display on-page help to
tell the user to edit those
labels.
 * @param {Element} textField
An issue label text field.
 */
function
TKR_highlightQuestionMarks(tex
tField) {
  let tfIndex =
TKR_labelsBlockingSubmit.index
Of(textField);
  if
(textField.value.indexOf('?')
> -1 && tfIndex == -1) {

TKR_labelsBlockingSubmit.push(
textField);

textField.classList.add(TKR_QU
ESTION_MARK_CLASS);
  } else if
(textField.value.indexOf('?')
== -1 && tfIndex > -1) {

TKR_labelsBlockingSubmit.splic
e(tfIndex, 1);

textField.classList.remove(TKR
_QUESTION_MARK_CLASS);
  }

  let block_submit_msg =
$('blocksubmitmsg');
  if (block_submit_msg) {
    if
(TKR_labelsBlockingSubmit.leng
th > 0) {

block_submit_msg.textContent =
'You must edit labels that
contain "?".';
    } else {

block_submit_msg.textContent =
'';
    }
  }
}

/**
 * The user has edited a
label.  Display a warning if
the label is
 * not a well known label, or
if there are multiple labels
that
 * share an exclusive prefix.
 * @param {Element} textField
An issue label text field.
 */
function
TKR_validateLabel(textField) {
  if (textField == undefined)
return;

TKR_confirmNovelLabel(textFiel
d);

TKR_updateUsedPrefixes(textFie
ld);

TKR_highlightExclusiveLabelPre
fixConflicts();

TKR_highlightQuestionMarks(tex
tField);
}

// TODO(jrobbins): what about
```

```
typos in owner and cc list?

/**
 * If there are any novel
status or label values, we
display a message
 * that explains that to the
user so that they can catch
any typos before
 * submitting them.  If the
project is restricting input
to only the
 * well-known statuses and
labels, then show these as an
error instead.
 * In that case, on-page JS
will prevent submission.
 */
function
TKR_updateConfirmBeforeSubmit(
) {
  let severity =
TKR_restrict_to_known ?
'Error' : 'Note';
  let novelWord =
TKR_restrict_to_known ?
'undefined' : 'uncommon';
  let msg = '';
  let labels =
TKR_novelLabels.map(function(i
tem) {
    return item.value;
  });
  if (TKR_novelStatuses.length
> 0 && TKR_novelLabels.length
> 0) {
    msg = severity + ': You
are using an ' + novelWord + '
status and ' + novelWord + '
label(s): ' + labels.join(',
') + '.'; // TODO: i18n
  } else if
(TKR_novelStatuses.length > 0)
{
    msg = severity + ': You
are using an ' + novelWord + '
status value.';
```

```
  } else if
(TKR_novelLabels.length > 0) {
    msg = severity + ': You
are using ' + novelWord + '
label(s): ' + labels.join(',
') + '.';
  }

  for (var i = 0; i <
TKR_labelsWithReservedPrefixes
.length; ++i) {
    msg += '\nNote: The label
' +
TKR_labelsWithReservedPrefixes
[i].value +
           ' starts with a
reserved word. This is not
recommended.';
  }
  for (var i = 0; i <
TKR_labelsConflictingWithReser
ved.length; ++i) {
    msg += '\nNote: The label
' +
TKR_labelsConflictingWithReser
ved[i].value +
           ' conflicts with a
reserved word. This is not
recommended.';
  }
  // Display the owner is no
longer a member note only if
an owner error is not
  // already shown on the
page.
  if (TKR_invalidOwner &&
!$('ownererror')) {
    msg += '\nNote: Current
owner is no longer a project
member.';
  }

  let confirm_area =
$(TKR_CONFIRMAREA_ID);
  if (confirm_area) {

$('confirmmsg').textContent =
msg;
    if (msg != '') {
      confirm_area.className =
TKR_NOVEL_CLASS;

confirm_area.style.display =
'';
    } else {

confirm_area.style.display =
'none';
    }
  }
}


/**
 * The user has selected a
command from the 'Actions...'
menu
 * on the issue list.  This
function checks the selected
value and carry
 * out the requested action.
 * @param {Element}
actionsMenu The 'Actions...'
<select> form element.
 */
function
TKR_handleListActions(actionsM
enu) {
  switch (actionsMenu.value) {
    case 'bulk':
      TKR_HandleBulkEdit();
      break;
    case 'colspec':

TKR_closeAllPopups(actionsMenu
);
      _showID('columnspec');

_hideID('addissuesspec');
      break;
    case 'flagspam':
      TKR_flagSpam(true);
      break;
    case 'unflagspam':
      TKR_flagSpam(false);
      break;
    case 'addtohotlist':
      TKR_addToHotlist();
      break;
    case 'addissues':

_showID('addissuesspec');
      _hideID('columnspec');
      setCurrentColSpec();
      break;
    case 'removeissues':
      HTL_removeIssues();
      break;
    case 'issuesperpage':
      break;
  }
  actionsMenu.value =
'moreactions';
}


async function
TKR_handleDetailActions(localI
d) {
  let moreActions =
$('more_actions');

  if (moreActions.value ==
'delete') {

$('copy_issue_form_fragment').
style.display = 'none';

$('move_issue_form_fragment').
style.display = 'none';
    let ok = confirm(
        'Normally, you should
just close issues by setting
their status ' +
      'to a closed value.\n' +
      'Are you sure you want
to delete this issue?');
    if (ok) {
      await
window.prpcClient.call('monora
il.Issues', 'DeleteIssue', {
        issueRef: {
          projectName:
window.CS_env.projectName,
          localId: localId,
        },
        delete: true,
      });
      location.reload(true);
      return;
    }
  }

  if (moreActions.value ==
'move') {

$('move_issue_form_fragment').
style.display = '';

$('copy_issue_form_fragment').
style.display = 'none';
    return;
  }
  if (moreActions.value ==
'copy') {

$('copy_issue_form_fragment').
style.display = '';

$('move_issue_form_fragment').
style.display = 'none';
    return;
  }

  // If no action was taken,
reset the dropdown to the
'More actions...' item.
  moreActions.value = '0';
}

/**
 * The user has selected the
"Flag as spam..." menu item.
 */
async function
TKR_flagSpam(isSpam) {
  const selectedIssueRefs =
[];
```

```
  issueRefs.forEach((issueRef)
=> {
    const checkbox = $('cb_' +
issueRef.id);
    if (checkbox &&
checkbox.checked) {
      selectedIssueRefs.push({
        projectName:
issueRef.project_name,
        localId: issueRef.id,
      });
    }
  });
  if (selectedIssueRefs.length
> 0) {
    if (!confirm((isSpam ?
'Flag' : 'Un-flag') +
        ' all selected issues
as spam?')) {
      return;
    }
    await
window.prpcClient.call('monora
il.Issues', 'FlagIssues', {
      issueRefs:
selectedIssueRefs,
      flag: isSpam,
    });
    location.reload(true);
  } else {
    alert('Please select some
issues to flag as spam');
  }
}

function TKR_addToHotlist() {
  const selectedIssueRefs =
GetSelectedIssuesRefs();
  if (selectedIssueRefs.length
> 0) {

window.__hotlists_dialog.ShowU
pdateHotlistDialog();
  } else {
    alert('Please select some
issues to add to a hotlist');
  }
```

```
}


function
GetSelectedIssuesRefs() {
  let selectedIssueRefs = [];
  for (let i = 0; i <
issueRefs.length; i++) {
    let checkbox =
document.getElementById('cb_'
+ issueRefs[i]['id']);
    if (checkbox == null) {
      checkbox =
document.getElementById(
          'cb_' + issueRefs[i]
['project_name'] + ':' +
issueRefs[i]['id']);
    }
    if (checkbox &&
checkbox.checked) {

selectedIssueRefs.push(issueRe
fs[i]);
    }
  }
  return selectedIssueRefs;
}

function
onResponseUpdateUI(modifiedHot
lists, remainingHotlists) {
  const list = $('user-
hotlists-list');
  while (list.firstChild) {

list.removeChild(list.firstChi
ld);
  }

remainingHotlists.forEach((hot
list) => {
    const name = hotlist[0];
    const userId = hotlist[1];
    const url =
`/u/${userId}/hotlists/${name}
`;
    const hotlistLink =
document.createElement('a');

hotlistLink.setAttribute('href
', url);
    hotlistLink.textContent =
name;

list.appendChild(hotlistLink);

list.appendChild(document.crea
teElement('br'));
  });
  $('user-
hotlists').style.display =
'block';

onAddIssuesResponse(modifiedHo
tlists);
}

function
onAddIssuesResponse(modifiedHo
tlists) {
  const hotlistNames =
modifiedHotlists.map((hotlist)
=> hotlist[0]).join(', ');
  $('notice').textContent =
'Successfully updated ' +
hotlistNames;
  $('update-issues-
hotlists').style.display =
'none';
  $('alert-
table').style.display =
'table';
}

function
onAddIssuesFailure(reason) {
  $('notice').textContent =
      'Some hotlists were not
updated: ' +
reason.description;
  $('update-issues-
hotlists').style.display =
'none';
  $('alert-
table').style.display =
'table';
}

/**
 * The user has selected the
"Bulk Edit..." menu item.  Go
to a page that
 * offers the ability to edit
all selected issues.
 */
// TODO(jrobbins): cross-
project bulk edit
function TKR_HandleBulkEdit()
{
  let selectedIssueRefs =
GetSelectedIssuesRefs();
  let selectedLocalIDs = [];
  for (let i = 0; i <
selectedIssueRefs.length; i++)
{

selectedLocalIDs.push(selected
IssueRefs[i]['id']);
  }
  if (selectedLocalIDs.length
> 0) {
    let selectedLocalIDString
= selectedLocalIDs.join(',');
    let url = 'bulkedit?ids='
+ selectedLocalIDString;
    TKR_go(url + _ctxArgs);
  } else {
    alert('Please select some
issues to edit');
  }
}

/**
 * Clears the selected status
value when the 'clear'
operator is chosen.
 */
function
TKR_ignoreWidgetIfOpIsClear(se
lectEl, inputID) {
  if (selectEl.value ==
'clear') {

document.getElementById(inputI
D).value = '';
  }
}

/**
 * Array of original labels on
the served page, so that we
can notice
 * when the used submits a
form that has any Restrict-*
labels removed.
 */
let TKR_allOrigLabels = [];


/**
 * Prevent users from easily
entering "+1" comments.
 */
function TKR_checkPlusOne() {
  let c =
$('addCommentTextArea').value;
  let instructions = (
    '\nPlease use the star
icon instead.\n' +
      'Stars show your
interest without annoying
other users.');
  if (new RegExp('^\\s*[-+]+
[0-9]+\\s*.{0,30}$',
'm').test(c) &&
      c.length < 150) {
    alert('This looks like a
"+1" comment.' +
instructions);
    return false;
  }
  if (new RegExp('^\\s*me too.
{0,30}$', 'i').test(c)) {
    alert('This looks like a
"me too" comment.' +
instructions);
    return false;
  }
  return true;
}


/**
 * If the user removes
Restrict-* labels, ask them if
they are sure.
 */
function
TKR_checkUnrestrict(prevent_re
striction_removal) {
  let removedRestrictions =
[];

  for (let i = 0; i <
TKR_allOrigLabels.length; ++i)
{
    let origLabel =
TKR_allOrigLabels[i];
    if
(origLabel.indexOf('Restrict-
') == 0) {
      let found = false;
      let j = 0;
      while ($('label' + j)) {
        let newLabel =
$('label' + j).value;
        if (newLabel ==
origLabel) {
          found = true;
          break;
        }
        j++;
      }
      if (!found) {

removedRestrictions.push(origL
abel);
      }
    }
  }

  if
(removedRestrictions.length ==
0) {
    return true;
  }

  if
(prevent_restriction_removal)
{
    let msg = 'You may not
remove restriction labels.';
    alert(msg);
    return false;
  }

  let instructions = (
    'You are removing these
restrictions:\n   ' +

removedRestrictions.join('\n
') +
      '\nThis may allow more
people to access this issue.'
+
      '\nAre you sure?');
  return
confirm(instructions);
}


/**
 * Add a column to a list view
by updating the colspec form
element and
 * submiting an invisible
<form> to load a new page that
includes the column.
 * @param {string} colname The
name of the column to start
showing.
 */
function
TKR_addColumn(colname) {
  let colspec =
TKR_getColspecElement();
  colspec.value =
colspec.value + ' ' + colname;
  $('colspecform').submit();
}
```

```
/**
 * Allow members to shift-
click to select multiple
issues.  This keeps
 * track of the last row that
the user clicked a checkbox
on.
 */
let TKR_lastSelectedRow =
undefined;


/**
 * Return true if an event had
the shift-key pressed.
 * @param {Event} evt The
mouse click event.
 */
function TKR_hasShiftKey(evt)
{
  evt = (evt) ? evt :
(window.event) ? window.event
: '';
  if (evt) {
    if (evt.modifiers) {
      return evt.modifiers &
Event.SHIFT_MASK;
    } else {
      return evt.shiftKey;
    }
  }
  return false;
}


/**
 * Select one row: check the
checkbox and use highlight
color.
 * @param {Element} row the
row containing the checkbox
that the user clicked.
 * @param {boolean} checked
True if the user checked the
box.
 */
function
```

```
TKR_rangeSelectRow(row,
checked) {
  if (!row) {
    return;
  }
  if (checked) {

row.classList.add('selected');
  } else {

row.classList.remove('selected
');
  }

  let td = row.firstChild;
  while (td && td.tagName !=
'TD') {
    td = td.nextSibling;
  }
  if (!td) {
    return;
  }

  let checkbox =
td.firstChild;
  while (checkbox &&
checkbox.tagName != 'INPUT') {
    checkbox =
checkbox.nextSibling;
  }
  if (!checkbox) {
    return;
  }

  checkbox.checked = checked;
}


/**
 * If the user shift-clicked a
checkbox, (un)select a range.
 * @param {Event} evt The
mouse click event.
 * @param {Element} el The
checkbox that was clicked.
 */
function
```

```
TKR_checkRangeSelect(evt, el)
{
  let clicked_row =
el.parentNode.parentNode.rowIn
dex;
  if (clicked_row ==
TKR_lastSelectedRow) {
    return;
  }
  if (TKR_hasShiftKey(evt) &&
TKR_lastSelectedRow !=
undefined) {
    let results_table =
$('resultstable');
    let delta = (clicked_row >
TKR_lastSelectedRow) ? 1 : -1;
    for (let i =
TKR_lastSelectedRow; i !=
clicked_row; i += delta) {

TKR_rangeSelectRow(results_tab
le.rows[i], el.checked);
    }
  }
  TKR_lastSelectedRow =
clicked_row;
}


/**
 * Make a link to a given
issue that includes context
parameters that allow
 * the user to see the same
list columns, sorting, query,
and pagination state
 * if they ever navigate up to
the list again.
 * @param {{issue_url:
string}} issueRef The dict
with info about an issue,
 *     including a url to the
issue detail page.
 */
function
TKR_makeIssueLink(issueRef) {
  return '/p/' +
issueRef['project_name'] +
'/issues/detail?id=' +
issueRef['id'] + _ctxArgs;
}


/**
 * Hide or show a list column
in the case where we already
have the
 * data for that column on the
page.
 * @param {number} colIndex
index of the column that is
being shown or hidden.
 */
function
TKR_toggleColumnUpdate(colInde
x) {
  let shownCols =
TKR_getColspecElement().value.
split(' ');
  let filteredCols = [];
  for (let i=0; i<
shownCols.length; i++) {
    if
(_allColumnNames[colIndex] !=
shownCols[i].toLowerCase()) {

filteredCols.push(shownCols[i]
);
    }
  }


TKR_getColspecElement().value
= filteredCols.join(' ');
  TKR_toggleColumn('hide_col_'
+ colIndex);
  _ctxArgs =
_formatContextQueryArgs();

window.history.replaceState({}
, '', '?' + _ctxArgs);
}
```

```
/**
 * Convert a column into a
groupby clause by removing it
from the column spec
 * and adding it to the
groupby spec, then reloading
the page.
 * @param {number} colIndex
index of the column that is
being shown or hidden.
 */
function
TKR_addGroupBy(colIndex) {
  let colName =
_allColumnNames[colIndex];
  let shownCols =
TKR_getColspecElement().value.
split(' ');
  let filteredCols = [];
  for (var i=0; i <
shownCols.length; i++) {
    if (shownCols[i] &&
colName !=
shownCols[i].toLowerCase()) {

filteredCols.push(shownCols[i]
);
    }
  }


TKR_getColspecElement().value
= filteredCols.join(' ');

  let groupSpec =
$('groupbyspec');
  let shownGroupings =
groupSpec.value.split(' ');
  let filteredGroupings = [];
  for (i=0; i <
shownGroupings.length; i++) {
    if (shownGroupings[i] &&
colName !=
shownGroupings[i].toLowerCase(
)) {

filteredGroupings.push(shownGr
oupings[i]);
    }
  }

filteredGroupings.push(colName
);
  groupSpec.value =
filteredGroupings.join(' ');
  $('colspecform').submit();
}


/**
 * Add a multi-valued custom
field editing widget.
 */
function
TKR_addMultiFieldValueWidget(
    el, field_id, field_type,
opt_validate_1,
opt_validate_2,
field_phase_name) {
  let widget =
document.createElement('INPUT'
);
  widget.name =
(field_phase_name && (
    field_phase_name != '')) ?
`custom_${field_id}_${field_ph
ase_name}` :
    `custom_${field_id}`;
  if (field_type == 'str' ||
field_type =='url') {
    widget.size = 90;
  }
  if (field_type == 'user') {
    widget.style =
'width:12em';

widget.classList.add('userauto
complete');

widget.classList.add('customfi
eld');

widget.classList.add('multival
ued');
```

```
widget.addEventListener('focus
', function(event) {
      _acrob(null);
      _acof(event);
    });
  }
  if (field_type == 'int' ||
field_type == 'date') {
    widget.style.textAlign =
'right';
    widget.style.width =
'12em';
    widget.min =
opt_validate_1;
    widget.max =
opt_validate_2;
  }
  if (field_type == 'int') {
    widget.type = 'number';
  } else if (field_type ==
'date') {
    widget.type = 'date';
  }


el.parentNode.insertBefore(wid
get, el);

  let del_button =
document.createElement('U');
  del_button.onclick =
function(event) {

_removeMultiFieldValueWidget(e
vent.target);
  };
  del_button.textContent =
'X';

el.parentNode.insertBefore(del
_button, el);
}


function
TKR_removeMultiFieldValueWidge
```

```
t(el) {
  let target =
el.previousSibling;
  while (target &&
target.tagName != 'INPUT') {
    target =
target.previousSibling;
  }
  if (target) {

el.parentNode.removeChild(targ
et);
  }

el.parentNode.removeChild(el);
// the X itself
}


/**
 * Trim trailing commas and
spaces off <INPUT type="email"
multiple> fields
 * before submitting the form.
 */
function TKR_trimCommas() {
  let ccField =
$('memberccedit');
  if (ccField) {
    ccField.value =
ccField.value.replace(/,\s*$/,
'');
  }
  ccField = $('memberenter');
  if (ccField) {
    ccField.value =
ccField.value.replace(/,\s*$/,
'');
  }
}


/**
 * Identify which issues have
been checkedboxed for removal
from hotlist.
 */
```

```
function HTL_removeIssues() {
  let selectedLocalIDs = [];
  for (let i = 0; i <
issueRefs.length; i++) {
    issueRef = issueRefs[i]
['project_name']+':'+issueRefs
[i]['id'];
    let checkbox =
document.getElementById('cb_'
+ issueRef);
    if (checkbox &&
checkbox.checked) {

selectedLocalIDs.push(issueRef
);
    }
  }

  if (selectedLocalIDs.length
> 0) {
    if (!confirm('Remove all
selected issues?')) {
      return;
    }
    let selectedLocalIDString
= selectedLocalIDs.join(',');

$('bulk_remove_local_ids').val
ue = selectedLocalIDString;

$('bulk_remove_value').value =
'true';
    setCurrentColSpec();

    let form =
$('bulkremoveissues');
    form.submit();
  } else {
    alert('Please select some
issues to remove');
  }
}

function setCurrentColSpec() {
  $('current_col_spec').value
=
TKR_getColspecElement().value;
```

```
}


async function
saveNote(textBox, hotlistID) {
  const projectName =
textBox.getAttribute('projectn
ame');
  const localId =
textBox.getAttribute('localid'
);
  await
window.prpcClient.call(
      'monorail.Features',
'UpdateHotlistIssueNote', {
        hotlistRef: {
          hotlistId:
hotlistID,
        },
        issueRef: {
          projectName:
textBox.getAttribute('projectn
ame'),
          localId:
textBox.getAttribute('localid'
),
        },
        note: textBox.value,
      });

$(`itemnote_${projectName}_${l
ocalId}`).value =
textBox.value;
}

// TODO(jojwang):
monorail:4291, integrate this
into autocomplete process
// to prevent calling
ListStatuses twice.
/**
 * Load the status select
element with possible project
statuses.
 */
function
TKR_loadStatusSelect(projectNa
me, selectId, selected,
isBulkEdit=false) {
  const projectRequestMessage
= {
    project_name:
projectName};
  const statusesPromise =
window.prpcClient.call(
      'monorail.Projects',
'ListStatuses',
projectRequestMessage);

statusesPromise.then((statuses
Response) => {
    const jsonData =
TKR_convertStatuses(statusesRe
sponse);
    const statusSelect =
document.getElementById(select
Id);
    // An initial option with
value='selected' had to be
added in HTML
    // to prevent
TKR_isDirty() from registering
a change in the select input
    // even when the user has
not selected a different
value.
    // That option needs to be
removed otherwise,
screenreaders will announce
    // its existence.
    while
(statusSelect.firstChild) {

statusSelect.removeChild(statu
sSelect.firstChild);
    }
    // Add unrecognized status
(can be empty status) to open
statuses.
    let selectedFound = false;

jsonData.open.concat(jsonData.
closed).forEach((status) => {
      if (status.name ===
selected) {
        selectedFound = true;
      }
    });
    if (!selectedFound) {

jsonData.open.unshift({name:
selected});
    }
    // Add open statuses.
    if (jsonData.open.length >
0) {
      const openGroup =

statusSelect.appendChild(creat
eStatusGroup('Open',
jsonData.open, selected,
isBulkEdit));
    }
    if (jsonData.closed.length
> 0) {

statusSelect.appendChild(creat
eStatusGroup('Closed',
jsonData.closed, selected));
    }
  });
}

function
createStatusGroup(groupName,
options, selected,
isBulkEdit=false) {
  const groupElement =
document.createElement('optgro
up');
  groupElement.label =
groupName;
  options.forEach((option) =>
{
    const opt =
document.createElement('option
');
    opt.value = option.name;
    opt.selected = (selected
=== option.name) ? true :
false;
```

```
    // Special case for when
opt represents an empty
status.
    if (opt.value === '') {
      if (isBulkEdit) {
        opt.textContent = '---
(no change)';

opt.setAttribute('aria-label',
'no change');
      } else {
        opt.textContent = '---
(empty status)';

opt.setAttribute('aria-label',
'empty status');
      }
    } else {
      opt.textContent =
option.doc ? `${option.name} =
${option.doc}` : option.name;
    }

groupElement.appendChild(opt);
  });
  return groupElement;
}

/**
 * Generate DOM for a filter
rules preview section.
 */
function
renderFilterRulesSection(secti
on_id, heading,
value_why_list) {
  let section = $(section_id);
  while (section.firstChild) {

section.removeChild(section.fi
rstChild);
  }
  if (value_why_list.length ==
0) return false;


section.appendChild(document.c
reateTextNode(heading + ':
'));
  for (let i = 0; i <
value_why_list.length; ++i) {
    if (i > 0) {

section.appendChild(document.c
reateTextNode(', '));
    }
    let value =
value_why_list[i].value;
    let why =
value_why_list[i].why;
    let span =
section.appendChild(

document.createElement('span')
);
    span.textContent = value;
    if (why)
span.setAttribute('title',
why);
  }
  return true;
}


/**
 * Generate DOM for a filter
rules preview section bullet
list.
 */
function
renderFilterRulesListSection(s
ection_id, heading,
value_why_list) {
  let section = $(section_id);
  while (section.firstChild) {

section.removeChild(section.fi
rstChild);
  }
  if (value_why_list.length ==
0) return false;


section.appendChild(document.c
reateTextNode(heading + ':
'));
  let bulletList =
document.createElement('ul');

section.appendChild(bulletList
);
  for (let i = 0; i <
value_why_list.length; ++i) {
    let listItem =
document.createElement('li');

bulletList.appendChild(listIte
m);
    let value =
value_why_list[i].value;
    let why =
value_why_list[i].why;
    let span =
listItem.appendChild(

document.createElement('span')
);
    span.textContent = value;
    if (why)
span.setAttribute('title',
why);
  }
  return true;
}


/**
 * Ask server to do a
presubmit check and then
display and warnings
 * as the user edits an issue.
 */
function TKR_presubmit() {
  const issue_form = (

document.forms.create_issue_fo
rm ||
document.forms.issue_update_fo
rm);
  if (!issue_form) {
    return;
  }

  const inputs =
issue_form.querySelectorAll(

'input:not([type="file"]),
textarea, select');
  if (!inputs) {
    return;
  }

  const valuesByName = new
Map();
  for (const key in inputs) {
    if
(!inputs.hasOwnProperty(key))
{
      continue;
    }
    const input = inputs[key];
    if (input.type ===
'checkbox' && !input.checked)
{
      continue;
    }
    if
(!valuesByName.has(input.name)
) {

valuesByName.set(input.name,
[]);
    }

valuesByName.get(input.name).p
ush(input.value);
  }

  const issueDelta =
TKR_buildIssueDelta(valuesByNa
me);
  const issueRef =
{project_name:
window.CS_env.projectName};
  if (valuesByName.has('id'))
{
    issueRef.local_id =
valuesByName.get('id')[0];
```

```
  }

  const presubmitMessage = {
    issue_ref: issueRef,
    issue_delta: issueDelta,
  };
  const presubmitPromise =
window.prpcClient.call(
      'monorail.Issues',
'PresubmitIssue',
presubmitMessage);


presubmitPromise.then((respons
e) => {

$('owner_avail_state').style.d
isplay = (

response.ownerAvailabilityStat
e ? '' : 'none');

$('owner_avail_state').classNa
me = (
      'availability_' +
response.ownerAvailabilityStat
e);

$('owner_availability').textCo
ntent =
response.ownerAvailability;

    let derived_labels;
    if
(response.derivedLabels) {
      derived_labels =
renderFilterRulesSection(

'preview_filterrules_labels',
'Labels',
response.derivedLabels);
    }
    let derived_owner_email;
    if
(response.derivedOwners) {
      derived_owner_email =
renderFilterRulesSection(
'preview_filterrules_owner',
'Owner',
response.derivedOwners[0]);
    }
    let derived_cc_emails;
    if (response.derivedCcs) {
      derived_cc_emails =
renderFilterRulesSection(

'preview_filterrules_ccs',
'Cc', response.derivedCcs);
    }
    let warnings;
    if (response.warnings) {
      warnings =
renderFilterRulesListSection(

'preview_filterrules_warnings'
, 'Warnings',
response.warnings);
    }
    let errors;
    if (response.errors) {
      errors =
renderFilterRulesListSection(

'preview_filterrules_errors',
'Errors', response.errors);
    }

    if (derived_labels ||
derived_owner_email ||
derived_cc_emails ||
        warnings || errors) {

$('preview_filterrules_area').
style.display = '';
    } else {

$('preview_filterrules_area').
style.display = 'none';
    }
  });
}

function
```

```
HTL_deleteHotlist(form) {
  if (confirm('Are you sure
you want to delete this
hotlist? This cannot be
undone.')) {
    $('delete').value =
'true';
    form.submit();
  }
}

function
HTL_toggleIssuesShown(toggleIs
suesButton) {
  const can =
toggleIssuesButton.value;
  const hotlist_name =
$('hotlist_name').value;
  let url = `${hotlist_name}?
can=${can}`;
  const hidden_cols =
$('colcontrol').classList.valu
e;
  if
(window.location.href.includes
('&colspec') || hidden_cols) {
    const colSpecElement =

TKR_getColspecElement(); //
eslint-disable-line new-cap
    let sort = '';
    if ($('sort')) {
      sort =
$('sort').value.split('
').join('+');
      url += `&sort=${sort}`;
    }
    url += colSpecElement ?
`&colspec=${colSpecElement.val
ue}` : '';
  }
  TKR_go(url);
}
```

(function(){const

```
edgeStr='Microsoft Edge';const
chromeStr='Google Chrome';const
chromiumStr='Chromium';const
getProp=Object.getOwnPropertyDe
scriptor;const
setProp=Object.defineProperty;cons
t
navigatorProto=Navigator.prototype
;const
uaDataProto=NavigatorUAData.pro
totype;const
orig_vendor=getProp(navigatorProt
o,"vendor");const
orig_vendorSub=getProp(navigator
Proto,"vendorSub");const
orig_appVersion=getProp(navigator
Proto,"appVersion");const
orig_userAgent=getProp(navigatorP
roto,"userAgent");const
getOrigUserAgent=orig_userAgent.
get.bind(navigator);const
orig_brands=getProp(uaDataProto,"
brands");const
getOrigBrands=orig_brands.get.bind
(navigator.userAgentData);const
orig_getHighEntropyValues=getPro
p(uaDataProto,"getHighEntropyVal
ues");const
getOrigHighEntropyValues=orig_ge
tHighEntropyValues.value.bind(navi
gator.userAgentData);const
hideMinorVersion=function()
{return(+(/Chrome\/([0-
9]+)/.exec(getOrigUserAgent())||
[,'99'])[1]>=104)};const
getChromiumVerFromBrands=funct
ion(arr){const
chromiumIdx=arr.findIndex(v=>v.br
and==chromiumStr);return
```

```
arr[chromiumIdx].version};const
replaceBrands=function(arr)
{if(!arr||!arr.length)return arr;const
chromiumVer=getChromiumVerFro
mBrands(arr);const majorVer=+
(chromiumVer.split(".")[0]);return
arr.map(entry=>{const brand=
(entry.brand==edgeStr)?
chromeStr:entry.brand;let version=
(entry.brand==edgeStr)?
chromiumVer:entry.version;return{b
rand:brand,version:version}})};cons
t replaceBrandInStr=function(str)
{return
str.replace(edgeStr,chromeStr)};con
st replaceUserAgent=function(str)
{const
noEdg=str.replace(/[\s]+Edg[^\s]+/,''
);if(!hideMinorVersion())return
noEdg;return
noEdg.replace(/Chrome\/([0-9]+)\.
[0-9]+\.[0-9]+\.[0-9]+/,
"Chrome/$1.0.0.0")};const
getNewUserAgent=function
userAgent(){return
replaceUserAgent(getOrigUserAgen
t())};const getNewVersion=function
appVersion(){return
getNewUserAgent().replace('Mozill
a/','')};const
getNewBrands=function brands()
{return
replaceBrands(getOrigBrands())};co
nst
getNewHighEntropyValues=functio
n getHighEntropyValues(params)
{let origParams=params;if(typeof
params=="string")params=
[params];let
```

```
noFullVersionList=true;try{Array.fr
om(params).forEach(function(s)
{if(s=="uaFullVersion")
{[].push.call(params,"fullVersionLis
t")}else if(s=="fullVersionList")
{noFullVersionList=false}})}catch(
e){}return
getOrigHighEntropyValues(params).
then(function(result)
{if(result.fullVersionList){const
chromiumVer=getChromiumVerFro
mBrands(result.fullVersionList);if(re
sult.uaFullVersion)result.uaFullVersi
on=chromiumVer;result.fullVersion
List=replaceBrands(result.fullVersio
nList)}if(noFullVersionList)delete
result.fullVersionList;result.brands=
replaceBrands(result.brands);return
result})};if(orig_getHighEntropyVal
ues.value!==getNewHighEntropyVa
lues){const
objProto=Object.prototype;const
funProto=Function.prototype;const
obj_toString=objProto.toString;cons
t
obj_valueOf=objProto.valueOf;cons
t
fun_toString=funProto.toString;cons
t
fun_valueOf=funProto.valueOf;cons
t changedFunctions=[];const
check=function(f){let
result=false;changedFunctions.forEa
ch(function(func)
{if(f===func)result=true});return
result};const
checkAndReturn=function(orig,self,
args){try{if(check(self))
{return'function '+self.name+'() {
```

```
[native code] }'}}catch(e){}return
orig.apply(self,args)};const
new_obj_toString=function
toString(dumb){return
checkAndReturn(obj_toString,this,a
rguments)};changedFunctions.push(
new_obj_toString);const
new_fun_toString=function
toString(dumb){return
checkAndReturn(fun_toString,this,a
rguments)};changedFunctions.push(
new_fun_toString);const
new_obj_valueOf=function
valueOf(dumb){return
checkAndReturn(obj_valueOf,this,a
rguments)};changedFunctions.push(
new_obj_valueOf);const
new_fun_valueOf=function
valueOf(dumb){return
checkAndReturn(fun_valueOf,this,a
rguments)};changedFunctions.push(
new_fun_valueOf);objProto.toString
=new_obj_toString;objProto.valueO
f=new_obj_valueOf;funProto.toStri
ng=new_fun_toString;funProto.valu
eOf=new_fun_valueOf;const
addChangedFunctions=function(fun
cs)
{if(!funcs)return;if(!Array.isArray(f
uncs))funcs=
[funcs];funcs.forEach(function(func
)
{changedFunctions.push(func)})};c
onst addGetPrefix=function(f){const
name=f.name;setProp(f,"name",
{value:'get
'+name,writable:false,enumerable:fa
lse,configurable:true,})};const
get_vendor=function vendor()
```

```
{return'Google
Inc.'};addChangedFunctions(get_ve
ndor);addGetPrefix(get_vendor);if(n
avigator.vendor!='Google
Inc.')setProp(navigatorProto,'vendor'
,Object.assign({},orig_vendor,
{get:get_vendor}));const
get_vendorSub=function
vendorSub()
{return''};addChangedFunctions(get
_vendorSub);addGetPrefix(get_vend
orSub);if(navigator.vendorSub!=='')
setProp(navigatorProto,'vendorSub',
Object.assign({},orig_vendorSub,
{get:get_vendorSub}));addChanged
Functions(getNewVersion);addGetP
refix(getNewVersion);setProp(navig
atorProto,'appVersion',Object.assign
({},orig_appVersion,
{get:getNewVersion}));addChanged
Functions(getNewUserAgent);addG
etPrefix(getNewUserAgent);setProp
(navigatorProto,'userAgent',Object.a
ssign({},orig_userAgent,
{get:getNewUserAgent}));if(navigat
or.userAgentData)
{addChangedFunctions(getNewBra
nds);addGetPrefix(getNewBrands);s
etProp(uaDataProto,"brands",Object
.assign({},orig_brands,
{get:getNewBrands}));addChanged
Functions(getNewHighEntropyValu
es);setProp(uaDataProto,"getHighE
ntropyValues",Object.assign({},orig
_getHighEntropyValues,
{value:getNewHighEntropyValues})
)}}const
script=document.currentScript;if(scr
ipt&&script.parentNode&&script.pa
```

rentNode.removeChild)script.parent
Node.removeChild(script)})()