Ingress Intel Total Conversion Documentation

Release official-docs

Jon Atkins et al. (IITC), Filip Wieland et al. (IITC Docs)

Aug 07, 2023

Contents

IITC is a userscript for the Ingress Intel map that makes using it bearable. Itself it exposes a plugin API which is rich and thoroughly undocumented. This project aims to fix the documentation issues, but is not itself affiliated with Niantic Labs.

Contents:

CHAPTER 1

The Plugin API

IITC has a plugin API that can be used by other userscripts to register as plugins. It also imports jQuery UI to serve as a widget toolkit, however the widgets are styled to match IITC's look.

1.1 Plugin lifecycle

Most plugins follow this plan:

1. Create a wrapper function
2. Add some extra variables to make the plugin known to IITC. Create some globals if they are not set yet.
3. Put actual plugin code inside the wrapper function, with a setup function inside
4. Append the setup function to the `window.bootPlugins` list
5. Stringify the wrapper function and inject it into the page context as an IIFE

With first-party plugins, enforcement of this plan is done by using special build-time syntax (yes, IITC is using a custom build/macro system).

1.2 Your First Plugin

Here is a simple "Hello, World" plugin that illustrates how IITC plugins work:

```
// ==UserScript==
// @id hello-iitc
// @name IITC Plugin: Hello World
// @category Misc
// @version 0.0.1
// @namespace https://tempuri.org/iitc/hello
// @description Hello, World plugin for IITC
// @include https://intel.ingress.com/intel*
```

(continues on next page)

(continued from previous page)

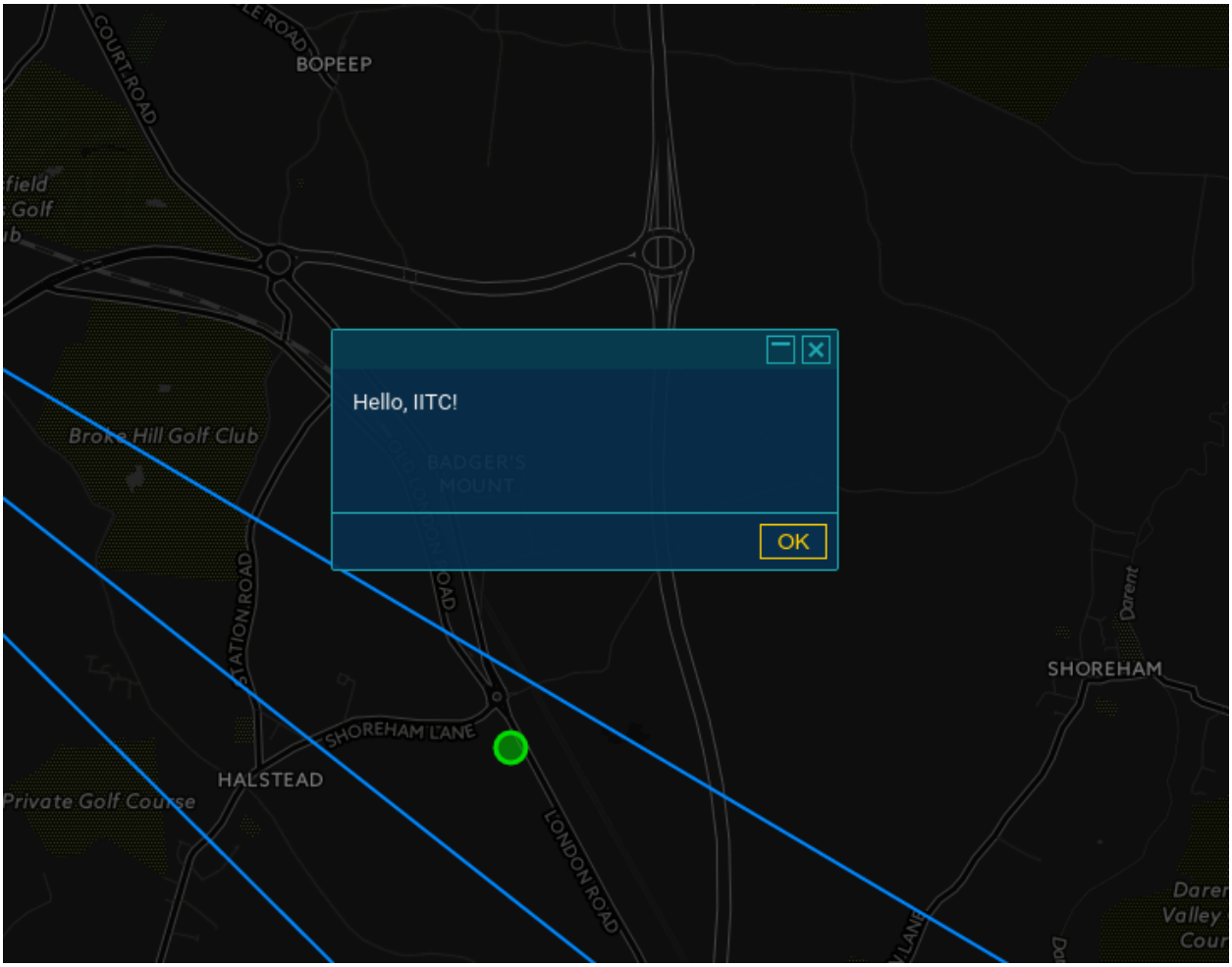
```
// @match https://intel.ingress.com/intel*
// @grant none
// ==/UserScript==

// Wrapper function that will be stringified and injected
// into the document. Because of this, normal closure rules
// do not apply here.
function wrapper(plugin_info) {
  // Make sure that window.plugin exists. IITC defines it as a no-op function,
  // and other plugins assume the same.
  if(typeof window.plugin !== 'function') window.plugin = function() {};

  // Name of the IITC build for first-party plugins
  plugin_info.buildName = 'hello';

  // Datetime-derived version of the plugin
  plugin_info.dateTimeVersion = '20150829103500';

  // ID/name of the plugin
  plugin_info.pluginId = 'hello';

  // The entry point for this plugin.
  function setup() {
    alert('Hello, IITC!');
  }

  // Add an info property for IITC's plugin system
  setup.info = plugin_info;

  // Make sure window.bootPlugins exists and is an array
  if (!window.bootPlugins) window.bootPlugins = [];
  // Add our startup hook
  window.bootPlugins.push(setup);
  // If IITC has already booted, immediately run the 'setup' function
  if (window.iitcLoaded && typeof setup === 'function') setup();
}

// Create a script element to hold our content script
var script = document.createElement('script');
var info = {};

// GM_info is defined by the assorted monkey-themed browser extensions
// and holds information parsed from the script header.
if (typeof GM_info !== 'undefined' && GM_info && GM_info.script) {
  info.script = {
    version: GM_info.script.version,
    name: GM_info.script.name,
    description: GM_info.script.description
  };
}

// Create a text node and our IIFE inside of it
var textContent = document.createTextNode('('+ wrapper +')('+ JSON.stringify(info) +')
↪');
// Add some content to the script element
script.appendChild(textContent);
// Finally, inject it... wherever.
```

(continues on next page)

(continued from previous page)

```
(document.body || document.head || document.documentElement).appendChild(script);
```

If all goes well, after you install the userscript and refresh intel, you should see the following:



Since IITC uses jQuery UI, and jQuery UI in turn patches `alert()`, no browser alert is expected.

Here's how it works:

1.2.1 The UserScript header

IITC plugins are themselves user scripts, which means they have to follow userscript conventions, and share some gotchas you need to keep in mind.

```
// ==UserScript==
// @id hello-iitc
// @name IITC Plugin: Hello World
// @category Misc
// @version 0.0.1
// @namespace https://tempuri.org/iitc/hello
// @description Hello, World plugin for IITC
// @include https://intel.ingress.com/intel*
// @match https://intel.ingress.com/intel*
```

(continues on next page)

(continued from previous page)

```
// @grant none
// ==/UserScript==
```

All user scripts that GreaseMonkey (for Firefox) and Tampermonkey (for Chrome) recognise should have this header. This allows them to extract metadata about the script, such as which sites it should run on, and the name and description of the script to be displayed on the script list. It's important to note the `@grant none` line; otherwise, Tampermonkey will complain about the script not specifying any grants. `@grant` can be used to gain access to some special userscript APIs - see @grant on GreaseSpot. This header is also parsed by the relevant platform's monkey and provided to your script as `GM_info`. For more info about the metadata block in general, see Metadata Block on GreaseSpot.

1.2.2 The wrapper function

```
function wrapper(plugin_info) { /* ... */ }
```

Userscripts execute in a separate context from the page - ie. the global object is not the same as `window`, however both `window` and `document` are accessible. The wrapper function exists to contain a script that will be injected into the page. This necessarily means the function will *not* close over any variables defined outside of it, as it will be stringified and added to the page's DOM as an IIFE.

Also note the `plugin_info` parameter - will be needed later.

1.2.3 The plugin framework

```
if(typeof window.plugin !== 'function') window.plugin = function() {};
```

The run order of userscripts is not guaranteed, so our script can be loaded before IITC gets around to creating the plugin framework. Not sure why this is a no-op function though.

1.2.4 Plugin info

```
// Name of the IITC build for first-party plugins
plugin_info.buildName = 'hello';

// Datetime-derived version of the plugin
plugin_info.dateTimeVersion = '20150829103500';

// ID/name of the plugin
plugin_info.pluginId = 'hello';
```

This is mostly for first-party plugins that are built with the same tool as IITC itself, but is included here for completeness. The standard plugin header includes this warning:

```
//PLUGIN AUTHORS: writing a plugin outside of the IITC build environment? if so,␣
↪delete these lines!!
//(leaving them in place might break the 'About IITC' page or break update checks)
```

Your mileage may vary.

1.2.5 The entry point

```
function setup() {
  alert('Hello, IITC!');
}
```

This will be our entry point to the plugin, and will be called by IITC when it finishes loading (or, we will call it yourself if IITC has already loaded).

1.2.6 Plugin properties

```
setup.info = plugin_info;
```

IITC expects the plugin entry point to also include some extra information about the plugin itself. Here, we use the plugin_info object for this.

1.2.7 Running your plugin

```
// Make sure window.bootPlugins exists and is an array
if (!window.bootPlugins) window.bootPlugins = [];
// Add our startup hook
window.bootPlugins.push(setup);
// If IITC has already booted, immediately run the 'setup' function
if (window.iitcLoaded && typeof setup === 'function') setup();
```

Again, since there are no guarantees about the order userscripts are run in, we need to make sure *bootPlugins* exists. We then add our entry point to that array - in case IITC has not finished loading yet, it will be called after it will. If it has, we need to call the entry point itself. Testing for whether the setup function is indeed a function will always be true so it can be omitted, but is included in the standard IITC plugin body.

1.2.8 Plugin info

Meanwhile, back in userscript land...

```
var info = {};
// GM_info is defined by the assorted monkey-themed browser extensions
// and holds information parsed from the script header.
if (typeof GM_info !== 'undefined' && GM_info && GM_info.script) {
  info.script = {
    version: GM_info.script.version,
    name: GM_info.script.name,
    description: GM_info.script.description
  };
}
```

GM_info contains information about the userscript itself parsed from the header. You can possibly just do `info = GM_info`, however that will also pass in a bunch of other things to IITC.

1.2.9 Injecting the script

```
// Create a script element to hold our content script
var script = document.createElement('script');
// Create a text node and our IIFE inside of it
var textContent = document.createTextNode('('+ wrapper +')('+ JSON.stringify(info) +')
↪');
// Add some content to the script element
script.appendChild(textContent);
// Finally, inject it... wherever.
(document.body || document.head || document.documentElement).appendChild(script);
```

Finally, to inject our script into the page, we need to create a new script element, containing a text node containing our wrapper function, and append it to the body (or head, or the document itself if necessary). Note that we have to call `JSON.stringify` on the info object to pass it to the wrapper function - again, this is due to the separate-context mechanic of userscripts.

CHAPTER 2

Core

IITC defines some top-level variables in *main.js* that its internal modules and first-party plugins use for configuration

2.1 Constants

`window.`**`PLAYER`**
: Defined by stock. Static (needs page reload to update). Stores information about the current player:

  - `ap`: AP the player has (string)
  - `available_invites`: Number of invitations this player can send
  - `energy`: XM the player currently holds
  - `min_ap_for_current_level`: AP required for the player's level (used for level progress)
  - `min_ap_for_next_level`: AP required for the next level (used for level progress)
  - `nickname`: The actual agent name
  - `team`: Player faction. Can be "ENLIGHTENED" or "RESISTANCE"
  - `verified_level`: Current player level

  IITC adds a few things in **:function:`~window.setupPlayerStat()`**:

  - `nickMatcher`: RegExp used to match the player's agent name in chat
  - `level`: Backwards compatibility, same as `verified_level`.

`window.`**`REFRESH`**
: Controls how often the map should refresh, in seconds, default 30.

`window.`**`ZOOM_LEVEL_ADJ`**
: Controls the extra refresh delay per zoom level, in seconds, default 5.

`window.`**`ON_MOVE_REFRESH`**
: Wait this long before refreshing the view after the map has been moved, in seconds, default 2.5

window.**MINIMUM_OVERRIDE_REFRESH**
: "limit on refresh time since previous refresh, limiting repeated move refresh rate" (?), in seconds, default 10

window.**REFRESH_GAME_SCORE**
: Controls how long to wait between refreshing the global score, in seconds, default 15*60 (15 mins)

window.**MAX_IDLE_TIME**
: Controls how long, at most, can the map be inactive before refreshing, in secods, default 15*60 (15 mins)

window.**HIDDEN_SCROLLBAR_ASSUMED_WIDTH**
: How much space to leave for scrollbars, in pixels, default 20.

window.**SIDEBAR_WIDTH**
: How wide should the sidebar be, in pixels, default 300.

window.**CHAT_REQUEST_SCROLL_TOP**
: Controls requesting chat data if chat is expanded based on the pixel distance from the line currently in view and the top of history, in pixels, default 200

window.**CHAT_SHRINKED**
: Controls height of chat when chat is collapsed, in pixels, default 60

window.**COLOR_SELECTED_PORTAL**
: What colour should the selected portal be, string(css hex code), default '#f0f' (hot pink)

window.**COLORS**

```
['#FF6600', '#0088FF', '#03DC03']; // none, res, enl
```

: Colour values for teams used in portals, player names, etc.

window.**COLORS_LVL**

```
['#000', '#FECE5A', '#FFA630', '#FF7315', '#E40000', '#FD2992', '#EB26CD', '
↪#C124E0', '#9627F4']
```

: Colour values for levels, consistent with Ingress, with index 0 being white for neutral portals.

window.**COLORS_MOD**

```
{VERY_RARE: '#b08cff', RARE: '#73a8ff', COMMON: '#8cffbf'}
```

: Colour values for displaying mods, consistent with Ingress. Very Rare also used for AXA shields and Ultra Links.

window.**ACCESS_INDICATOR_COLOR**
: What colour should the hacking range circle be (the small circle that appears around a selected portal, marking a ~40 metre radius), string(css colour value), default 'orange'

window.**RANGE_INDICATOR_COLOR**
: What colour should the linkable range circle be, string(css colour value), default 'red'

window.**MIN_ZOOM**
: "min zoom for intel map - should match that used by stock intel", in (leaflet zoom levels?), default 3

window.**NOMINATIM**

```
'//nominatim.openstreetmap.org/search?format=json&polygon_geojson=1&q='
```

URL to call the Nominatim geocoder service, string.

window.**RESO_NRG**
Resonator energy per level, 1-based array, XM

window.**HACK_RANGE**
Maximum radius around a portal from which the portal is hackable, metres.

window.**OCTANTS**

```
['E', 'NE', 'N', 'NW', 'W', 'SW', 'S', 'SE']
```

Resonator octant cardinal directions

window.**OCTANT_ARROW**

```
['→', '', '↑', '', '←', '', '↓', '']
```

Resonator octant arrows

window.**DESTROY_RESONATOR**
window.**DESTROY_LINK**
window.**DESTROY_FIELD**
window.**CAPTURE_PORTAL**
window.**DEPLOY_RESONATOR**
window.**COMPLETION_BONUS**
window.**UPGRADE_ANOTHERS_RESONATOR**
AP values for performing in-game actions. *COMPLETION_BONUS*: refers to the extra AP for deploying the last resonator on a portal.

window.**MAX_PORTAL_LEVEL**
Maximum portal level.

window.**MAX_RESO_PER_PLAYER**

```
[0, 8, 4, 4, 4, 2, 2, 1, 1]
```

How many resonators of a given level can one deploy; 1-based array where the index is the resonator level.

window.**TEAM_NONE**
window.**TEAM_RES**
window.**TEAM_ENL**
Faction. NONE is 0, RES is 1, ENL is 2.

window.**TEAM_TO_CSS**

```
['none', 'res', 'enl']
```

Maps team to its CSS class. Presumably to be used like `TEAM_TO_CSS[TEAM_ENL]`.

2.2 Variables

`window.`**`refreshTimeout`**
: Stores the id of the timeout that kicks off the next refresh (ie value returned by `setTimeout()`)

`window.`**`urlPortal`**
: Portal GUID if the original URL had it.

`window.`**`urlPortalLL`**
: Portal lng/lat if the orignial URL had it.

`window.`**`selectedPortal`**
: Stores the ID of the selected portal, or is `null` if there is none.

`window.`**`portalRangeIndicator`**
: Reference to the linking range indicator of the selected portal. This is a Leaflet layer.

`window.`**`portalAccessIndicator`**
: Reference to the hacking range indicator of the selected portal. This is a Leaflet layer.

`window.`**`portals`**
`window.`**`links`**
`window.`**`fields`**
: References to Leaflet objects for portals, links, and fields. These are indexed by the entity ID in an object, ie. `{ id1: feature1, ...}`

  Note: Although these will be Leaflet objects, not all may be added to the map if render limits are reached.

`window.`**`overlayStatus`**
: An object, where the keys are layer names and their values are bools true if the layer is enabled. Should mirror the layer selector UI.

  Note: The variable comment states that "you should use **:function:'~window.isLayerGroupDisplayed(name)'** to check the [layer] status"

`window.`**`isLayerGroupDisplayed`**`(`*name*`)`
: Read layerGroup status from *`overlayStatus`* if it was added to map, read from cookie if it has not added to map yet. `return 'defaultDisplay'` if both `overlayStatus` and cookie didn't have the record

`window.`**`plugin`**`()`
: A noop function/namespace/"plugin framework".

`window.`**`bootPlugins`**
: A list of hooks that should be called after IITC has finished booting. Mostly used to initialise plugins. **Note:** These will not run if some blacklisted plugins are detected.

CHAPTER 3

Boot

3.1 Functions

These were found in *code/boot.js*, and govern the initialisation process of IITC. The header for the file says:

```
/// SETUP //////////////////////////////////////////////////////
// these functions set up specific areas after the boot function
// created a basic framework. All of these functions should only ever
// be run once.
```

`window.`**`setupLargeImagePreview`**`()`
Sets up event listeners for large portal image view. This is the dialogue you get when you click on the portal photo in the sidebar.

`window.`**`setupLayerChooserSelectOne`**`()`

Adds listeners to the layer chooser such that a long press hides all custom layers except the long pressed one.

Actually, it seems you can also use meta-click, ctrl-click, shift-click or alt-click to trigger this behaviour.

`window.`**`setupLayerChooserStatusRecorder`**`()`
Sets up the *overlayStatus* dict from what layers are visible on the map, and sets up event listeners that update this dict based on layers being hidden and removed from the Leaflet map.

Note: This does not actually modify the dict directly, but rather it uses the **:function:'~window.updateDisplayedLayerGroup(name, display)'** function.

`window.`**`updateDisplayedLayerGroup`**`(`*name*, *display*`)`
Update layerGroups display status to window.overlayStatus and localStorage 'ingress.intelmap.layergroupdisplayed'

`window.`**`layerChooserSetDisabledStates`**`()`
Enables/disables portal layers in the selector based on zoom level.

`window.`**`setupStyles`**`()`
Adds IITC's CSS to `<head>`.

createDefaultBaseMapLayers()
This function is private. It will not be accessible from the console or other scripts.

Sets up the default basemap tiles: MapQuest, CartoDB, Google Ingress, Google Roads, Google Satellite, Google Hybrid and Google Terrain.

window.**setupMap**()
Sets up the Leaflet map. Note that there is a TODO entry there to move IITC's DOM into Leaflet control areas (it is currently just overlaying the map div completely).

This also sets up a few interesting event listeners. First, when the user moves the map, all requests that go through `window.requests` are aborted, and the refresh timeout is cleared. Second, it calls **:function:'~window.layerChooserSetDisabledStates()'** on zoom end. Finally, it sets up the map data requester and starts refreshing.

window.**setMapBaseLayer**()
Adds a basemap (tile layer) to the Leaflet map. As documented in source, this is done separately from **:function:'~window.setupMap()'** to allow plugins to add their own tile layers (ie. Stamen tiles, OSM tiles).

window.**setupPlayerStat**()
Renders player details into the website. Since the player info is included as inline script in the original site, the data is static and cannot be updated. for historical reasons IITC expects *PLAYER*.''level'' to contain the current player level.

window.**setupSidebarToggle**()
Sets up the sidebar toggle button.

window.**setupTooltips**()
Sets up the tooltips and the `window.tooltipClearerHasBeenSetup` flag.

window.**setupTaphold**()
Container for the Taphold jQuery plugin.

window.**setupQRLoadLib**()
Container for the qrcode jQuery plugin.

window.**setupLayerChooserApi**()
Sets up the layer chooser API. In particular, it helps unify the HTML layer chooser and the IITCm Android app layer chooser (which is a native component).

window.layerChooser.**getLayers**()

Returns Layer settings grouped by `baseLayers` and `overlayLayers`

Gets the available layers. Both layer arrays contain objects like `{ active: bool, layerId: int, name: string }`

window.layerChooser.**showLayer**(*id*[, *show*])

Arguments

- **id** (*int*) – The layer ID
- **show** (*bool*) – Pass `false` to hide the layer

Shows or hides the basemap or overlay layer with id `id`.

window.**boot**()
Main boot function. This also boots the plugins using the plugin API. It also maintains a blacklist of plugins that, if present, will prevent a normal startup of the plugin system (ie. none of `window.bootPlugins` functions will be called).

CHAPTER 4

Indices and tables

- genindex
- search

Index

C

W