```
/**
 * @file plugin.js
 * Insert Inline plugin for Ckeditor.
 * Replace all <!-- INLINE:xxx;w=x;h=x --> tags with images (vice versa).
 */
(function() {
  var pluginRequires = [ 'fakeobjects' ];
  if (Drupal.ckeditor_ver == 3) {
    pluginRequires = [ 'fakeobjects', 'htmldataprocessor' ];
  }

  // Make the br tag to be self-closing tag.
  CKEDITOR.dtd.$empty['br '] = 1;

  // Initialize the plugin.
  CKEDITOR.plugins.add('insertinline', {
    icons: 'insertinline',
    init: function(editor) {

      var addInlineCssObj = CKEDITOR;
      if (Drupal.ckeditor_ver == 3) {
        addInlineCssObj = editor;
      }
      // Add the styles that renders our fake objects.
      addInlineCssObj.addCss(
        '.wysiwyg-inline-image' +
        '{' +
        'background-image: url(' + CKEDITOR.getUrl(this.path + 'icons/image.jpg') + ');' +
        'background-position: center center;' +
        'background-repeat: no-repeat;' +
        'background-color: LightSteelBlue;' +
        'display: block;' +
        'float: left;' +
        'margin: 1em 1em 0 0;' +
        'border: #333 1px dashed;' +
        '}'
      );

      editor.addCommand('insertInline', {
        exec : function(editor) {
          alert(Drupal.t('This is a pseudo plugin/button.\nIt does not insert an inline image.\n\nPlease use the INSERT button from the image
upload fieldset.'));
        }
      });
      editor.ui.addButton('insertinline', {
        label: 'Insert Inline Image',
        command: 'insertInline',
        icon: this.path + 'icons/insertinline.gif'
      });
```

```
  editor.config.contentsCss = this.path + 'css/insertinline.css';

// Run these after attaching ang detaching the ckeditor.
afterInit: function (editor) {

  var dataProcessor = editor.dataProcessor,

    // Ckeditor in this state is enabled.
    dataFilter = dataProcessor && dataProcessor.dataFilter,


    // Ckeditor in this state is disabled.
    htmlFilter = dataProcessor && dataProcessor.htmlFilter;

  var writer = editor.dataProcessor.writer;
  // The way to close self closing tags, like <br />.
  writer.selfClosingEnd = '>';

  /**
   * Clear paragraph that has no inline image inside.
   */
  jQuery('iframe.cke_wysiwyg_frame').contents().find("p:not(:has(img[class='wysiwyg-inline-image']))").css("clear", "right");

  // Editor status: 1
  // Executes when the ckeditor loads.
  if (dataFilter) {

    dataFilter.addRules({

      // Run this rule when it found any comment in the node body.
      comment: function(value) {
        // Perform this if the return value has ":"
        // We are sure that only inline comment has ":" in it.
        if (value.indexOf(":") > -1 && value.match(/INLINE:\d+;w=\d+;h=\d+/)) {
          // Get the right part of INLINE (124566;w=240;h240);
          var split_inline = value.split(":")[1];
          // Break into array.
          var params = split_inline.split(";");
          // The image file ID.
          var fid = params[0];
          // The image width.
          var w = params[1].substring(2);
          // The mage height.
          var h = params[2].substring(2);

          // Get the actual inline image using fid from INLINE comment.
          var ImgData = Drupal.cnngo_inline.getInlineImageDetail(fid, w, h);
          var inline_alt = jQuery.trim(ImgData.alt);
```

```
            // Load the itok and alt array.
            // Parse this according to ImgData.alt.
            var withItokParam = Drupal.cnngo_inline.getItok(inline_alt);
          }

          // Copied from ckeditor.js.
          var k = CKEDITOR.dtd,
            m = {
              br: 1, p: 1, div: 1, h1: 1,
              h2: 1, h3: 1, h4: 1, h5: 1,
              h6: 1, ul: 1, ol: 1, li: 1,
              pre: 1, dl: 1, blockquote: 1
            }, l = {
              br: 1, p: 1, div: 1,
              h1: 1, h2: 1, h3: 1,
              h4: 1, h5: 1, h6: 1
            }, n = CKEDITOR.tools.extend({}, k.$inline);

          // Fake element constructor.
          CKEDITOR.editor.prototype.createFakeParserInlineElement = function (a, b, c, d) {
            var e = this.lang.fakeobjects,
              e = e[c] || e.unknown,
              f;
            f = new CKEDITOR.htmlParser.basicWriter;
            a.writeHtml(f);
            f = f.getHtml();

            // This is the very important as "b" contains required attributes of inline physical image.
            b = {
              "data-cke-saved-src": ImgData.src,
              "src": withItokParam[0][0],
              alt: ImgData.alt,
              "width": w,
              "height": h,
              "class": 'wysiwyg-inline-image',
            };

            // Set the source of spacer image.
            b.src = withItokParam[0][0];
            // Get the "data-cke-real-element-type" attribute value.
            c && (b["data-cke-real-element-type"] = c);
            // Could be the actual image path or undefined.
            d && (b["data-cke-resizable"] = d,
            d = a.attributes, a = new i, c = d.width, d = d.height, void 0 != c && (a.rules.width = h(c)), void 0 != d && (a.rules.height =
h(d)), a.populate(b));

            return new CKEDITOR.htmlParser.element("img", b);
          };
```

```
            // Replace every inline comment it will found with physical image (inline).
            if (value.match(/INLINE:\d+;w=\d+;h=\d+/)) {
              var inlineImages = editor.createFakeParserInlineElement(new CKEDITOR.htmlParser.comment(value), 'wysiwyg-inline-image', 'img');
              return inlineImages;
            }

            // Return original value for non inline.
            else {
              return value;
            }
          }
        });
      }

      // Editor status: 0
      // Executes when in html source mode.
      if (htmlFilter) {
        //console.log("Enter CKEditor Source view mode: ");

        htmlFilter.addRules({
          // Loop through all elements
          elements: {

            // Perform this when it found image tag.
            img : function(element) {
              // Replace inline image with INLINE placeholder (comment).
              // Do this if the editor hit "Save" button.
              jQuery('#edit-submit').click(function () {
                return element;
              });

              // Return the actual element when detaching/attaching the "Ckeditor".
              if (element.attributes.class == 'wysiwyg-inline-image') {
                // Return the image alt value.
                return new CKEDITOR.htmlParser.comment(element.attributes.alt);
              }

              // Return br comment.
              if (element.attributes.class == 'wysiwyg-addbr') {
                // Make the br tag to be self-closing tag.
                return new CKEDITOR.htmlParser.element('br ', {'class': 'clear-both'});
              }
            }
          }
        });
      }
    }
  });
})();
```