# Custom Plugins

Custom plugins allow you to extend the capabilities of the Profound.js low-code interface. They can be generic, similar to the plugins shipped with Profound.js, or specific to your business. Typically, an experienced Node.js developer or a 3rd party organization will write a reusable plugin, and then other developers at your company can use that plugin in a point-and-click manner to create applications.

## Plugin File

A plugin is a Node.js module that exports a number of properties, including a generator function. For example:

**Logger Plugin (date-time-logger.js)**

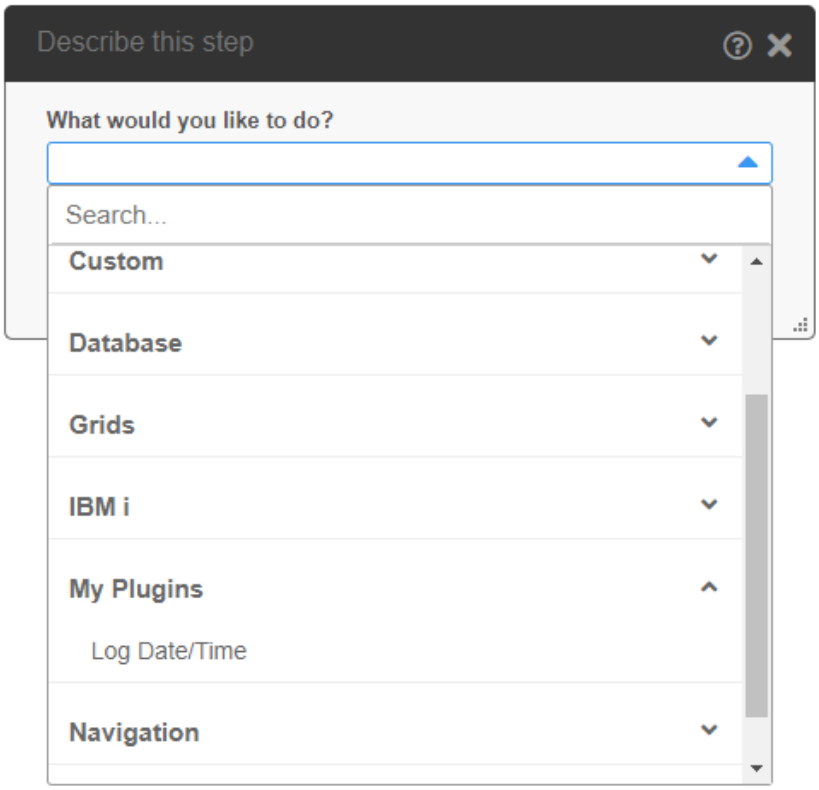
```
module.exports = {
  name: "date-time-logger",
  text: "Log Date/Time",
  category: "My Plugins",
  help: "This action outputs the current date and time to the console.",

  generator: function(answers) {
    return `console.log(new Date());`;
  }
}
```

The custom plugin should be saved as a JavaScript file into the *profoundjs/plugins* directory, where *profoundjs* is the directory where you installed Profound.js.

If you're working in the cloud, using the Profound.js Spaces environment, you must create a *plugins* directory at the root of your workspace, and save the plugin file there.

After adding a plugin, you should restart your server and refresh the IDE.

The plugin will then appear in the low-code interface:




## Plugin Properties

You can export the following properties from your plugin file:

- **name** – a unique name to identify the plugin; this name will be saved to the Rich Display JSON to identify the plugin when it is selected
- **text** – the title of the plugin that will be visible to the user

- **defaultDescription** - the default description to use if the user does not describe a step
- **category** – plugins are categorized for convenience; choose an existing category name or provide a new name; if a new name is provided, a new category will be created
- **clientSide** – setting this to true indicates that this is a client-side plugin and the generated code should execute on the client rather than the server
- **context** - optionally restricts the plugin to a specific context; you can specify "rdf" for Rich Display File development or "api" for API development
- **structure** – setting this to true indicates that this is a condition or loop structure that encompasses other steps in the routine
- **help** – help text that should appear when the user clicks the **?** icon on the dialog
- **questions** – an array of questions to ask the user; see below
- **generator** – a function that receives a list of the user's answers, as a JavaScript object keyed by question id's, and returns generated code in String format

## Question Properties

Each question in the **questions** array is an object with the following possible properties:

- **id** – unique id for the question; this id is used to save the answer to the Rich Display JSON file
- **text** – the question text that the user actually sees
- **dynamicText** - optional function that returns additional text dynamically based on other answers
- **type** – question type; the following types are available:
    - **textbox** – free form textbox to capture the answer
    - **textarea** – multi-line input control
    - **checkbox** – allows the user to specify a true or false value
    - **multi-select** – allows the user to select one or more multiple values from a list of choices and optionally to type a value manually
    - **dropdown** – allows the user to select one choice from a list
    - **combo-box** – allows the user to select from a list or type in a value
    - **browser-dropdown** – a list of choices rendered using the <SELECT> tag in HTML
    - **code-editor** – allows the user write custom code
    - **column-values** – allows the user to specify a value for each table column
    - **record-property-values** – allows the user to specify a value for each record property
    - **api-output-values** - allows the user to specify a value for each API output property
    - **screen-values** – allows the user to specify a value for each screen field
    - **join** – allows the user to join multiple tables together
    - **and-or** – allows the user to select a value of AND or OR for a condition
    - **ibmi-parm** – allows the user to define an IBM i parameter
    - **widget-preview** – allows you to show a preview of a selected widget
- **required** - set this to true if the user is required to answer this question
- **showOptional** - set this to true to indicate to the user that this answer is optional
- **defaultValue** – the value to prefill as a default
- **condition** – if this question should only be asked based on answers to other questions, provide a function that receives answers, and returns a Boolean value to indicate whether this question should be asked
- **validation** – a function that can return a validation message; it receives the answer as the first parameter and all answers as the second parameter
- **occurs** – identifies this as a multi-occurrence question and specifies the maximum number of times the question can be asked
- **help** – help text to associate with this question
- **inputType** – specifies the type attribute for a textbox (e.g. “number”)
- **placeholder** – specifies the placeholder text for a multi-select or a textbox question
- **parmType** – specifies the IBM i parameter type, with valid values being: “program”, “service program”, “return value”
- **height** – height of a code editor or textarea
- **language** – code editor language
- **insertDynamic** – allow the user to insert dynamic content into the editor based on low-code suggestions
- **dynamicFormat** – an expression that specifies how dynamic content is inserted; the word “value” in the expression is replaced by the actual value selected by the user
- **dynamicSource** – specify “widgets” to have the source be a list of widgets instead of the standard list of low-code suggestion items
- **dynamicFilter** – specifies whether the list of suggestions should be filtered by “values”, “records”, or “lists”
- **singleSelection** – specifies that a multi-select question only allows one selection
- **selectAll** – set this to true to provide the ability to select all options in a multi-select question
- **freeForm** – set this to true to specify that you can type your own value into a multi-select question
- **multiLine** – set this to true to specify that the free form area of a multi-select question is a multi-line text area
- **search** – set this to true to specify that a search box will be shown that allows the user to search the choices in a multi-select question
- **source** – specifies the source for the choices in a multi-select question; valid values are:
    - **columns** – list of database table columns
    - **comparison-types** – list of comparison operators
    - **connections** – list of database connections the user has configured
    - **criteria-snippets** – selection criteria snippets for an SQL-based WHERE clause
    - **css-classes** – list of CSS classes defined within the currently loaded CSS files
    - **endpoints** – list of routes or endpoints the user has defined
    - **files** – list of files within the current workspace
    - **grid-fields** – fields in a grid
    - **grid-input-fields** – only input fields in a grid
    - **grid-numeric-fields** – only numeric fields in a grid
    - **grids** – list of grids defined within the Rich Display file
    - **plugins** – list of defined plugins
    - **properties** – list of widget properties
    - **routines** – current list of routines defined within the Rich Display file
    - **screens** – list of screens defined within the Rich Display file
    - **tables** – list of tables in the configured database

    - **variables** – list of variables captured from other steps
    - **widgets** – list of widgets on the screen
- **basedOn** – specifies the id of a question on which the source is based on; for example, a list of columns can be based on a question that asks to select a database table
- **basedOnList** - specifies the id of a question that asks for a list on which the source is based on
- **forOrderBy** - specifies that a list of columns is for an "order by" selection with options for ascending and descending
- **collapsibleGroups** – if the source provides a grouped list, this property determines if the groups should be collapsible
- **filter** – filters the list of variables by classification, such as “values”, “records”, or “lists”
- **varTypes** – an array of field/variable types to list; can include “display”, “work”, “global”, and “session”
- **captureInto** – specifies whether the answer to this question should be captured as a “global” property or a “work” variable
- **captureType** – specifies whether the captured variables are captured as “values”, “records”, or “lists”
- **captureValuesBasedOn** – when capturing records or lists of records, specifies the id of the question which would provide properties for those records
- **criteriaSettings** – when set to true, specifies that the multi-select question should present criteria settings
- **extension** – when the source is “files”, specifies which file extension to filter by (e.g. ".json")
- **showGridsAsLists** – when set to true, include Rich Display file grids for for “variables” source with a filter of “lists”
- **isForList** – specifies that a record-property-values question is collecting values for a list