

# D&B Visitor Intelligence Form Fill

## WHAT IS VISITOR INTELLIGENCE FORM FILL?

Dun & Bradstreet's Visitor Intelligence Form Fill helps customers increase form completion rates, streamline the submission process, and improve accuracy. This data comes from the full universe of company information tied to the D-U-N-S® number and associated family tree linkages and can include attributes that would be impossible to obtain via regular user interactions.

The underlying APIs are triggered by multiple different form inputs including company name and e-mail address. The output of the form can be sent into any customer-controlled platform. With no major changes required to the form itself, this should speed time-to-market and minimize customer upkeep requirements.

## PREREQUISITES

The following is required in order to use Visitor Intelligence Form Fill:

- The web form(s) in question is a standard HTML <form> object with attendant <input> fields
- It is hosted in a location where additional JavaScript code can be placed
- At least three specific inputs are defined (either visible or hidden): e-mail address, company name, and country

The following non-standard cases can potentially be supported via customization:

- Forms that are built entirely within Flash or other technologies
- A site that is using an overarching JavaScript library which defines a proprietary data layer outside of the standard DOM (for example, ReactJS)
- Progressive or dynamic forms whose layout and contents may change between visitors, over time, or both

**Please note:** The code examples provided within this document are for reference purposes only and are not to be used as-is.

## OVERVIEW

Visitor Intelligence Form Fill is a self-contained solution that leverages Dun & Bradstreet's industry-leading global database of commercial entities to return pertinent data back to your form, improving completion rates and the accuracy of the submitted data. There are three major components to the JavaScript code provided:

- A JavaScript library file that contains the client-side code, plus a link to our privacy disclosures and opt-out mechanism

```
<!--START DNB VIFF JS-->
<!--For opt-out information, please visit:
https://d41.co/-->
<script type="text/javascript" src="//cdn-0.
d41.co/tags/ff-2.min.js" charset="utf-8"></
script>
```

- A mapping between your form input fields and our data elements, along with other preferences for how the application behaves

```
<script type="text/javascript">
if (window.ActiveXObject) {
    window.ActiveXObject = null;
}
var dpa = new Fill.LeadFormApp({
    visitorIntelligenceApiKey: "@API_KEY@",
    defaultCompanyCountry: "US",
    leadFormName: "@FORM_NAME@",
    companyCountrySearchFieldName: "@COUNTRY@",
    contactEmailSearchFieldName: "@EMAIL@",
    companyNameSearchFieldName: "@COMPANY@",
    useLIDropdowns: true,
    visitorIDEnabled: false,
    initialFocusFieldName: "",
    dunsFieldName: "@DUNS@",
    companyNameFieldName: "@COMPANY@",
    address1FieldName: "@ADDRESS1@",
    address2FieldName: "@ADDRESS2@",
    cityFieldName: "@CITY@",
    stateFieldName: "@STATE@",
    postalFieldName: "@POSTALCODE@",
    countryFieldName: "@COUNTRY@",
    naicsCodeFieldName: "@NAICS_CODE@",
    naicsDescriptionFieldName: "@NAICS_DESCRIPTION@",
    sicCodeFieldName: "@SIC_CODE@",
    sicDescriptionFieldName: "@SIC_DESCRIPTION@",
    revenueFieldName: "@REVENUE@",
    employeeSiteCountFieldName: "@EMPLOYEES@",
    firstNameFieldName: "@FIRST_NAME@",
    lastNameFieldName: "@LAST_NAME@",
    vanityTitleFieldName: "@VANITY_TITLE@",
    globalUltimateDunsFieldName: "@GU_DUNS@",
    globalUltimatePrimaryNameFieldName: "@GU_NAME@",
    domesticUltimateDunsFieldName: "@DU_DUNS@",
    domesticUltimatePrimaryNameFieldName: "@DU_NAME@",
    parentDunsFieldName: "@PARENT_DUNS@",
    parentPrimaryNameFieldName: "@PARENT_NAME@",
    globalUltimateFamilyTreeMembersCountFieldName: "@TREE_SIZE@",
    phoneFieldName: "@COMPANY_PHONE@",
    telephoneNumberFieldName: "@PERSONAL_PHONE@",
    domainFieldName: "@DOMAIN@",
    tradeStyleNameFieldName: "@TRADE_NAME@",
    currencyFieldName: "@CURRENCY@"
});
```

- A function call that imposes the relevant listeners upon the mapped input fields, at which point our solution can be considered fully active on that web page

```
dpa.attach();
</script>
<!-- END DNB VIFF JS-->
```

Generally, only the second item – application preferences and data mapping – will require your input.

The underlying matching methodology operates in two pieces: form pre-population and user-directed searches. Form pre-population is disabled by default due to its potential impact on the overall user experience; please see the appendix for more information on how to enable it. User-directed searches rely on either of two pieces of information:

- E-mail address: This will trigger two separate searches, one for the precise e-mail address provided and another for the domain portion only. The former takes precedence and is the only situation in which we are able to return individual-level data such as name, title, and personal phone number. The latter will be utilized as the secondary result and will return the company that is associated with that domain on a global basis.
- Company name typeahead: This initiates a low-latency connection to the server, producing a list of the ten most likely candidates given the current input. The results list will refresh dynamically as additional letters are added. The country input will determine the scope of the results returned.

Either of the above can be disabled, in which case that input field will behave as normal and will not lead to any data being injected. More information on this can be found in the appendix.

Keep in mind that the layout of the form itself will have a substantial role in how the user interacts with the application and what search method is likely to be used. Best practice is to begin with e-mail input, followed by country and company name: that ensures the flow is from the most precise option to the least, minimizing the likelihood of user input being overwritten.

## MAPPING

An example input element is below. Normally, both the “id” and “name” attributes will exist. Visitor Intelligence Form Fill requires the use of names. Support for IDs is coming in a future update, but will still require that those values be unique to ensure proper functioning.

```
<input id="field3" name="jobTitle"
type="text" value="" class="field-size-
top-large">
```

If your IDs or names are static (do not change over time), then the mapping process is simple: the right-hand side of the mapping should be updated to the name of the appropriate input in your form. Using the example above for job title, the resulting line in the final JavaScript would be:

```
vanityTitleFieldName: "jobTitle",
```

The surrounding quotes should be left intact. In addition, please confirm that they are “straight” quotation marks and not the “fancy” version that is used by default in Microsoft Office.

Unused mappings can be commented out of the code altogether, or that entire line removed:

```
//naicsDescriptionFieldName: "@NAICS_
DESCRIPTION@",
```

As mentioned in the pre-requisites, there are three special mappings that must exist for the application to operate correctly. These represent the inputs used for initiating searches. Even if the relevant search functionality has been disabled, please continue to map these, even if it is to hidden fields that have no bearing on the user experience. As named in the mapping array, they are:

- companyCountrySearchFieldName
- contactEmailSearchFieldName
- companyNameSearchFieldName

The remaining input field mappings are all intended to be outputs where the relevant data will be injected if the application has found a successful match and there is applicable information on file.

There is an equivalent output field mapping for each of the three input fields. In most cases, the same mapping will apply to both. However, this does allow you to split output from input in use cases such as fraud detection or user verification: all outputs from Visitor Intelligence Form Fill are sent to hidden fields so that the website owner can compare against what the user provided after-the-fact.

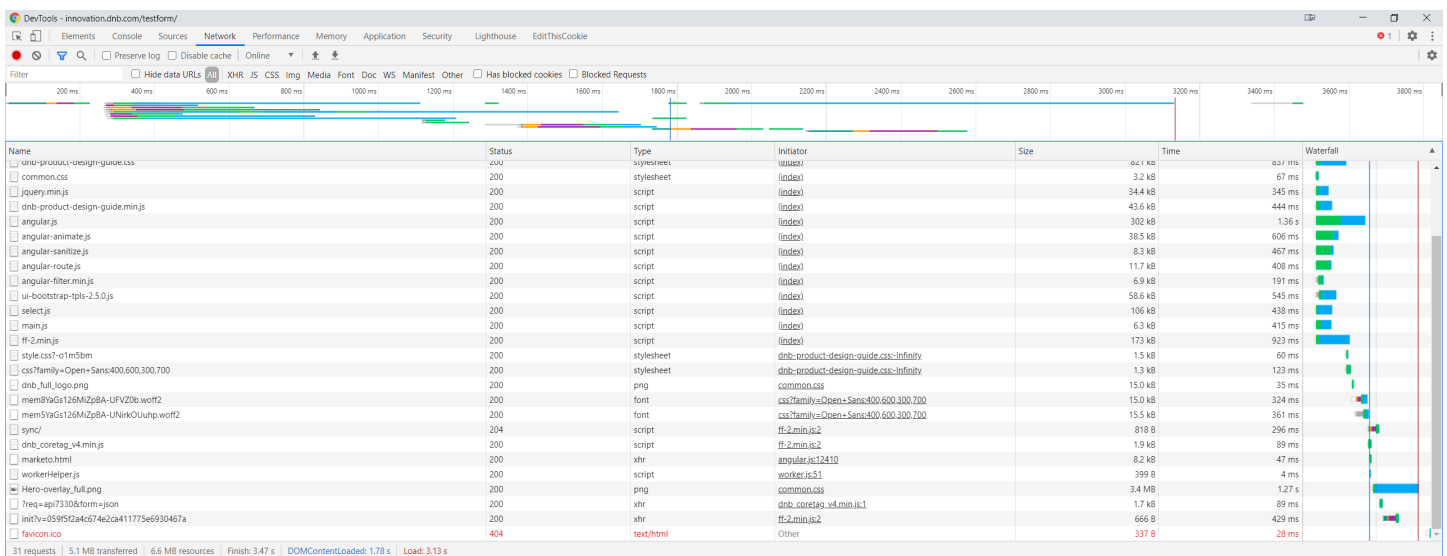
## FINDING INPUT FIELDS

There are multiple ways to obtain the proper mapping for each of your input fields. The most common methods are discussed below, with the assumption that the reference remains static.

If you are not certain that yours are static, we recommend going through the process below a few times, refreshing the page between each one. If the output is not consistent between page refreshes, please refer to the “Dynamic Mapping” section.

Pressing F12 should open the Developer Tools window for your browser. This is a screen that lays bare most of the underpinnings of the website and is a necessary part of this process. There are a few panes running across the top of the window, three of which will be used within this documentation:

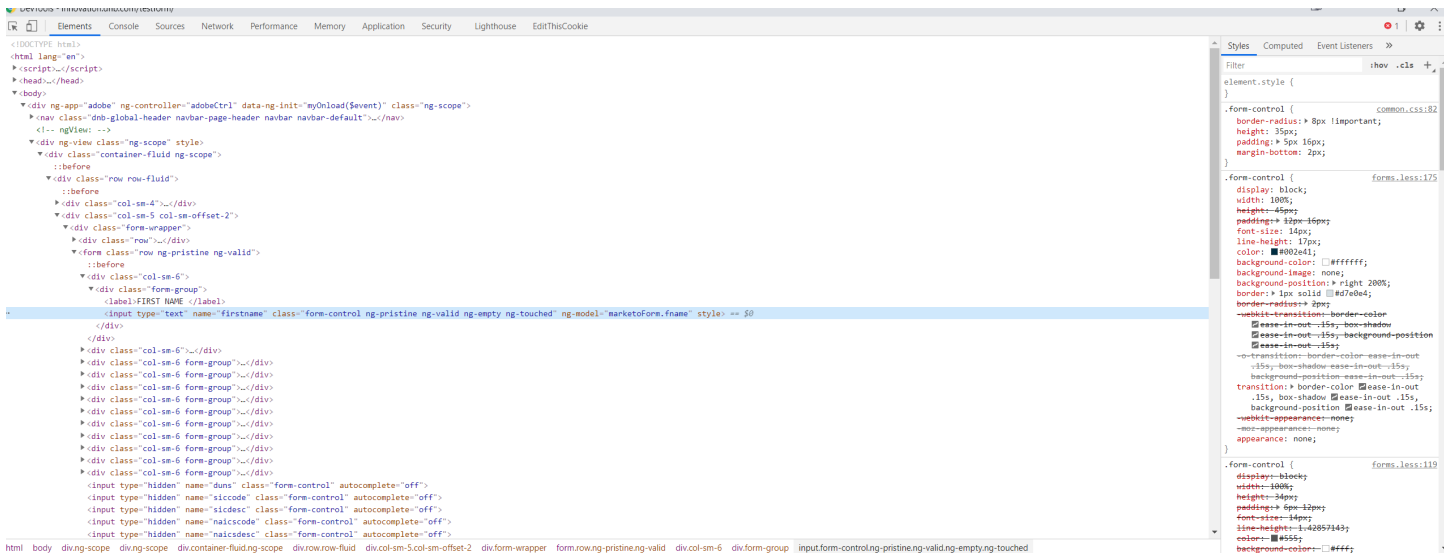
- **ELEMENTS:** An expanded view of the underlying source code for the site
- **CONSOLE:** The browser console, through which JavaScript can be run and the output displayed; this component is also the default option for the bottom module of the Developer Tools window, meaning this remains accessible when viewing the other panes
- **NETWORK:** A complete accounting of every network call that has been incurred while on the current page



If your form was created in Marketo, Eloqua, or a similar platform, the name of each input was likely defined within that interface.

## INSPECT

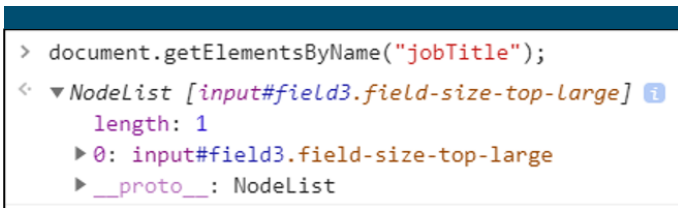
In all modern browsers, right-clicking on a form input field – or any other component of a web page – should bring up a context menu that includes an “Inspect” option. Doing so will bring up the Elements pane with that element in focus. From there, you should be able to copy the “name” value and use that for the mapping.



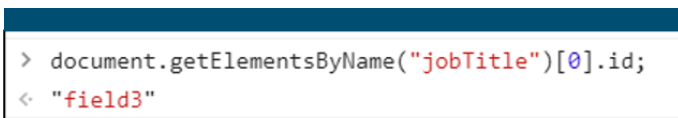
## document.getElementsByName()

This JavaScript method will return a list of objects on the page that match the name provided. As names are not intended to be unique, the expected output is an array listing out all items that match your input.

This example console command will return an array of all elements with the name “jobTitle”.



For verification, you can query directly for the “id” of any item in the nodelist (keeping in mind that JavaScript arrays start counting from zero):



## document.getElementById()

This is the more precise option and returns the object that matches the ID provided. IDs are intended to be unique, so will only return the first matching result found.



## Country Filtering on Typeahead

The typeahead search capability currently allows for the use of a country filter to restrict the set of returned results. This can be done in one of two ways:

- Via a user-selectable input field which is then mapped to “companyCountrySearchFieldName”
- As a hard-coded list in the code itself, which the user will not be able to override

The application currently accepts country inputs via three different standards: two-character ISO country code (“US”), three-character ISO country code (“USA”), and plaintext (“United States”).

Setting the appropriate country format or using a hard-coded country list can both be done via additional parameters in the JavaScript.

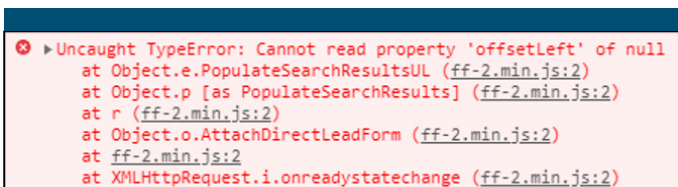
## COMMON ERRORS

Once the mapping has been updated in the code and placed on the page, you will be able to interact with the active form search fields to trigger the expected functionality. Installation directly in the site code or injection through a tag management system are both fine.

Keep in mind JavaScript is case-sensitive! Mapping the input field “jobTitle” is not the same thing as “jobtitle”.

### Country Error: Uncaught TypeError: Cannot read property 'offsetLeft' of null

This will be logged to the console as an error message in red text.

A screenshot of a browser's developer console showing a red error message. The message is: "Uncaught TypeError: Cannot read property 'offsetLeft' of null". Below the message, the stack trace is visible, listing several lines of code with file names and line numbers, all pointing to "ff-2.min.js:2". The error message and stack trace are displayed in red text on a light background.

```
✖ Uncaught TypeError: Cannot read property 'offsetLeft' of null
    at Object.e.PopulateSearchResultsUL (ff-2.min.js:2)
    at Object.p [as PopulateSearchResults] (ff-2.min.js:2)
    at r (ff-2.min.js:2)
    at Object.o.AttachDirectLeadForm (ff-2.min.js:2)
    at ff-2.min.js:2
    at XMLHttpRequest.i.onreadystatechange (ff-2.min.js:2)
```

The form will continue to function, but no input will appear from Visitor Intelligence Form Fill. If you have checked your mappings and don't see a problem, this is most likely due to a timing issue: the form object itself or at least one of the three mandatory input fields did not exist at the same time that the Visitor Intelligence Form Fill code was loaded.

You can work around this easily by decoupling the listener attachment command from the rest of the code. Please refer to the appropriate section in Advanced Techniques for more information.

### Network Error: 401 HTTP status code

The application will also appear inactive here, but in lieu of a console error logged, the relevant networks calls being made to the application will appear with a status code of 401.

In some cases, the application may appear to mysteriously stop responding after some time, with all subsequent requests changing over to a 401 status code.

If the application was working at any point prior, please also provide your public (externally-visible) IP address. You can obtain that by typing “what is my ip address” into any search engine.

## ADVANCED TECHNOLOGIES

The following information is intended for users with a web development background.

### Decoupling dpa.attach();

The third section of the JavaScript code calls “dpa.attach();” to enable the listeners, after which the application can be considered fully live. By default, this is included in the same block of JavaScript as the mapping array itself.

The mapping does not truly apply – more specifically, the application does not check for the input fields listed in the mapping – until that call is invoked. The likely scenarios in which this would be necessary are:

- The “offsetLeft” console error mentioned earlier
- A progressive or dynamic form that loads one or more mandatory input fields only if certain criteria are met

The solution is to invoke “dpa.attach();” at a later point, ideally off of a trigger such as the successful loading of the form itself.

## MULTIPLE FORMS

Visitor Intelligence Form Fill is designed to work with multiple forms on the same page. The “dpa” object as defined in the JavaScript is not set in stone, and you can easily create multiple concurrent copies of the application by defining each form as a separate object (dpa1, dpa2, etc) and then running the equivalent .attach method against each of them when needed (dpa1.attach, dpa2.attach, etc).

## DYNAMIC MAPPING

If your input field names are not static, there are various ways to programmatically obtain the proper mapping. The only requirement is that there be something static associated with the input fields, such as a common class. (Put another way, you must be able to narrow down to the appropriate element in the console via some combination of JavaScript methods.)

The solution is to write a separate function that can find the necessary name given other inputs. The necessary mapping table can then be constructed by iterating over that function for each relevant field.

An example is below that is based on the following assumptions:

- All inputs are part of the class “input”
- The “name” component to each input is static, although the “id” is not

```
getInput = function(className, tag='input'){
    var getInputByClass = document.getElementsByClassName(className)[0];
    var input = getInputByClass.
    getElementsByTagName(tag)[0];
    return input.name;
};
```

This can be followed by a separate set of code that runs the above function for each relevant input field and assigns them to a variable name. The variables are then used in the main mapping within the body of the Visitor Intelligence Form Fill code, allowing for that entire block of code (the getInput function and associated variable mapping) to be reused across your site.

## APPENDIX

PARAMETER NAME	DEFAULT	DATA TYPE	DESCRIPTION
leadFormName*		string	The name of the form. If left blank, it will attach itself to the first form found on the page.
contactEmailSearchFieldName*		string	The name of the e-mail field.
companyNameSearchFieldName*		string	The name of the company name field.
companyCountrySearchFieldName*		string	The name of the country field.
visitorIDEnabled	TRUE	boolean	If enabled Visitor Intelligence will pre-populate form if a match is found on IP lookup and cookie matching.
visitorIDSyncEnabled	TRUE	boolean	If enabled the /sync event will occur (disable to preclude the use of partner cookies in matching).
companySearchEnabled	TRUE	boolean	If enabled typeahead searching will be performed on company field.
contactSearchEnabled	TRUE	boolean	If enabled e-mail search will be performed on email field.
liSelectedClass		string	Allows user to specify a css class to style which option from dropdown is selected. If left blank it will use the default style.
defaultCompanyCountry		string	Sets the default ISO-2 country code.
companyCountryFilter	[]	array	Will filter out country options on a SELECT tag to only include countries in the given array.
displayHiddenFields	FALSE	boolean	Will display hidden fields on form. This should only be used for testing and debugging.
clearCompanyAfterCountryChange	TRUE	boolean	If set to true, will clear the company data after country field value changes.
searchCompanyAfterCountryChange	TRUE	boolean	If set to true, will perform company name search after country field changes.
clearCompanyAfterTypeaheadChange	FALSE	boolean	If set to true, will clear the company data after company name value changes.
globalCountrySearch	FALSE	boolean	Whether the system should always search globally for typeahead (ignoring input country filters).
countryValueType		string	Eligible values are "iso2", "iso3", and "plaintext".
emailSearchAfterChars	2	integer	Number of characters after "." to search searching (where "." is after "@").
emailSearchOnBlur	TRUE	boolean	If TRUE, search will be on blur; otherwise, on keyup.
companyNameSearchAfterChars	2	integer	Number of input characters before system attempts typeahead queries.
initialFocusFieldName		string	Field where initial focus should be placed; use an empty string (" ") to disable focus stealing entirely.



dropDownSize	10	integer	Size of dropdown list. Only valid if useLIDropdowns is false and CSS has not set the height otherwise.
noSearchResultsClearFieldEnabled	TRUE	boolean	Clear all mapped form fields (even if filled in via other means) depending on success of e-mail search input.
clearFieldsIfNoEmailSearchMatch	FALSE	boolean	Clears relevant persona fields (name, title, phone) depending on success of e-mail search input.
attributeForFieldLookup	name	string	Eligible values are "name" or "id". Indicates what attribute should be used for field mapping.
viffResponse		function	Function on the host page that will be called in lieu of changing form fields directly via the mapping. Output will be in JSON.
setInitialCountryByIp	TRUE	boolean	If disabled, the user's IP address will not be used to automatically determine typeahead search scope.
typeaheadSearchResultsDisplayType	default	string	Eligible values are "default", "noStreet", and "nameOnly". Please be aware that the last one may cause apparent duplicate results to appear.

\*Recommended parameter

#### ABOUT DUN & BRADSTREET

Dun & Bradstreet, a leading global provider of business decisioning data and analytics, enables companies around the world to improve their business performance. Dun & Bradstreet's Data Cloud fuels solutions and delivers insights that empower customers to accelerate revenue, lower cost, mitigate risk, and transform their businesses. Since 1841, companies of every size have relied on Dun & Bradstreet to help them manage risk and reveal opportunity. Twitter: [@DunBradstreet](#)