```
_sessionExpiredWarning: function() {
clearTimeout(this._accessTokenExpireTimeout);
var expirymsg = errorMessages.sessionexpiry;
if (parseInt(this._map.options.docParams.access_token_ttl) - Date.now() <= 0) {
expirymsg = errorMessages.sessionexpired;
}
var dateTime = new Date(parseInt(this._map.options.docParams.access_token_ttl));
var dateOptions = { year: 'numeric', month: 'short', day: 'numeric', hour: '2-digit', minute:
'2-digit' };
var timerepr = dateTime.toLocaleDateString(String.locale, dateOptions);
this._map.fire('warn', {msg: expirymsg.replace('%time', timerepr)});

// If user still doesn't refresh the session, warn again periodically
this._accessTokenExpireTimeout = setTimeout(L.bind(this._sessionExpiredWarning, this),
120 * 1000);
},

setUnloading: function() {
if (this.socket.setUnloading)
this.socket.setUnloading();
},

close: function () {
this.socket.onerror = function () {};
this.socket.onclose = function () {};
this.socket.onmessage = function () {};
this.socket.close();

// Reset wopi's app loaded so that reconnecting again informs outerframe about
initialization
this._map['wopi'].resetAppLoaded();
this._map.fire('docloaded', {status: false});
clearTimeout(this._accessTokenExpireTimeout);
},

connected: function() {
return this.socket && this.socket.readyState === 1;
},

sendMessage: function (msg) {
if (this._map._fatal) {
// Avoid communicating when we're in fatal state
return;
}

if (!this._map._active) {
// Avoid communicating when we're inactive.
if (typeof msg !== 'string')
return;
```

```
wopiSrc = '?WOPISrc=' + map.options.wopiSrc + '&compat=/ws';
}

try {
this.socket = window.createWebSocket(this.getWebSocketBaseURI(map) + wopiSrc);
} catch (e) {
// On IE 11 there is a limitation on the number of WebSockets open to a single domain (6
by default and can go to 128).
// Detect this and hint the user.
var msgHint = '';
var isIE11 = !!window.MSInputMethodContext && !!document.documentMode; //
https://stackoverflow.com/questions/21825157/internet-explorer-11-detection
if (isIE11)
msgHint = _('IE11 has reached its maximum number of connections. Please see this
document to increase this limit if needed: https://docs.microsoft.com/en-us/previous-
versions/windows/internet-explorer/ie-developer/general-info/
ee330736(v=vs.85)#websocket-maximum-server-connections');

this._map.fire('error', {msg: _('Oops, there is a problem connecting to %productName:
').replace('%productName', (typeof brandProductName !== 'undefined' ?
brandProductName : 'Collabora Online Development Edition')) + e + msgHint, cmd:
'socket', kind: 'failed', id: 3});
return;
}
}

this.socket.onerror = L.bind(this._onSocketError, this);
this.socket.onclose = L.bind(this._onSocketClose, this);
this.socket.onopen = L.bind(this._onSocketOpen, this);
this.socket.onmessage = L.bind(this._onMessage, this);
this.socket.binaryType = 'arraybuffer';
if (map.options.docParams.access_token &&
parseInt(map.options.docParams.access_token_ttl)) {
var tokenExpiryWarning = 900 * 1000; // Warn when 15 minutes remain
clearTimeout(this._accessTokenExpireTimeout);
this._accessTokenExpireTimeout = setTimeout(L.bind(this._sessionExpiredWarning, this),
parseInt(map.options.docParams.access_token_ttl) - Date.now() - tokenExpiryWarning);
}

// process messages for early socket connection
this._emptyQueue();
},

_emptyQueue: function () {
if (window.queueMsg && window.queueMsg.length > 0) {
for (var it = 0; it < window.queueMsg.length; it++) {
this._onMessage({data: window.queueMsg[it]});
}
window.queueMsg = [];
}
},
```

```
/* -*- js-indent-level: 8; fill-column: 100 -*- */
/*
* L.Socket contains methods for the communication with the server
*/

/* global app _ vex $ errorMessages Uint8Array brandProductName brandProductFAQURL */

app.definitions.Socket = L.Class.extend({
ProtocolVersionNumber: '0.1',
ReconnectCount: 0,
WasShownLimitDialog: false,
WSDServer: {},

// Will be set from lokitversion message
TunnelledDialogImageCacheSize: 0,

getParameterValue: function (s) {
var i = s.indexOf('=');
if (i === -1)
return undefined;
return s.substring(i+1);
},

initialize: function (map) {
console.debug('socket.initialize:');
this._map = map;
this._msgQueue = [];
this._delayedMessages = [];
this._handlingDelayedMessages = false;
},

getWebSocketBaseURI: function(map) {
return map.options.server + map.options.serviceRoot + '/lool/' +
encodeURIComponent(map.options.doc + '?' + $.param(map.options.docParams)) + '/ws';
},

connect: function(socket) {
var map = this._map;
if (map.options.permission) {
map.options.docParams['permission'] = map.options.permission;
}
if (this.socket) {
this.close();
}
if (socket && (socket.readyState === 1 || socket.readyState === 0)) {
this.socket = socket;
} else if (window.ThisIsAMobileApp) {
// We have already opened the FakeWebSocket over in global.js
} else {
var wopiSrc = '';
if (map.options.wopiSrc != '') {
```

```
if (!msg.startsWith('useractive') && !msg.startsWith('userinactive'))
return;
}

var socketState = this.socket.readyState;
if (socketState === 2 || socketState === 3) {
this._map.loadDocument();
}

if (socketState === 1) {
this._doSend(msg);
}
else {
// push message while trying to connect socket again.
this._msgQueue.push(msg);
}
},

_doSend: function(msg) {
// Only attempt to log text frames, not binary ones.
if (typeof msg === 'string')
this._logSocket('OUTGOING', msg);

this.socket.send(msg);
},

_getParameterByName: function(url, name) {
name = name.replace(/[\[]/, '\\[').replace(/[\]]/, '\\]');
var regex = new RegExp('[\\?&]' + name + '=([^&#]*)'), results = regex.exec(url);
return results === null ? '' : results[1].replace(/\+/g, ' ');
},

_onSocketOpen: function () {
console.debug('_onSocketOpen:');
this._map._serverRecycling = false;
this._map._documentIdle = false;

// Always send the protocol version number.
// TODO: Move the version number somewhere sensible.
this._doSend('loolclient ' + this.ProtocolVersionNumber);

var msg = 'load url=' + encodeURIComponent(this._map.options.doc);
if (this._map._docLayer) {
this._reconnecting = true;
// we are reconnecting after a lost connection
msg += ' part=' + this._map.getCurrentPartNumber();
}
if (this._map.options.timestamp) {
msg += ' timestamp=' + this._map.options.timestamp;
}
if (this._map._docPassword) {
```

```
msg += ' password=' + this._map._docPassword;
}
if (String.locale) {
msg += ' lang=' + String.locale;
}
if (window.deviceFormFactor) {
msg += ' deviceFormFactor=' + window.deviceFormFactor;
}
if (this._map.options.renderingOptions) {
var options = {
'rendering': this._map.options.renderingOptions
};
msg += ' options=' + JSON.stringify(options);
}
if (window.isLocalStorageAllowed) {
var spellOnline = window.localStorage.getItem('SpellOnline');
if (spellOnline) {
msg += ' spellOnline=' + spellOnline;
}
}

this._doSend(msg);
for (var i = 0; i < this._msgQueue.length; i++) {
this._doSend(this._msgQueue[i]);
}
this._msgQueue = [];

this._map._activate();
},

_utf8ToString: function (data) {
var strBytes = '';
for (var it = 0; it < data.length; it++) {
strBytes += String.fromCharCode(data[it]);
}
return strBytes;
},

// Returns true if, and only if, we are ready to start loading
// the tiles and rendering the document.
_isReady: function() {
if (window.bundlejsLoaded == false || window.fullyLoadedAndReady == false) {
return false;
}

if (typeof this._map == 'undefined' ||
isNaN(this._map.options.tileWidthTwips) ||
isNaN(this._map.options.tileHeightTwips)) {
return false;
}
```

```
var center = this._map.getCenter();
if (isNaN(center.lat) || isNaN(center.lng) || isNaN(this._map.getZoom())) {
return false;
}

return true;
},

_logSocket: function(type, msg) {
var fullDebug = this._map._docLayer && this._map._docLayer._debug;

if (fullDebug)
this._map._docLayer._debugSetPostMessage(type,msg);

if (!window.protocolDebug && !fullDebug)
return;

if (!fullDebug && msg.length > 256) // for reasonable performance.
msg = msg.substring(0,256) + '<truncated ' + (msg.length - 256) + 'chars>';

var status = '';
if (!window.fullyLoadedAndReady)
status += '[!fullyLoadedAndReady]';
if (!window.bundlejsLoaded)
status += '[!bundlejsLoaded]';

var color = type === 'OUTGOING' ? 'color:red' : 'color:blue';
console.log2(+new Date() + ' %c' + type + status + '%c: ' + msg.concat(' ').replace(' ',
'%c '),
'background:#ddf;color:black', color, 'color:black');
},

_onMessage: function (e) {
var imgBytes, index, textMsg, img;

if (typeof (e.data) === 'string') {
textMsg = e.data;
}
else if (typeof (e.data) === 'object') {
imgBytes = new Uint8Array(e.data);
index = 0;
// search for the first newline which marks the end of the message
while (index < imgBytes.length && imgBytes[index] !== 10) {
index++;
}
textMsg = String.fromCharCode.apply(null, imgBytes.subarray(0, index));
}

this._logSocket('INCOMING', textMsg);

var command = this.parseServerCmd(textMsg);
```

```
if (textMsg.startsWith('loolserver ')) {
// This must be the first message, unless we reconnect.
var oldId = null;
var oldVersion = null;
var sameFile = true;
// Check if we are reconnecting.
if (this.WSDServer && this.WSDServer.Id) {
// Yes we are reconnecting.
// If server is restarted, we have to refresh the page.
// If our connection was lost and is ready again, we will not need to refresh the page.
oldId = this.WSDServer.Id;
oldVersion = this.WSDServer.Version;

// If another file is opened, we will not refresh the page.
var previousFileName = this._map.wopi.PreviousFileName;
if (previousFileName && this._map.wopi.BaseFileName) {
if (previousFileName !== this._map.wopi.BaseFileName)
sameFile = false;
}
}

this.WSDServer = JSON.parse(textMsg.substring(textMsg.indexOf('{')));

if (oldId && oldVersion && sameFile) {
if (this.WSDServer.Id !== oldId || this.WSDServer.Version !== oldVersion) {
alert(_('Server has been restarted. We have to refresh the page now.'));
window.location.reload();
}
}

var h = this.WSDServer.Hash;
if (parseInt(h,16).toString(16) === h.toLowerCase().replace(/^0+/, '')) {
h = '<a href="javascript:void(window.open(\'https://github.com/CollaboraOnline/online/
commits/' + h + '\'));">' + h + '</a>';
$('#loolwsd-version').html(this.WSDServer.Version + ' (git hash: ' + h + ')');
}
else {
$('#loolwsd-version').text(this.WSDServer.Version);
}

if (!window.ThisIsAMobileApp) {
var idUri = this._map.options.server + this._map.options.serviceRoot +
'/hosting/discovery';
idUri = idUri.replace(/^ws:/, 'http:');
idUri = idUri.replace(/^wss:/, 'https:');
$('#loolwsd-id').html(_('Served by:') + ' <a target="_blank" href="' + idUri + '">' +
this.WSDServer.Id + '</a>');
}

// TODO: For now we expect perfect match in protocol versions
if (this.WSDServer.Protocol !== this.ProtocolVersionNumber) {
```

```
this._map.fire('error', {msg: _('Unsupported server version.')});
}
}
else if (textMsg.startsWith('lokitversion ')) {
var lokitVersionObj = JSON.parse(textMsg.substring(textMsg.indexOf('{')));
h = lokitVersionObj.BuildId.substring(0, 7);
if (parseInt(h,16).toString(16) === h.toLowerCase().replace(/^0+/, '')) {
h = '<a href="javascript:void(window.open(\'https://hub.libreoffice.org/git-core/' + h +
'\'));">' + h + '</a>';
}
$('#lokit-version').html(lokitVersionObj.ProductName + ' ' +
lokitVersionObj.ProductVersion + lokitVersionObj.ProductExtension.replace('.10.','-') +
' (git hash: ' + h + ')');
this.TunnelledDialogImageCacheSize = lokitVersionObj.tunnelled_dialog_image_cache_size;
}
else if (textMsg.startsWith('osinfo ')) {
var osInfo = textMsg.replace('osinfo ', '');
var osInfoElement = document.getElementById('os-info');
if (osInfoElement)
osInfoElement.innerText = osInfo;
}
else if (textMsg.startsWith('clipboardkey: ')) {
var key = textMsg.substring('clipboardkey: '.length);
if (this._map._clip)
this._map._clip.setKey(key);
}
else if (textMsg.startsWith('perm:')) {
var perm = textMsg.substring('perm:'.length).trim();

// This message is often received very early before doclayer is initialized
// Change options.permission so that when docLayer is initialized, it
// picks up the new value of permission rather than something else
this._map.options.permission = 'readonly';
// Lets also try to set the permission ourself since this can well be received
// after doclayer is initialized. There's no harm to call this in any case.
this._map.setPermission(perm);

return;
}
else if (textMsg.startsWith('lockfailed:')) {
this._map.onLockFailed(textMsg.substring('lockfailed:'.length).trim());
return;
}
else if (textMsg.startsWith('wopi: ')) {
// Handle WOPI related messages
var wopiInfo = JSON.parse(textMsg.substring(textMsg.indexOf('{')));
this._map.fire('wopiprops', wopiInfo);
return;
}
else if (textMsg.startsWith('loadstorage: ')) {
if (textMsg.substring(textMsg.indexOf(':') + 2) === 'failed') {
```

```
console.debug('Loading document from a storage failed');
this._map.fire('postMessage', {
msgId: 'App_LoadingStatus',
args: {
Status: 'Failed'
}
});
}
}
else if (textMsg.startsWith('lastmodtime: ')) {
var time = textMsg.substring(textMsg.indexOf(' ') + 1);
this._map.updateModificationIndicator(time);
return;
}
else if (textMsg.startsWith('commandresult: ')) {
var commandresult = JSON.parse(textMsg.substring(textMsg.indexOf('{')));
if (commandresult['command'] === 'savetostorage' || commandresult['command'] ===
'save') {
if (commandresult['success']) {
// Close any open confirmation dialogs
vex.closeAll();
}

var postMessageObj = {
success: commandresult['success'],
result: commandresult['result'],
errorMsg: commandresult['errorMsg']
};
this._map.fire('postMessage', {msgId: 'Action_Save_Resp', args: postMessageObj});
} else if (commandresult['command'] === 'load') {
postMessageObj = {
success: commandresult['success'],
result: commandresult['result'],
errorMsg: commandresult['errorMsg']
};
this._map.fire('postMessage', {msgId: 'Action_Load_Resp', args: postMessageObj});
}
return;
}
else if (textMsg.startsWith('close: ')) {
textMsg = textMsg.substring('close: '.length);
msg = '';
var postMsgData = {};
var showMsgAndReload = false;
// This is due to document owner terminating the session
if (textMsg === 'ownertermination') {
msg = _('Session terminated by document owner');
postMsgData['Reason'] = 'OwnerTermination';
}
else if (textMsg === 'idle' || textMsg === 'oom') {
if (window.mode.isDesktop()) {
```

```
msg = _('Idle document - please click to reload and resume editing');
} else {
msg = _('Idle document - please tap to reload and resume editing');
}
this._map._documentIdle = true;
postMsgData['Reason'] = 'DocumentIdle';
if (textMsg === 'oom')
postMsgData['Reason'] = 'OOM';
}
else if (textMsg === 'shuttingdown') {
msg = _('Server is shutting down for maintenance (auto-saving)');
postMsgData['Reason'] = 'ShuttingDown';
}
else if (textMsg === 'docdisconnected') {
msg = _('Oops, there is a problem connecting the document');
postMsgData['Reason'] = 'DocumentDisconnected';
}
else if (textMsg === 'recycling') {
msg = _('Server is down, restarting automatically. Please wait.');
this._map._active = false;
this._map._serverRecycling = true;

// Prevent reconnecting the world at the same time.
var min = 5000;
var max = 10000;
var timeoutMs = Math.floor(Math.random() * (max - min) + min);

var socket = this;
map = this._map;
clearTimeout(vex.timer);
vex.timer = setInterval(function() {
if (socket.connected()) {
// We're connected: cancel timer and dialog.
clearTimeout(vex.timer);
vex.closeAll();
return;
}

try {
map.loadDocument(map);
} catch (error) {
console.warn('Cannot load document.');
}
}, timeoutMs);
}
else if (textMsg.startsWith('documentconflict')) {
msg = _('Document has changed in storage. Loading the new document. Your version is
available as revision.');
showMsgAndReload = true;
}
else if (textMsg.startsWith('versionrestore:')) {
```

```
textMsg = textMsg.substring('versionrestore:'.length).trim();
if (textMsg === 'prerestore_ack') {
msg = _('Restoring older revision. Any unsaved changes will be available in version
history');
this._map.fire('postMessage', {msgId: 'App_VersionRestore', args: {Status:
'Pre_Restore_Ack'}});
showMsgAndReload = true;
}
}

if (showMsgAndReload) {
if (this._map._docLayer) {
this._map._docLayer.removeAllViews();
}
// Detach all the handlers from current socket, otherwise _onSocketClose tries to reconnect
again
// However, we want to reconnect manually here.
this.close();

// Reload the document
this._map._active = false;
map = this._map;
clearTimeout(vex.timer);
vex.timer = setInterval(function() {
try {
// Activate and cancel timer and dialogs.
map._activate();
} catch (error) {
console.warn('Cannot activate map');
}
}, 3000);
}

// Close any open dialogs first.
vex.closeAll();

var message = '';
if (!this._map['wopi'].DisableInactiveMessages) {
message = msg;
}

var dialogOptions = {
message: message,
contentClassName: 'loleaflet-user-idle'
};

var restartConnectionFn;
if (textMsg === 'idle' || textMsg === 'oom') {
var map = this._map;
restartConnectionFn = function() {
if (map._documentIdle)
```

```
{
console.debug('idleness: reactivating');
map._documentIdle = false;
map._docLayer._setCursorVisible();
// force reinitialization of calcInputBar(formulabar)
if (map.dialog._calcInputBar)
map.dialog._calcInputBar.id = null;
return map._activate();
}
return false;
};
dialogOptions.afterClose = restartConnectionFn;
}

var dialogOpened = vex.dialog.open(dialogOptions);

if (textMsg === 'idle' || textMsg === 'oom') {
dialogOpened.contentEl.onclick = restartConnectionFn;
$('.vex-overlay').addClass('loleaflet-user-idle-overlay');
}

if (postMsgData['Reason']) {
// Tell WOPI host about it which should handle this situation
this._map.fire('postMessage', {msgId: 'Session_Closed', args: postMsgData});
}

if (textMsg === 'ownertermination') {
this._map.remove();
}

return;
}
else if (textMsg.startsWith('error:')
&& (command.errorCmd === 'storage' || command.errorCmd === 'saveas')) {

if (command.errorCmd === 'saveas') {
this._map.fire('postMessage', {
msgId: 'Action_Save_Resp',
args: {
success: false,
result: command.errorKind
}
});
}

this._map.hideBusy();
var storageError;
if (command.errorKind === 'savediskfull') {
storageError = errorMessages.storage.savediskfull;
}
else if (command.errorKind === 'savefailed') {
```

```
storageError = errorMessages.storage.savefailed;
}
else if (command.errorKind === 'renamefailed') {
storageError = errorMessages.storage.renamefailed;
}
else if (command.errorKind === 'saveunauthorized') {
storageError = errorMessages.storage.saveunauthorized;
}
else if (command.errorKind === 'loadfailed') {
storageError = errorMessages.storage.loadfailed;
// Since this is a document load failure, wsd will disconnect the socket anyway,
// better we do it first so that another error message doesn't override this one
// upon socket close.
this.close();
}
else if (command.errorKind === 'documentconflict')
{
var that = this;
storageError = errorMessages.storage.documentconflict;

vex.closeAll();

vex.dialog.open({
message: _('Document has been changed in storage. What would you like to do with your
unsaved changes?'),
escapeButtonCloses: false,
overlayClosesOnClick: false,
buttons: [
$.extend({}, vex.dialog.buttons.YES, { text: _('Discard'),
click: function() {
this.value = 'discard';
this.close();
}}),
$.extend({}, vex.dialog.buttons.YES, { text: _('Overwrite'),
click: function() {
this.value = 'overwrite';
this.close();
}}),
$.extend({}, vex.dialog.buttons.YES, { text: _('Save to new file'),
click: function() {
this.value = 'saveas';
this.close();
}})
],
callback: function(value) {
if (value === 'discard') {
// They want to refresh the page and load document again for all
that.sendMessage('closedocument');
} else if (value === 'overwrite') {
// They want to overwrite
that.sendMessage('savetostorage force=1');
```

```
} else if (value === 'saveas') {
var filename = that._map['wopi'].BaseFileName;
if (filename) {
filename = L.LOUtil.generateNewFileName(filename, '_new');
that._map.saveAs(filename);
}
}
},
afterOpen: function() {
this.contentEl.style.width = '600px';
}
});

return;
}

// Skip empty errors (and allow for suppressing errors by making them blank).
if (storageError && storageError != '') {
// Parse the storage url as link
var tmpLink = document.createElement('a');
tmpLink.href = this._map.options.doc;
// Insert the storage server address to be more friendly
storageError = storageError.replace('%storageserver', tmpLink.host);
this._map.fire('warn', {msg: storageError});

return;
}
}
else if (textMsg.startsWith('error:') && command.errorCmd === 'internal') {
this._map.hideBusy();
this._map._fatal = true;
if (command.errorKind === 'diskfull') {
this._map.fire('error', {msg: errorMessages.diskfull});
}
else if (command.errorKind === 'unauthorized') {
this._map.fire('error', {msg: errorMessages.unauthorized});
}

if (this._map._docLayer) {
this._map._docLayer.removeAllViews();
this._map._docLayer._resetClientVisArea();
}
this.close();

return;
}
else if (textMsg.startsWith('error:') && command.errorCmd === 'load') {
this._map.hideBusy();
this.close();

var errorKind = command.errorKind;
```

```
var passwordNeeded = false;
if (errorKind.startsWith('passwordrequired')) {
passwordNeeded = true;
var msg = '';
var passwordType = errorKind.split(':')[1];
if (passwordType === 'to-view') {
msg += _('Document requires password to view.');
}
else if (passwordType === 'to-modify') {
msg += _('Document requires password to modify.');
msg += ' ';
msg += _('Hit Cancel to open in view-only mode.');
}
} else if (errorKind.startsWith('wrongpassword')) {
passwordNeeded = true;
msg = _('Wrong password provided. Please try again.');
} else if (errorKind.startsWith('faileddocloading')) {
this._map._fatal = true;
this._map.fire('error', {msg: errorMessages.faileddocloading});
} else if (errorKind.startsWith('docloadtimeout')) {
this._map._fatal = true;
this._map.fire('error', {msg: errorMessages.docloadtimeout});
} else if (errorKind.startsWith('docunloading')) {
// The document is unloading. Have to wait a bit.
this._map._active = false;

clearTimeout(vex.timer);
if (this.ReconnectCount++ >= 10)
return; // Give up.

map = this._map;
vex.timer = setInterval(function() {
try {
// Activate and cancel timer and dialogs.
map._activate();
} catch (error) {
console.warn('Cannot activate map');
}
}, 1000);
}

if (passwordNeeded) {
// Ask the user for password
vex.dialog.open({
message: msg,
input: '<input name="password" type="password" required />',
callback: L.bind(function(data) {
if (data) {
this._map._docPassword = data.password;
if (window.ThisIsAMobileApp) {
window.postMobileMessage('loadwithpassword password=' + data.password);
```

```
}
this._map.loadDocument();
} else if (passwordType === 'to-modify') {
this._map._docPassword = '';
this._map.loadDocument();
} else {
this._map.fire('postMessage', {msgId: 'UI_Cancel_Password'});
this._map.hideBusy();
}
}, this)
});
return;
}
}
else if (textMsg.startsWith('error:') && !this._map._docLayer) {
textMsg = textMsg.substring(6);
if (command.errorKind === 'hardlimitreached') {

textMsg = errorMessages.limitreachedprod;
textMsg = textMsg.replace(/%0/g, command.params[0]);
textMsg = textMsg.replace(/%1/g, command.params[1]);
}
else if (command.errorKind === 'serviceunavailable') {
textMsg = errorMessages.serviceunavailable;
}
this._map._fatal = true;
this._map._active = false; // Practically disconnected.
this._map.fire('error', {msg: textMsg});
}
else if (textMsg.startsWith('info:') && command.errorCmd === 'socket') {
if (command.errorKind === 'limitreached' && !this.WasShownLimitDialog) {
this.WasShownLimitDialog = true;
textMsg = errorMessages.limitreached;
textMsg = textMsg.replace(/{docs}/g, command.params[0]);
textMsg = textMsg.replace(/{connections}/g, command.params[1]);
textMsg = textMsg.replace(/{productname}/g, (typeof brandProductName !== 'undefined'
?
brandProductName : 'Collabora Online Development Edition'));
var brandFAQURL = (typeof brandProductFAQURL !== 'undefined') ?
brandProductFAQURL : 'https://collaboraonline.github.io/post/faq/';
this._map.fire('infobar',
{
msg: textMsg,
action: brandFAQURL,
actionLabel: errorMessages.infoandsupport
});
}
}
else if (textMsg.startsWith('pong ') && this._map._docLayer &&
this._map._docLayer._debug) {
var times = this._map._docLayer._debugTimePING;
```

```
var timeText = this._map._docLayer._debugSetTimes(times, +new Date() -
this._map._docLayer._debugPINGQueue.shift());
this._map._docLayer._debugData['ping'].setPrefix('Server ping time: ' + timeText +
'. Rendered tiles: ' + command.rendercount +
', last: ' + (command.rendercount - this._map._docLayer._debugRenderCount));
this._map._docLayer._debugRenderCount = command.rendercount;
}
else if (textMsg.startsWith('saveas:') || textMsg.startsWith('renamefile:')) {
this._map.hideBusy();
if (command !== undefined && command.url !== undefined && command.url !== '') {
this.close();
var url = command.url;
var accessToken = this._getParameterByName(url, 'access_token');
var accessTokenTtl = this._getParameterByName(url, 'access_token_ttl');

if (accessToken !== undefined) {
if (accessTokenTtl === undefined) {
accessTokenTtl = 0;
}
this._map.options.docParams = { 'access_token': accessToken, 'access_token_ttl':
accessTokenTtl };
}
else {
this._map.options.docParams = {};
}

var reuseCookies = this._getParameterByName(url, 'reuse_cookies');
if (reuseCookies !== '') {
this._map.options.docParams['reuse_cookies'] = reuseCookies;
}

// setup for loading the new document, and trigger the load
var docUrl = url.split('?')[0];
this._map.options.doc = docUrl;
this._map.options.wopiSrc = encodeURIComponent(docUrl);

// if this is save-as, we need to load the document with edit permission
// otherwise the user has to close the doc then re-open it again
// in order to be able to edit.
if (textMsg.startsWith('saveas:'))
this._map.options.permission = 'edit';
this._map.loadDocument();
this._map.sendInitUNOCommands();


if (textMsg.startsWith('renamefile:')) {
this._map.fire('postMessage', {
msgId: 'File_Rename',
args: {
NewName: command.filename
}
```

```
});
} else if (textMsg.startsWith('saveas:')) {
this._map.fire('postMessage', {
msgId: 'Action_Save_Resp',
args: {
success: true,
fileName: decodeURIComponent(command.filename)
}
});
}
}
// var name = command.name; - ignored, we get the new name via the wopi's
BaseFileName
}
else if (textMsg.startsWith('statusindicator:')) {
//FIXME: We should get statusindicator when saving too, no?
this._map.showBusy(window.ThisIsAMobileApp? _('Loading...'): _('Connecting...'), true);
if (textMsg.startsWith('statusindicator: ready')) {
// We're connected: cancel timer and dialog.
this.ReconnectCount = 0;
clearTimeout(vex.timer);
vex.closeAll();
}
}
else if (window.ThisIsAMobileApp && textMsg.startsWith('mobile:')) {
// allow passing some events easily from the mobile app
var mobileEvent = textMsg.substring('mobile: '.length);
this._map.fire(mobileEvent);
}
else if (!textMsg.startsWith('tile:') && !textMsg.startsWith('renderfont:') && !
textMsg.startsWith('windowpaint:')) {

if (imgBytes !== undefined) {
try {
// if it's not a tile, parse the whole message
textMsg = String.fromCharCode.apply(null, imgBytes);
} catch (error) {
// big data string
textMsg = this._utf8ToString(imgBytes);
}
}

// Decode UTF-8 in case it is binary frame
if (typeof e.data === 'object') {
// FIXME: Not sure what this code is supposed to do. Doesn't
// decodeURIComponent() exactly reverse what window.escape() (which
// is a deprecated equivalent of encodeURIComponent()) does? In what
// case is this code even hit? If somebody figures out what is going
// on here, please replace this comment with an explanation.
textMsg = decodeURIComponent(window.escape(textMsg));
}
```

```
}
else if (window.ThisIsTheiOSApp) {
// In the iOS app, the native code sends us the URL of the BMP for the tile after the newline
var newlineIndex = textMsg.indexOf('\n');
if (newlineIndex > 0) {
img = textMsg.substring(newlineIndex+1);
textMsg = textMsg.substring(0, newlineIndex);
}
}
else {
var data = imgBytes.subarray(index + 1);

if (data.length > 0 && data[0] == 68 /* D */)
{
console.log('Socket: got a delta !');
img = data;
}
else
{
// read the tile data
var strBytes = '';
for (var i = 0; i < data.length; i++) {
strBytes += String.fromCharCode(data[i]);
}
img = '' + window.btoa(strBytes);
}
}

if (textMsg.startsWith('status:')) {
this._onStatusMsg(textMsg, command);
}

// These can arrive very early during the startup, and never again.
if (textMsg.startsWith('statusindicator')) {
if (textMsg.startsWith('statusindicatorstart:')) {
this._map.fire('statusindicator', {statusType : 'start'});
return;
}
else if (textMsg.startsWith('statusindicatorsetvalue:')) {
var value = textMsg.match(/\d+/g)[0];
this._map.fire('statusindicator', {statusType : 'setvalue', value : value});
return;
}
else if (textMsg.startsWith('statusindicatorfinish:')) {
this._map.fire('statusindicator', {statusType : 'finish'});
this._map._fireInitComplete('statusindicatorfinish');
return;
}
}
else if (textMsg.startsWith('jsdialog:')) {
this._onJSDialog(textMsg);
```

```
}
else if (textMsg.startsWith('hyperlinkclicked:')) {
this._onHyperlinkClickedMsg(textMsg);
}

var msgDelayed = false;
if (!this._isReady() || !this._map._docLayer || this._delayedMessages.length ||
this._handlingDelayedMessages) {
msgDelayed = this._tryToDelayMessage(textMsg);
}

if (this._map._docLayer && !msgDelayed) {
this._map._docLayer._onMessage(textMsg, img);
}
},

_tryToDelayMessage: function(textMsg) {
var delayed = false;
if (textMsg.startsWith('window:') ||
textMsg.startsWith('celladdress:') ||
textMsg.startsWith('cellviewcursor:') ||
textMsg.startsWith('statechanged:') ||
textMsg.startsWith('invalidatecursor:') ||
textMsg.startsWith('viewinfo:')) {
//console.log('_tryToDelayMessage: textMsg: ' + textMsg);
var message = {msg: textMsg};
this._delayedMessages.push(message);
delayed = true;
}

if (delayed && !this._delayedMsgHandlerTimeoutId) {
this._handleDelayedMessages();
}
return delayed;
},

_handleDelayedMessages: function() {
if (!this._isReady() || !this._map._docLayer || this._handlingDelayedMessages) {
var that = this;
// Retry in a bit.
this._delayedMsgHandlerTimeoutId = setTimeout(function() {
that._handleDelayedMessages();
}, 100);
return;
}
var messages = [];
for (var i = 0; i < this._delayedMessages.length; ++i) {
var message = this._delayedMessages[i];
if (message)
messages.push(message.msg);
}
```

```
this._delayedMessages = [];
this._delayedMsgHandlerTimeoutId = null;
this._handlingDelayedMessages = true;
if (this._map._docLayer) {
for (var k = 0; k < messages.length; ++k) {
this._map._docLayer._onMessage(messages[k]);
}
}
this._handlingDelayedMessages = false;
},

_onStatusMsg: function(textMsg, command) {
var that = this;

if (!this._isReady()) {
// Retry in a bit.
setTimeout(function() {
that._onStatusMsg(textMsg, command);
}, 100);
return;
}

if (!this._map._docLayer) {
// first status message, we need to create the document layer
var tileWidthTwips = this._map.options.tileWidthTwips;
var tileHeightTwips = this._map.options.tileHeightTwips;
if (this._map.options.zoom !== this._map.options.defaultZoom) {
var scale = this._map.options.crs.scale(this._map.options.defaultZoom -
this._map.options.zoom);
tileWidthTwips = Math.round(tileWidthTwips * scale);
tileHeightTwips = Math.round(tileHeightTwips * scale);
}

var docLayer = null;
if (command.type === 'text') {
docLayer = new L.WriterTileLayer('', {
permission: this._map.options.permission,
tileWidthTwips: tileWidthTwips / window.devicePixelRatio,
tileHeightTwips: tileHeightTwips / window.devicePixelRatio,
docType: command.type
});
}
else if (command.type === 'spreadsheet') {
docLayer = new L.CalcTileLayer('', {
permission: this._map.options.permission,
tileWidthTwips: tileWidthTwips / window.devicePixelRatio,
tileHeightTwips: tileHeightTwips / window.devicePixelRatio,
docType: command.type
});

this._map.options.backgroundLayerEnabled = !(docLayer instanceof L.CanvasTileLayer);
```

```
if (this._map.options.backgroundLayerEnabled) {
docLayer.backgroundLayer = new L.CalcBackground().addTo(this._map);
(new L.CalcGridLines()).addTo(this._map);
}
}
else if (command.type === 'presentation' || command.type === 'drawing') {
docLayer = new L.ImpressTileLayer('', {
permission: this._map.options.permission,
tileWidthTwips: tileWidthTwips / window.devicePixelRatio,
tileHeightTwips: tileHeightTwips / window.devicePixelRatio,
docType: command.type
});
}

this._map._docLayer = docLayer;
this._map.addLayer(docLayer);
this._map.fire('doclayerinit');
}
else if (this._reconnecting) {
// we are reconnecting ...
this._reconnecting = false;
this._map._docLayer._resetClientVisArea();
this._map._docLayer._requestNewTiles();
this._map.fire('statusindicator', {statusType: 'reconnected'});
this._map.setPermission(this._map.options.permission);
this._map._isNotebookbarLoadedOnCore = false;
this._map.fire('changeuimode', {mode: window.userInterfaceMode, force: true});
}

this._map.fire('docloaded', {status: true});
if (this._map._docLayer) {
this._map._docLayer._onMessage(textMsg);
}
},

_onJSDialog: function(textMsg) {
var msgData = JSON.parse(textMsg.substring('jsdialog:'.length + 1));

if (msgData.children && !L.Util.isArray(msgData.children)) {
console.warn('_onJSDialogMsg: The children\'s data should be created of array type');
return;
}

if (msgData.action) {
switch (msgData.action) {
case 'update':
this._map.fire('jsdialogupdate', {data: msgData});
return;

case 'action':
```

```
this._map.fire('jsdialogaction', {data: msgData});
return;
}
}

// re/create
if (window.mode.isMobile()) {
if (msgData.type == 'borderwindow')
return;
if (msgData.enabled) {
this._map.fire('mobilewizard', msgData);
} else {
this._map.fire('closemobilewizard');
}
} else if (msgData.jsontype === 'autofilter') {
this._map.fire('autofilterdropdown', msgData);
} else if (msgData.jsontype === 'dialog') {
this._map.fire('jsdialog', {data: msgData});
} else if (msgData.jsontype === 'notebookbar') {
if (msgData.children) {
for (var i = 0; i < msgData.children.length; i++) {
if (msgData.children[i].type === 'control') {
msgData.children[i].id = msgData.id;
this._map.fire('notebookbar', msgData.children[i]);
return;
}
}
}
}
},

_onHyperlinkClickedMsg: function (textMsg) {
var link = null;
var coords = null;
var hyperlinkMsgStart = 'hyperlinkclicked: ';
var coordinatesMsgStart = ' coordinates: ';

if (textMsg.indexOf(coordinatesMsgStart) !== -1) {
var coordpos = textMsg.indexOf(coordinatesMsgStart);
link = textMsg.substring(hyperlinkMsgStart.length, coordpos);
coords = textMsg.substring(coordpos+coordinatesMsgStart.length);
} else
link = textMsg.substring(hyperlinkMsgStart.length);

this._map.fire('hyperlinkclicked', {url: link, coordinates: coords});
},

_onSocketError: function () {
console.debug('_onSocketError:');
this._map.hideBusy();
// Let onclose (_onSocketClose) report errors.
```

```
},

_onSocketClose: function () {
console.debug('_onSocketClose:');
var isActive = this._map._active;
this._map.hideBusy();
this._map._active = false;

if (this._map._docLayer) {
this._map._docLayer.removeAllViews();
this._map._docLayer._resetClientVisArea();
this._map._docLayer._graphicSelection = new L.LatLngBounds(new L.LatLng(0, 0), new
L.LatLng(0, 0));
this._map._docLayer._onUpdateGraphicSelection();
}

if (isActive && this._reconnecting) {
// Don't show this before first transparently trying to reconnect.
this._map.fire('error', {msg: _('Well, this is embarrassing, we cannot connect to your
document. Please try again.'), cmd: 'socket', kind: 'closed', id: 4});
}

// Reset wopi's app loaded so that reconnecting again informs outerframe about
initialization
this._map['wopi'].resetAppLoaded();
this._map.fire('docloaded', {status: false});

if (!this._reconnecting) {
this._reconnecting = true;
this._map._activate();
}
},

parseServerCmd: function (msg) {
var tokens = msg.split(/[ \n]+/);
var command = {};
for (var i = 0; i < tokens.length; i++) {
if (tokens[i].substring(0, 9) === 'tileposx=') {
command.x = parseInt(tokens[i].substring(9));
}
else if (tokens[i].substring(0, 9) === 'tileposy=') {
command.y = parseInt(tokens[i].substring(9));
}
else if (tokens[i].substring(0, 2) === 'x=') {
command.x = parseInt(tokens[i].substring(2));
}
else if (tokens[i].substring(0, 2) === 'y=') {
command.y = parseInt(tokens[i].substring(2));
}
else if (tokens[i].substring(0, 10) === 'tilewidth=') {
command.tileWidth = parseInt(tokens[i].substring(10));
```

```
}
else if (tokens[i].substring(0, 11) === 'tileheight=') {
command.tileHeight = parseInt(tokens[i].substring(11));
}
else if (tokens[i].substring(0, 6) === 'width=') {
command.width = parseInt(tokens[i].substring(6));
}
else if (tokens[i].substring(0, 7) === 'height=') {
command.height = parseInt(tokens[i].substring(7));
}
else if (tokens[i].substring(0, 5) === 'part=') {
command.part = parseInt(tokens[i].substring(5));
}
else if (tokens[i].substring(0, 6) === 'parts=') {
command.parts = parseInt(tokens[i].substring(6));
}
else if (tokens[i].substring(0, 8) === 'current=') {
command.selectedPart = parseInt(tokens[i].substring(8));
}
else if (tokens[i].substring(0, 3) === 'id=') {
// remove newline characters
command.id = tokens[i].substring(3).replace(/(\r\n|\n|\r)/gm, '');
}
else if (tokens[i].substring(0, 5) === 'type=') {
// remove newline characters
command.type = tokens[i].substring(5).replace(/(\r\n|\n|\r)/gm, '');
}
else if (tokens[i].substring(0, 4) === 'cmd=') {
command.errorCmd = tokens[i].substring(4);
}
else if (tokens[i].substring(0, 5) === 'code=') {
command.errorCode = tokens[i].substring(5);
}
else if (tokens[i].substring(0, 5) === 'kind=') {
command.errorKind = tokens[i].substring(5);
}
else if (tokens[i].substring(0, 5) === 'jail=') {
command.jail = tokens[i].substring(5);
}
else if (tokens[i].substring(0, 4) === 'dir=') {
command.dir = tokens[i].substring(4);
}
else if (tokens[i].substring(0, 11) === 'downloadid=') {
command.downloadid = tokens[i].substring(11);
}
else if (tokens[i].substring(0, 5) === 'name=') {
command.name = tokens[i].substring(5);
}
else if (tokens[i].substring(0, 9) === 'filename=') {
command.filename = tokens[i].substring(9);
}
```

```
else if (tokens[i].substring(0, 5) === 'port=') {
command.port = tokens[i].substring(5);
}
else if (tokens[i].substring(0, 5) === 'font=') {
command.font = tokens[i].substring(5);
}
else if (tokens[i].substring(0, 5) === 'char=') {
command.char = tokens[i].substring(5);
}
else if (tokens[i].substring(0, 4) === 'url=') {
command.url = tokens[i].substring(4);
}
else if (tokens[i].substring(0, 7) === 'viewid=') {
command.viewid = tokens[i].substring(7);
}
else if (tokens[i].substring(0, 8) === 'nviewid=') {
command.nviewid = tokens[i].substring(8);
}
else if (tokens[i].substring(0, 7) === 'params=') {
command.params = tokens[i].substring(7).split(',');
}
else if (tokens[i].substring(0, 9) === 'renderid=') {
command.renderid = tokens[i].substring(9);
}
else if (tokens[i].substring(0, 12) === 'rendercount=') {
command.rendercount = parseInt(tokens[i].substring(12));
}
else if (tokens[i].startsWith('wid=')) {
command.wireId = this.getParameterValue(tokens[i]);
}
else if (tokens[i].substring(0, 6) === 'title=') {
command.title = tokens[i].substring(6);
}
else if (tokens[i].substring(0, 12) === 'dialogwidth=') {
command.dialogwidth = tokens[i].substring(12);
}
else if (tokens[i].substring(0, 13) === 'dialogheight=') {
command.dialogheight = tokens[i].substring(13);
}
else if (tokens[i].substring(0, 10) === 'rectangle=') {
command.rectangle = tokens[i].substring(10);
}
else if (tokens[i].substring(0, 12) === 'hiddenparts=') {
var hiddenparts = tokens[i].substring(12).split(',');
command.hiddenparts = [];
hiddenparts.forEach(function (item) {
command.hiddenparts.push(parseInt(item));
});
}
else if (tokens[i].startsWith('selectedparts=')) {
var selectedParts = tokens[i].substring(14).split(',');
```

```
command.selectedParts = [];
selectedParts.forEach(function (item) {
command.selectedParts.push(parseInt(item));
});
}
else if (tokens[i].startsWith('hash=')) {
command.hash = tokens[i].substring('hash='.length);
}
else if (tokens[i] === 'nopng') {
command.nopng = true;
}
else if (tokens[i].startsWith('masterpagecount='))
command.masterPageCount = parseInt(tokens[i].substring(16));
}
if (command.tileWidth && command.tileHeight && this._map._docLayer) {
var defaultZoom = this._map.options.zoom;
var scale = command.tileWidth / this._map._docLayer.options.tileWidthTwips;
// scale = 1.2 ^ (defaultZoom - zoom)
// zoom = defaultZoom -log(scale) / log(1.2)
command.zoom = Math.round(defaultZoom - Math.log(scale) / Math.log(1.2));
}
return command;
}
});
```