



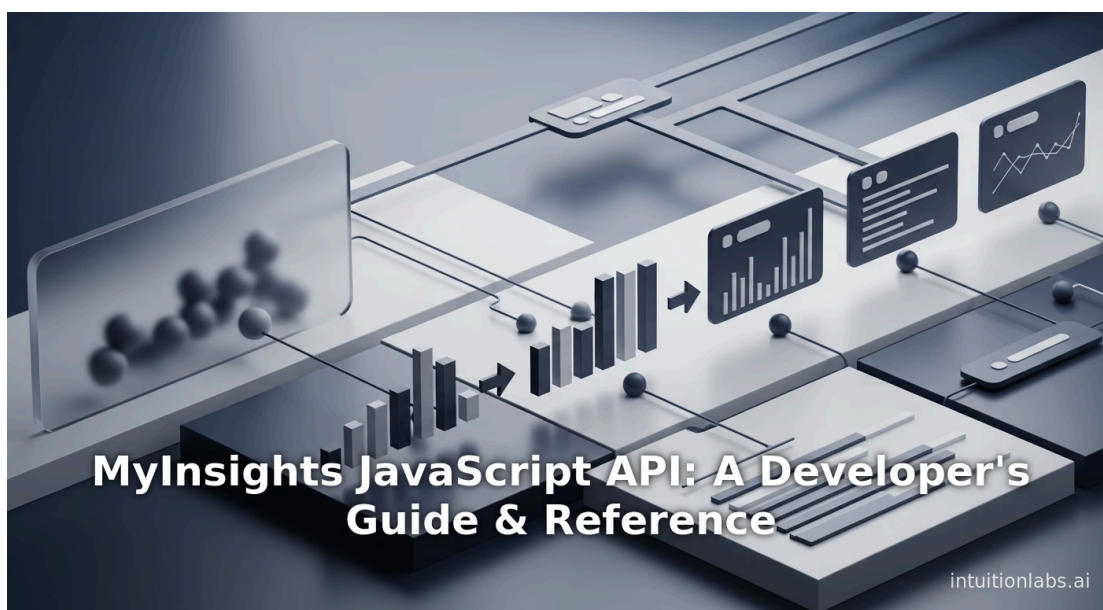
INTUITIONLABS / RESEARCH & PRACTICAL GUIDANCE

MyInsights JavaScript API: A Developer's Guide & Reference

Build practical AI for pharma and biotech with IntuitionLabs. We help life-science teams turn complex information and workflows into useful software, governed knowledge systems and AI tools.

Published by IntuitionLabs · 2026-01-10 · myinsights · veeva crm · javascript api · myinsights api · runquery · veeva development · data visualization

Original article: <https://intuitionlabs.ai/articles/myinsights-javascript-api-reference>



In this article

1. [Executive Summary](#)
 2. [Introduction and Background](#)
 3. [MyInsights Platform and Architecture](#)
 4. [MyInsights JavaScript API: Overview and Core Concepts](#)
 5. [Data Query Methods](#)
 6. [Label and Metadata Methods](#)
 7. [Utility and Integration Methods](#)
 8. [Navigation and UI Methods](#)
 9. [Summary of Key API Methods](#)
 10. [MyInsights Development and Usage Patterns](#)
 11. [Case Studies and Real-World Examples](#)
 12. [Technical and Platform Considerations](#)
 13. [Implications and Future Directions](#)
 14. [Conclusion](#)
-

Executive Summary

MyInsights is a powerful data visualization and analytics platform integrated into Veeva CRM, designed to deliver tailored, actionable insights directly within the CRM environment. Leveraging a JavaScript API, MyInsights enables developers and business users to create custom dashboards and interfaces – accessible offline on mobile devices – that query real-time Veeva CRM data and allied sources to help field teams make data-driven decisions. Introduced in 2017, MyInsights has since evolved into a mature framework, underpinning use cases such as sales performance tracking, territory planning, pre-call engagement preparation, and more (www.veeva.com) (www.veeva.com). This report offers a comprehensive analysis of the MyInsights JavaScript API and its ecosystem, covering historical context, technical architecture, core methods, usage scenarios, and future directions. We examine each API method in depth – from `runQuery` and `getFeedbackData` to utility functions like `getSFDCSessionID` and `request` – and explain their roles, parameters, and returned data (with examples) (developer.veevacrmqa.com) (developer.veevacrmqa.com). Insights are provided from multiple perspectives: developer and administrator guidance from official Veeva documentation, end-user impact illustrated by case studies (e.g. GSK's reported 30-minute-per-day time savings) (www.veeva.com), and industry context drawn from partner comments and market analyses. The report also includes tables summarizing key API functions and real-world deployments. All technical claims and examples are supported by primary sources from Veeva's official documentation, help portals, and industry publications (www.veeva.com) (www.veeva.com) (developer.veevacrmqa.com). We discuss performance considerations (such as offline sync constraints and query syntax rules (crmhelp.veeva.com)), present evidence from field implementations, and consider future implications (e.g. integration with [Veeva Nitro analytics](#) and AI-driven insights). Ultimately, this report serves as an in-depth reference for understanding the MyInsights JavaScript API's capabilities, usage patterns, and its strategic role in life sciences commercial operations.

Introduction and Background

Veeva announced that support for Veeva CRM ends in December 2029 ([Veeva announcement](#)).

Veeva Systems is a leading provider of cloud-based software solutions for the life sciences industry. Its flagship product, **Veeva CRM**, is a purpose-built customer relationship management platform for pharmaceutical and biotech companies. Serving roles such as medical and commercial field teams, Veeva CRM captures vast amounts of data on healthcare providers (HCPs), accounts, orders, and other key metrics. **MyInsights** is a data visualization and reporting capability within Veeva CRM, introduced in 2017, that empowers users to view *integrated analytics* within the CRM interface ([www.veeva.com](#)) ([www.veeva.com](#)). As a cloud-based library of HTML/JavaScript dashboards and widgets, MyInsights provides “tailored insights right in Veeva CRM” so that “field teams are empowered with the right information exactly when and where they need it” ([www.veeva.com](#)) ([www.veeva.com](#)). Prior to MyInsights, users often needed to pull data from multiple systems (sales data, market data, CRM records) and manually assemble reports. MyInsights eliminated much of this fragmentation by allowing custom content – from simple charts to interactive pre-call planning pages – to be delivered **directly in the CRM**, even on mobile devices.

Architecturally, MyInsights content is packaged as **HTML and JavaScript** and uploaded into Veeva CRM. Administrators create specialized Veeva CRM content records (e.g. **HTML_Report_vod** entries) that serve as entry points for MyInsights dashboards. The JavaScript code runs inside the Veeva CRM client (web or mobile), and leverages the **MyInsights JavaScript library** (a client-side API) to query CRM data, Nitro Analytics data, and other sources. By 2017, Veeva stated that “since Veeva CRM MyInsights is based on standard HTML and JavaScript, it’s easy to create new or extend pre-built data visualizations” ([www.veeva.com](#)). Furthermore, an ecosystem of partners and developers has grown around MyInsights; by late 2017, nearly 20 technology and service partners had become certified to develop MyInsights content ([www.veeva.com](#)). This reflects the platform’s strategic importance: providing *in-context analytics* enables “the fastest, most effective planning and execution” for sales and medical teams ([www.veeva.com](#)).

In recent years, user feedback has corroborated MyInsights’ value. For example, GlaxoSmithKline (GSK) reported that custom MyInsights pages helped its field teams save **30 minutes per day** on engagement planning, by consolidating all relevant call preparation data into one screen ([www.veeva.com](#)). IntuitionLabs – a Veeva partner – described use cases where combined dashboards and workflows (built on **Veeva X-Pages** and MyInsights) delivered “real-time analytics” and “significant reduction in pre-call planning time” for customers ([intuitionlabs.ai](#)). These real-world outcomes underscore that dismantling data silos and surfacing analytics at the point of care can measurably boost productivity and decision quality.

However, building effective MyInsights content requires a solid understanding of the underlying API and platform constraints. Queries must be carefully formulated to work on both online and offline devices, and the MyInsights library relies on a particular JavaScript runtime model (based on promises via the **Q library**) ([developer.veevacrm.com](#)) ([developer.veevacrmqa.com](#)). Developers often ask: *How do I retrieve CRM records or territory feedback data in MyInsights? How are queries structured? What API methods are available?* This report addresses these questions by providing an in-depth technical reference to the MyInsights JavaScript API, drawing on official Veeva documentation, case studies, and expert guidance.

MyInsights Platform and Architecture

MyInsights operates as a client-side framework embedded in Veeva CRM. When a user--often a pharmaceutical sales representative--opens a MyInsights report, the Veeva CRM application (desktop or mobile) renders the custom HTML/JavaScript bundled for that report. The JavaScript includes the official **MyInsights library** (commonly named `MyInsightsLibrary.js`), which exposes a global `ds` (`DataService`) object in the client's window context (developer.veevacrmqa.com). This object provides the interface for data queries and other interactions with CRM. In practice, developers create an `index.html` (or similar) that loads the MyInsights library and optionally other scripts, such as the *Q promise library* and a custom `main.js` (containing user code) (developer.veevacrm.com) (developer.veevacrm.com).

The architecture supports offline operation on mobile devices: a device periodically syncs critical data (via Nitro or mobile object sync) such that MyInsights content can query that local cache when offline. For instance, the Nitro integration allows large datasets (e.g. sales or prescription data) to be made available on the device, while the MyInsights library abstracts whether a query is hitting local copy or making a server request. `getInStatement(ids)` returns a string for use in a query's WHERE clause. Do not call `.then()` on that string.

MyInsights content is delivered via Veeva objects. For example, administrators can create a *MyInsights dashboard* by uploading a ZIP of HTML and JS files through the CRM UI. Alternatively, MyInsights **Studio** provides a no-code interface where business users define pages and data elements (e.g. *Get CRM Data*, *Get Sales Data*, *Filter List*, etc.) (crmhelp.veeva.com) (crmhelp.veeva.com). These Studio elements are higher-level constructs, but ultimately they translate into API calls behind the scenes. (This report focuses on the lower-level JavaScript API, but will reference Studio capabilities where relevant.)

Key characteristics of the MyInsights environment:

- **JavaScript-based:** Custom code is written in HTML/JS. MyInsights relies on standard web technologies, making it accessible to web developers and also to non-coders via Studio.
- **Q promises:** The MyInsights library depends on *Q* for promise support (developer.veevacrmqa.com). Thus, asynchronous calls must use `.then()` or `.done()`.
- **Global DataService (ds):** As soon as the MyInsights library loads, it instantiates a global `DataService` object (aliases `window.ds`) that provides the API methods (developer.veevacrm.com).
- **Cross-platform queries:** Queries executed by MyInsights can run in an **online mode** (via SOQL against Salesforce) or **offline** (against a device-local SQLite store). Care must be taken with string quoting, IDs, and syntax to ensure cross-platform compatibility (developer.veevacrm.com) (crmhelp.veeva.com). For instance, when targeting offline mobile, queries may not support the SQL 'IN' operator in the same way as online SOQL (the `getInStatement` helper assists with that) (developer.veevacrmqa.com).
- **Dependencies on CRM metadata:** The API often relies on the CRM data model, including custom objects (e.g. `Call2_vod__c`) and fields defined by a customer's Veeva CRM schema. Queries must use **case-sensitive API names** matched exactly, especially on mobile (developer.veevacrm.com) (crmhelp.veeva.com).

- **Integration with Nitro:** MyInsights can query data from Veeva Nitro (Analytics), enabling dashboards with aggregated sales or HCP data. Configuring Nitro sync involves metadata and YAML definitions (see Nitro documentation (nitrohelp.veeva.com)), but once deployed, data appears as special “Account_Sales_Summary_Insights” or Nitro tables that MyInsights can query via `querySalesData` (developer.veevacrmqa.com) (developer.veevacrmqa.com).

In summary, MyInsights is essentially an embedded web app platform inside Veeva CRM. Experience with web development is directly applicable, but developers must learn the specialized API (`ds` and its methods) that connects to CRM. The sections below dissect that API in detail.

MyInsights JavaScript API: Overview and Core Concepts

The heart of MyInsights functionality is provided through the **DataService (ds) object**, which exposes numerous methods for querying data and interacting with the CRM environment. These methods fall into several categories:

- **Data Query Methods:** For retrieving records and datasets (e.g. `runQuery`, `queryRecord`, `querySalesData`, `getDataForCurrentObject`, `getFeedbackData`).
- **Metadata/Label Methods:** For obtaining human-readable labels (e.g. `getFieldLabels`, `getObjectLabels`, `getPicklistValueLabels`, `getVeevaMessagesWithDefault`).
- **Utility and Integration Methods:** For external operations (e.g. `getSFDCSessionID`, `getSSOAccessToken`, `generic request(config)`).
- **Navigation/UI Methods:** For navigating within the CRM interface (e.g. `viewRecord`, `viewSection`).
- **Helper Methods:** To ease query construction (e.g. `getInStatement`, `joinQueries`) or provide context (e.g. `getAvailableObjects`, `getObjectMetadata`).

`joinQueries(q1, q2, keyForID, fieldsLabelsToExclude, primaryObjectName)` returns a joined object; use the returned object directly, without `.then()`.

Broadly speaking, the **DataService** is instantiated automatically when the MyInsights library loads and is globally accessible as `ds` (or `window.ds`) (developer.veevacrm.com). For example, calling `ds.runQuery(config)` begins an asynchronous query operation, and the developer supplies a callback to handle the response (developer.veevacrmqa.com).

Under the hood, MyInsights translates these JavaScript calls into appropriate platform queries. Online (browser/desktop) clients use Salesforce SOQL conditions; offline mobile queries use a local store. The API abstracts these differences, but with some caveats on query syntax (described later).

In the sections that follow, we examine each major category of methods, provide examples of usage, and highlight returned data structures. Where available, we include excerpts from the official MyInsights documentation (Veeva Developer site and help) to illustrate parameters and behavior (developer.veevacrmqa.com) (developer.veevacrmqa.com) (developer.veevacrmqa.com). We also include a table summarizing key methods and their purposes to aid understanding.

Data Query Methods

Most MyInsights dashboards require fetching CRM or Nitro records. The DataService provides several query methods:

- **runQuery(queryConfig)**: General-purpose query for most objects. Returns multiple records with metadata.
- **queryRecord(queryConfig)**: Returns raw data for a single record (or single aggregate query), without the extra metadata.
- **querySalesData(queryConfig)**: Specialized method to fetch Nitro (Account Sales Summary) data.
- **getDataForCurrentObject(object, field)**: Fetches **one field's value** from the current record (context).
- **getFeedbackData()**: Retrieves Align/Territory feedback data for the current user's territory.
- **joinQueries(q1, q2, ...)**: Joins two query result sets (like a SQL outer join).
- **getDataForCurrentObject and others**: quick helper to get one value.

Below, we detail each.

runQuery(queryConfig)

Purpose: Execute a data query on the specified object, retrieving multiple records and associated labels (developer.veevacrmqa.com) (developer.veevacrmqa.com). This is the most commonly used method for general data retrieval.

- **Usage:** Pass an object with object, fields, and optional where, sort, limit. For example:

```
var queryConfig = {
  object: "Account",
  fields: ["Name", "Account_Owner_vod__c"],
  where: "Status = 'Active'",
  sort: ["Name ASC"],
  limit: 50
};
ds.runQuery(queryConfig).then(function(result) {
  console.log(result);
  // result.data is an array of records
});
```

- **Example Response:** On success, MyInsights returns JSON containing: data (array of record objects), fieldLabels (list of field names and displays), object metadata, etc (developer.veevacrmqa.com). For instance:

```
{
  "data": [{
    "Name": {
      "value": "Acme Corp",
      "display": "Acme Corp",
```

```

"dataType": "string",
"label": "Name"
},
{
...second record...
},
],
"object": {
"plural": "Accounts",
"singular": "Account",
"name": "Account"
},
"name": "Account",
"fieldLabels": [{
"name": "Name",
"display": "Name"
}]
}

```

- **Parameters:** The queryConfig object supports:
 - object (string): API name of object to query (e.g. "Account").
 - fields (array of strings): API names of fields to retrieve.
 - where (string, optional): WHERE clause conditions (without the word "WHERE"; e.g. "OwnerId = '005D...'"). **Note:** On mobile, only limited syntax is supported. Quotes and IDs must match exactly, and global IDs must be 15- or 18-character as appropriate (developer.veevacrm.com) (crmhelp.veeva.com).
 - sort (array of strings, optional): ORDER BY clause (e.g. ["LastModifiedDate DESC"]).
 - limit (number, optional): Max records to return.
 - **Return Value:** A promise that resolves to a JSON object (as above). The developer can iterate `result.data` or use in charts, etc (developer.veevacrmqa.com) (developer.veevacrmqa.com).
 - **Example** (from official docs):

```

ds.runQuery({
object: "Account",
fields: ["Name"],
limit: 1
}).then(function(resp) {
console.log(resp.data);
});

```

The Response snippet in the docs shows an account name returned (developer.veevacrmqa.com).

- **Notes:** The documentation warns that fields must be unique in the query (no duplicate field names) and to refer to [SOQL syntax for online and iOS predicate syntax for mobile] (developer.veevacrmqa.com) (developer.veevacrmqa.com).

queryRecord(queryConfig)

Purpose: A variant that **fetches a single record's data** (or an aggregate), returning only raw field values without extra metadata or labels (developer.veevacrm.com). It's used when you need to retrieve one record (e.g. a specific account or HCP) and don't need the wrapper structure that runQuery provides.

- **Usage:** Similar to runQuery, but expecting only one record (by primary key or limiting to 1). For example:

```
ds.queryRecord({
  object: "Account",
  where: "Account_ID_vod__c = '00136000001'"
})
.then(function(rec) {
  console.log(rec.Account.SOMA__c); // raw field
});
```

- **Example Response:** Returns an object mapping API names to values. E.g.:

```
{
  "Account": {
    "Name": "Acme Corp",
    "Revenue__c": 12345
  },
  "success": true
}
```

(Structure may vary; official doc doesn't show but implies raw key-value) (developer.veevacrmqa.com).

- **Key Difference:** Unlike runQuery, does not include fieldLabels or data array; it gives raw data (likely as key-value pairs nested under the object name) (developer.veevacrm.com).
- **When to use:** Useful for retrieving context data (not listing multiple records). For example, getting HCP name by ID, etc.
- **Note:** The MyInsights Dev Guide states that this returns "native interface" (likely meaning raw) (developer.veevacrmqa.com) and says it's for when you don't want added metadata.

getDataForCurrentObject(objectApiName, fieldApiName)

Purpose: Quickly fetches **one field's value** from the current record context (e.g. the account or plan being viewed) (developer.veevacrmqa.com).

- **Usage:** ds.getDataForCurrentObject("Account", "Specialty_vod__c") returns a promise with the field's value.
- **Example** (from docs):

```
ds.getDataForCurrentObject('Account', 'Credentials_vod__c')
.then(function(resp) {
```

```
console.log(resp.Account.Credentials_vod__c);
});
```

The sample shows a response `{"Account":{"Credentials_vod__c":"D0"}, "success": true}` ([developer.veevacrmqa.com](#)) ([developer.veevacrmqa.com](#)), and notes it returns the value of that field for the current record.

- **Return:** The resolved result will have the object name mapping to an object with the requested `fieldName:value` ([developer.veevacrmqa.com](#)).
- **Typical Use:** E.g. in an Account details MyInsights page, get the Account's info. After retrieval you can e.g. update the interface.
- **Limitations:** Only gets one field; for multiple fields use `runQuery`.

querySalesData(queryConfig)

Purpose: Fetch CRM sales (Nitro) data at the account level. Specifically queries the special `Account_Sales_Summary_Insights` object, which holds aggregated sales data loaded via Nitro ([developer.veevacrmqa.com](#)) ([developer.veevacrmqa.com](#)).

- **Usage:**

```
var config = {
  object: "Account_Sales_Summary_Insights",
  fields: ["Account_Id_vod__c", "Market_vod__c", "Sales_Date_vod__c", "Sales_Data_vod__c"],
  limit: 5
};
ds.querySalesData(config).then(function(resp) {
  console.log(resp.Account_Sales_Summary_Insights);
});
```

- **Behavior:** The method returns the results of the query in a promise.
- **Example:** Official example shows two records for one account's multiple dates ([developer.veevacrmqa.com](#)). The response JSON includes `"record_count"` and an array under `"Account_Sales_Summary_Insights"`.
- **Notes:** To use, CRM Org must have Nitro sales data properly loaded and must follow restrictions (no Goals or Payer-specific data, see CRM help ([crmhelp.veeva.com](#))). It's mainly for retrieving account-level time series of sales (columns like TRx, NRx).

getFeedbackData()

Purpose: Returns territory feedback data from Align and CRM for the current user's territory. Useful in territory planning pages. It includes account lists, goal metadata, and challenge/target info ([developer.veevacrmqa.com](#)) ([developer.veevacrmqa.com](#)).

- **Usage:** Simply call `ds.getFeedbackData()`; no parameters. ([developer.veevacrmqa.com](#)).
- **Example Response:** The API returns a large JSON containing multiple fields:

- `accountDetailMetadata` (labels of account data fields),
- `addedAccounts`, `droppedAccounts`, lists of new/dropped accounts/zipcodes,
- `goalMetadata` and `productMetricMetadata` (detail territory goals),
- Flags like `manager`, `canChallenge`, `targetingEnabled`,
- `accountData` (detailed per-account info with nested goal details, challenge info, team selling, etc.) ([developer.veevacrmqa.com](#)) ([developer.veevacrmqa.com](#)).
- **Uses:** This method is typically used in territory and feedback dashboards, where reps need to see changes to their assigned accounts or goals. The API docs call it “data about upcoming changes to the current territory” ([developer.veevacrmqa.com](#)).
- **Documentation:** Parameter-less; returns full set of feedback data in JSON ([developer.veevacrmqa.com](#)) ([developer.veevacrmqa.com](#)).
- **Returned Fields:** Selected response fields include:

Field	Description
<code>addedAccounts</code> (int)	Number of accounts newly added to the territory.
<code>droppedAccounts</code> (int)	Number of accounts removed from the territory.
<code>goalMetadata</code> (array)	Info about channel/product goals for territory (fields: <code>channelOrProductName</code> , <code>goals</code>).
<code>productMetricMetadata</code>	Info on metrics and products (list of available metrics, products) (developer.veevacrmqa.com).
<code>readOnly</code> (bool)	If the territory is locked/read-only.
<code>teamSellingEnabled</code>	If team selling feature is enabled.
<code>territoryName</code>	Territory name, ID.
<code>accountData</code> (array)	Details for each account: addresses, <code>goalDetails</code> , <code>productMetricDetails</code> , etc. (developer.veevacrmqa.com) (developer.veevacrmqa.com).

Table: Selected fields from `getFeedbackData` response (source: Veeva MyInsights docs ([developer.veevacrmqa.com](#)) ([developer.veevacrmqa.com](#))).

(The full list is lengthy; see documentation for exhaustive field descriptions ([developer.veevacrmqa.com](#)) ([developer.veevacrmqa.com](#))).

- **Notes:** The API text notes the default behavior is to return all data (no parameters) ([developer.veevacrmqa.com](#)). If an org has Territory Feedback feature, this method is essential to access that data.

`joinQueries(q1, q2, keyForID, fieldsLabelsToExclude, primaryObjectName)`

Purpose: Merge (outer-join) the results of two separate queries into one combined result by matching on an ID key. For example, join a list of products (q1) with related email/send metrics (q2) on product ID ([developer.veevacrmqa.com](#)) ([developer.veevacrmqa.com](#)).

- **Usage:** Provide two query-result objects (the .data responses as returned by runQuery), specify the join key and which query's fields should be considered "primary". E.g.:

```
var joined = ds.joinQueries(q1, q2, 'Product_vod__c', {
  ID: true
}, 'Sent_Email_vod__c');
```

- **Example:** The documentation gives a detailed example:
- q1 has Product IDs and Names,
- q2 has Sent_Email records linking product ID to record ID,
- They call joinQueries(q1, q2, 'Product_vod__c', {ID:true, Product_vod__c:true}, 'Sent_Email_vod__c'). The response merges rows where ID in q1 matches Product_vod__c in q2 ([developer.veevacrmqa.com](#)) ([developer.veevacrmqa.com](#)), producing a combined data array.
- **Result:** The returned object contains a single result set with data combining fields from both queries, one per matching row ([developer.veevacrmqa.com](#)) ([developer.veevacrmqa.com](#)), and new fieldLabels.
- **Notes:** This is a client-side join, not a SOQL join. Useful for correlating data from different objects in a single table view. The method's doc emphasizes similar to an SQL outer join ([developer.veevacrmqa.com](#)).
- **Parameters** (from docs):
- q1, q2: results from runQuery or similar.
- keyForID (string): the join field name in the **primary** query.
- fieldsLabelsToExclude (object): a map of field names (API names) to exclude from the final join (to avoid duplicates).
- primaryObjectName (string): the API name of the primary object (determines which query is primary).

See Table 1 for a summary of key parameters and usage.

Label and Metadata Methods

MyInsights provides methods to retrieve human-readable labels for objects, fields, and picklist values, which is useful for dynamic interfaces or translations.

getFieldLabels(queryConfig)

Purpose: Given an object and list of field API names, returns the **custom labels** for those fields ([developer.veevacrmqa.com](#)) ([developer.veevacrmqa.com](#)). This helps display nice names rather than raw API names.

- **Usage:** Pass { object: "ObjectApiName", fields: ["Field1", "Field2"] }.

```
ds.getFieldLabels({
  object: "Contact",
  fields: ["LastName", "BirthDate"]
})
```

```

})
.then(function(resp) {
  console.log(resp.Contact);
});

```

- **Example** (from docs): Request:

```

ds.getFieldLabels({
  object: "Clinical_Trial__c",
  fields: ["Name", "Display"]
}).then(function(resp) {
  console.log(resp);
});

```

Response:

```

{
  "Clinical_Trial__c": {
    "display": "Start Date",
    "name": "Start_Date_vod__c"
  }
}

```

This shows the display labels for the fields “Display” and “Name” in the Clinical_Trial__c object (developer.veevacrmqa.com).

- **Parameters:** A single object queryConfig:
 - object (string): API name of object.
 - fields (array): array of field API names (as strings).
- **Returns:** A promise resolving to an object mapping the object name to another object mapping each requested field to its label, e.g. { "ObjectApi": { "FieldApi": "Field Label" } } (developer.veevacrmqa.com).
- **Type:** The resolved promise is essentially a JSON object.
- **Use Cases:** Display dynamic column headers in tables, ensure UI respects translations, etc.

getObjectLabels(objectList)

Purpose: For an array of object API names, returns their singular and plural labels (developer.veevacrmqa.com). This is useful for internationalization or dynamically generating UI elements.

- **Usage:** Provide an array of object names:

```

ds.getObjectLabels(["Case", "Lead"]).then(function(resp) {
  // resp.Case will have singular/plural
});

```

- **Example** (from docs):

```
ds.getObjectLabels(["Medical_Event_vod__c"])
  .then(function(resp) {
    console.log(resp);
  });
```

Response:

```
{
  "success": true,
  "Medical_Event_vod__c": [{
    "plural": "Medical Events",
    "singular": "Medical Event"
  }]
}
```

(It returns an array for each object, containing one object with plural and singular labels.)
(developer.veevacrmqa.com).

- **Parameters:**
- The method accepts a single array of strings (`objectList`).
- **Returns:** Promise that resolves to an object where each input object name maps to an array of label objects (developer.veevacrmqa.com).
- **Type:** Promise resolving to an object.
- **Note:** If only one object is requested, the response still wraps the labels in an array.

getPicklistValueLabels(object, field)

Purpose: Given an object and picklist field, returns the translations (labels) for each picklist value of that field (developer.veevacrmqa.com).

- **Usage:**

```
ds.getPicklistValueLabels("Account", "Industry_vod__c")
  .then(function(resp) {
    console.log(resp.Account.Industry_vod__c); // e.g. { "Pharmaceuticals_vod": "Pharmaceuticals", ... }
  });
```

- **Example** (from docs):

```
ds.getPicklistValueLabels("Account", "Career_Status_vod__c")
  .then(function(resp) {
    console.log(resp);
  });
```

Response:

```
{
  "Account": {
    "Career_Status_vod__c": {
      "Emerging_vod": "Emerging",
      "Peak_vod": "Peak",
      "Retired_vod": "Retired"
    }
  },
  "command": "queryReturn",
  "deferredId": "callback_1482531900000"
}
```

(This maps API values to their labels (developer.veevacrmqa.com).)

- **Parameters:**
 - object: string, API name of the object (e.g. "Account").
 - field: string, API name of the picklist field (e.g. "Industry_vod__c").
- **Returns:**
 - Promise resolving to an object like { object: { field: { valueLabelMap } } }.
- **Type:** Promise.
- **Use:** Useful to populate dynamic filter dropdowns or for display of picklist-based columns.

getVeevaMessagesWithDefault(tokens, localeKey)

Purpose: Retrieves translated text for a set of Veeva Message tokens in a given language. Veeva Messages are standard labels used in the UI, stored in a custom object. This method allows retrieving them by category/name, with a fallback to a default text if not found (developer.veevacrmqa.com).

- **Usage:** Provide an array of tokens ({msgCategory, msgName} pairs) and a locale key (e.g. "en_US").

```
var tokens = [{
  msgCategory: "Common",
  msgName: "UPCOMING"
}];
ds.getVeevaMessagesWithDefault(tokens, "en_US")
.then(function(resp) {
  console.log(resp);
});
```

- **Example** (from docs): Calling with tokens for category Common name UPCOMING yields:

```
[{
  "Language_vod__c": {
    "value": "en_US",
    "display": "en_US",
```

```

...
},
"Name": {
  "value": "UPCOMING",
  "display": "UPCOMING",
  ...
},
"Text_vod__c": {
  "value": "Upcoming",
  "display": "Upcoming",
  ...
},
"Category_vod__c": {
  "value": "Common",
  "display": "Common",
  ...
}
}]

```

This means the message “UPCOMING” has text “Upcoming” in English (developer.veevacrmqa.com).

- **Parameters:**
 - tokens: Array of objects {msgCategory:string, msgName:string}.
 - localeKey: string, e.g. "en_US".
- **Returns:** A promise that resolves to an array of message records, each containing fields like Language_vod__c, Name, Text_vod__c, Category_vod__c (developer.veevacrmqa.com).
- **Type:** Promise resolving to an array of JSON objects.
- **Use:** Displaying localized text in UI, ensuring consistent message labels across MyInsights pages.

Utility and Integration Methods

These methods help with session management, single sign-on, and calling external services.

getSFDCSessionID()

Purpose: Retrieves information about the current Salesforce session, including the session ID, instance URL, and whether the org is a sandbox (developer.veevacrm.com). The session ID cannot be used for authentication outside the current context; it's mainly for passing to back-end systems in custom integrations.

- **Usage:**

```

ds.getSFDCSessionID().then(function(resp) {
  console.log(resp.data.sessionId);
});

```

- **Response:** A JSON of the form:

```
{
  "success": true,
  "data": {
    "sessionId": "00Dxxxxxxxx!... (Long session ID)",
    "instanceURL": "https://csxx.salesforce.com",
    "isSandbox": false
  }
}
```

(As shown in documentation (developer.veevacrm.com).)

- **Parameters:** None.
- **Returns:** Promise resolving to session data. If offline or invalid, errors are returned with codes (e.g. 72 = invalid session, 73 = offline) (developer.veevacrm.com).
- **Use:** Enables integration with external apps with SSO or API calls via the Salesforce session. (Note: The docs warn this session ID *cannot* authenticate outside; it's specific to enabling certain XHR calls or to pass to custom code (developer.veevacrm.com).)

getSSOAccessToken(authIdentifier, providerName, oldToken)

Purpose: Retrieves a valid Single Sign-On access token for a third-party provider that has been configured in Salesforce, optionally refreshing an existing token. This is useful when MyInsights content needs to call external SSO APIs (developer.veevacrm.com) (developer.veevacrm.com).

- **Usage:**

```
ds.getSSOAccessToken("MyConnName", "ProviderName", existingToken)
  .then(function(resp) {
    console.log(resp.data.token);
  });
```

- **Response:** JSON with `data.token` containing the new access token (developer.veevacrm.com).
- **Parameters:**
 - `authIdentifier` (string): The **DeveloperName** of a Salesforce AuthProvider configuration.
 - `providerName` (string): The friendly name of the SSO provider (often matches `ConnectionUrl`).
 - `oldToken` (string, optional): Existing token to refresh.
- **Returns:** Promise resolving to `{success: true, data: {token: "..."} }`. On failure, error code 62 may indicate "unable to obtain access token" (developer.veevacrm.com).
- **Use:** To facilitate OAuth flows within MyInsights (e.g., connecting to external services from a dashboard).

request(configObject)

Purpose: Perform a generic HTTP request to an **external** URL from MyInsights content (developer.veevacrm.com). This lets MyInsights fetch data from web services. Important: The external server must allow CORS requests from Salesforce origins.

- **Usage:**

```
var cfg = {
  url: "https://api.example.com/data",
  method: "GET",
  headers: {
    "Accept-Language": "en-US"
  },
  timeout: 30,
  expect: "json"
};
ds.request(cfg).then(function(resp) {
  console.log(resp);
});
```

The docs example shows a full config with url, method, headers, body, timeout, etc (developer.veevacrm.com).

- **Parameters:** A config object that can include:
 - url (string): The full URL to call.
 - method (string): HTTP verb (GET, POST, etc).
 - headers (map): Any custom headers.
 - body (string or JSON): Request body (for POST/PUT).
 - timeout (seconds): Max wait time.
 - expect (string): e.g. "text" vs "json", hint on response type.
- **Returns:** A promise resolving to the HTTP response. (The docs do not show exact structure, likely similar to XMLHttpRequest response).
- **Platform Notes:** On Windows (mobile) large downloads may error (developer.veevacrm.com). On online (browser), the body must be a string (use `JSON.stringify` if needed) (developer.veevacrm.com).
- **Use:** Integrate external APIs or images. For example, charts could fetch JSON from a BI system, or translate text via Google Translate API.

Navigation and UI Methods

MyInsights allows a custom dashboard to cause CRM navigation actions. The key methods are:

viewRecord(configObject)

Purpose: Navigate from the current MyInsights context to another record's detail page in the native Veeva CRM UI (developer.veevacrmqa.com) (developer.veevacrm.com). For instance, clicking a button can open a specific call or account in the CRM.

- **Usage:** Provide a config with:
 - object: API name of target object (e.g. "Call2_vod__c").
 - fields: an object specifying identifying field(s) for the record. Usually, an Id or External_ID_vod__c.
 - Optionally target: if multiple MyInsights tabs exist on destination object, specify which tab to open (discussed below).

```
ds.viewRecord({
  object: "Call2_vod__c",
  fields: {
    Id: "a040U000000UJPUQA4"
  }
}).then(function(resp) {
  // The user is navigated to the Call detail page.
});
```

- **Example** (from docs): The snippet shows an HTML button calling `ds.viewRecord` with `object: "Call2_vod__c"` and `fields: {Id: "..."} (developer.veevacrmqa.com)`. This launches the native "Call2" detail page for that record.
- **Parameters:**
 - `configObject.object` (string): API name of object.
 - `configObject.fields` (object): key-value pair of field API names to values that identify the record. Typically `Id` field.
 - `configObject.target` (object or list, optional): Specifies which MyInsights tab to open at destination if multiple exist (developer.veevacrm.com).
- **Behavior:** When called, the CRM UI navigates to the target record's detail page. This is essentially a hyperlink action. The promise resolves when done (or perhaps immediately); in practice UI changes focus.
- **Return:** No data payload; success means navigation happened. The docs say "the application displays the native interface" (developer.veevacrmqa.com).
- **Notes:**
 - If navigating between MyInsights tabs, `target` can include `Id`, `External_Id_vod__c`, or `Studio_Id_vod__c` of the tab to go to (developer.veevacrm.com).
 - Maximum 20 targets can be specified.

viewSection(configObject)

Purpose: Navigate between MyInsights tabs of the *same account or territory* (account-level or territory-level context) (developer.veevacrm.com). Use this to switch tabs rather than leaving the account.

- **Usage:** Similar to `viewRecord`, but only `target` is specified (no `object` as the object is implied to be same).

```
ds.viewSection({
  External_Id_vod__c: "someTabId"
});
```

- **Parameters:**
 - A similar list of identifiers (`Id`, `External_Id_vod__c`, or `Studio_Id_vod__c`) of the MyInsights tabs to switch to.

- **Behavior:** CRM stays on the same account/territory, but switches to the specified tab (like different report sections in a notebook). The docs state it only applies at account/territory level (developer.veevacrm.com).
- **Return:** Similar to `viewRecord`, the UI changes and no JSON returned.

getAvailableObjects()

Purpose: Returns a list of all CRM objects for which the current user has Read access

(developer.veevacrmqa.com) (developer.veevacrm.com). Useful for dynamically populating object selectors.

- **Usage:**

```
ds.getAvailableObjects().then(function(resp) {
  console.log(resp.data); // e.g. {Account: {}, Call12_vod__c: {}, ...}
});
```

- **Response:** The returned JSON has a data section listing object API names as keys, each mapped to an empty object (or metadata) (developer.veevacrmqa.com). Example:

```
{
  "success": true,
  "data": {
    "Sample_Lot_vod__c": {},
    "User": {},
    "Account": {},
    "Contact": {}
  }
  // ... etc
}
```

- **Use:** If you want a UI dropdown of objects or to validate object names, this gives the master list.
- **Docs:** The code example in [46] shows it returning a JSON with data containing object names (developer.veevacrmqa.com).

getObjectMetadata(configObject)

Purpose: Retrieves metadata (field list) for a given object – essentially, returns the fields on which the user has Read access (developer.veevacrmqa.com).

- **Usage:**

```
ds.getObjectMetadata({
  object: "Account"
}).then(function(resp) {
  console.log(resp.Account); // list of fields
});
```

- **Response:** The doc example is truncated, but presumably returns something like an array of fields or a map of {fieldName: fieldLabel}.
- **Use:** To discover object schema at runtime / to build dynamic queries.
- **Notes:** It returns field-level metadata, but we'd need to consult Veeva docs for exact structure. (Not fully shown in our snippet.)

Summary of Key API Methods

The table below summarizes many of the above methods:

Method	Description	Key Parameters	Return/Promise
runQuery(queryConfig)	Query CRM/Sales Data for multiple records with metadata.	queryConfig = {object, fields [], where (str), sort [], limit}.	Promise → { data: [{field:value,...}], fieldLabels, object, name } (developer.veevacrmqa.com)
queryRecord(queryConfig)	Query CRM/Sales Data for a single record. Returns raw key-values (no metadata).	Same queryConfig as runQuery (object, fields, where, etc.).	Promise → raw record object (nested under object name) (developer.veevacrm.com)
querySalesData(queryConfig)	Query Nitro sales summary data (Account-level).	Same as runQuery, object must be Account_Sales_Summary_Insights.	Promise → sales data JSON (includes record_count, list) (developer.veevacrmqa.com)
getDataForCurrentObject(obj, field)	Get single field value of current record context.	object (API name), field (API name).	Promise → { ObjectName: { fieldValue } } (developer.veevacrmqa.com)
getFeedbackData()	Get Align/Territory feedback data (accounts, goals, etc.).	None.	Promise → Feedback JSON (see [14] for fields) (developer.veevacrmqa.com) (developer.veevacrmqa.com)
joinQueries(q1,q2, keyForID,...)	Join two query results on a key (SQL outer join).	q1, q2 (results from runQuery), keyForID string, exclude map, primaryObj	Object → Combined result set (developer.veevacrmqa.com) (developer.veevacrmqa.com)
getFieldLabels(cfg)	Get label translations for a list of fields.	cfg = {object, fields []}.	Promise → e.g. {Object: {Field1:"Label", Field2:"Label"}} (developer.veevacrmqa.com)
getObjectLabels(objectList)	Get singular/plural labels for objects.	Array of object names.	Promise → e.g. {"Account": [{"singular":"Account","plural":"Accounts"}]} (developer.veevacrmqa.com)
getPicklistValueLabels(obj, field)	Get labels for picklist values of a field.	obj string, field string.	Promise → mapping e.g. {FieldApi: {Value1:"Label", Value2:"Label",...}} (developer.veevacrmqa.com)
getVeevaMessagesWithDefault(tokens, locale)	Get translated Veeva message texts.	tokens=array of {msgCategory,msgName}, localeKey string.	Promise → array of message objects (developer.veevacrmqa.com)

Method	Description	Key Parameters	Return/Promise
getSFDCSessionID()	Get current Salesforce session ID and org info.	None.	Promise → {data: {sessionId, instanceURL, isSandbox}} (developer.veevacrm.com)
getSSOAccessToken(id, provider, oldToken)	Get/refresh SSO token for external auth.	authIdentifier, providerName, oldToken.	Promise → {data: {token}} (developer.veevacrm.com)
request(config)	Perform HTTP request to external URL.	Config obj: {url, method, headers, body, timeout, expect}.	Promise → response JSON/text.
viewRecord(cfg)	Navigate to another record's detail page in CRM (native UI).	object, fields (e.g. {Id: "..."}), optional target.	Promise → (UI navigates; no data return).
viewSection(cfg)	Switch MyInsights tab within same account.	cfg includes target tab ID(s).	Promise → (UI changes; no data return).
getAvailableObjects()	List CRM objects user can read.	None.	Promise → {data: {Object1: {}, Object2: {}, ...}} (developer.veevacrmqa.com)
getObjectMetadata(cfg)	List fields user can read for a specific object.	cfg = {object}.	Promise → field metadata (structure not fully shown).

Table 1: Summary of key MyInsights JavaScript API methods (source: Veeva MyInsights Dev Docs ([developer.veevacrmqa.com](#)) ([developer.veevacrmqa.com](#)) ([developer.veevacrm.com](#))). The promise indicates asynchronous call.

The above table is a concise reference. Subsequent sections will dive deeper into usage patterns and examples.

MyInsights Development and Usage Patterns

Having outlined the API, we now discuss how developers typically build MyInsights content and apply the API in practice.

Setting Up MyInsights Content

To create a MyInsights page, one usually:

1. **Prepare HTML/JS files:** This often involves an `index.html` with your layout and references to necessary scripts (MyInsights library, Q, and custom code) ([developer.veevacrm.com](#)). For example:

```
<!DOCTYPE html>
<html>

<head>
<title>MyInsights Page</title>
</head>
```

```

<body>
<div id="chartContainer"></div>
<script src="q.js"></script>
<script src="MyInsightsLibrary.js"></script>
<script src="main.js"></script>
</body>

</html>

```

The `main.js` is your custom logic. 2. **Include Q.js:** The MyInsights library requires the [Q promise library](#) to function. Including `q.js` before `MyInsightsLibrary.js` ensures `window.Q` (or similar) is available ([developer.veevacrm.com](#)). 3. **Use ds methods in code:** In `main.js`, after `DOMContentLoaded`, call `ds.runQuery`, etc. For example:

```

document.addEventListener("DOMContentLoaded", function() {
  ds.runQuery({
    object: "Account",
    fields: ["Name"],
    limit: 1
  })
  .then(function(resp) {
    document.getElementById("chartContainer").innerText = JSON.stringify(resp.data);
  });
});

```

This pattern is shown in Veeva docs ([developer.veevacrmqa.com](#)) ([developer.veevacrmqa.com](#)). 4. **Upload to Veeva:** Package the HTML and JS files (often zipped) and upload via Veeva repository. For developers, the relevant Veeva object is “HTML_Report_vod”; administrators upload the zip there. The entry point is the main HTML. 5. **Configuration:** Admins can configure which Users or Profiles see which MyInsights content, as separate tabs or embedded views (outside this report’s scope).

Once deployed, the MyInsights content runs inside the Veeva CRM application. The API calls made in JS will fetch data from the CRM, Nitro, or external endpoints as coded.

Example: Simple Query in MyInsights

As an illustrative mini-case, consider we want an Account Profile page to display the account name and phone. In `main.js` one might write:

```

function printToScreen(jsonObj) {
  document.getElementById("accountDisplay").innerText = jsonObj;
}

function start() {
  var queryConfig = {
    object: "Account",
    fields: ["Name", "Phone"],

```

```

where: "Account_ID_vod__c = '" + VeevaObjects.recordId + "'",
limit: 1
};
ds.runQuery(queryConfig).then(
function(resp) {
// resp.data [0].Name.display has the Account Name
var name = resp.data[0].Name.display;
var phone = resp.data[0].Phone.display;
printToScreen("Name: " + name + ", Phone: " + phone);
},
function(err) {
printToScreen("Error: " + err);
}
);
}
document.addEventListener("DOMContentLoaded", function() {
start();
});

```

Here, `VeevaObjects.recordId` is a common global in Veeva JS indicating the current record's ID. The WHERE clause uses the account's ID (single quotes needed for browser) (developer.veevacrm.com) (crmhelp.veeva.com). Note the `.display` property yields the string for each field.

Query Syntax and Platform Differences

A frequent developer pitfall is query syntax differences between online (browser) and offline (iPad) modes. Key guidelines include (developer.veevacrm.com) (crmhelp.veeva.com):

- **String Literals:** When writing a where clause in JavaScript, surround values in single quotes ('). Veeva's parser will handle them correctly on browser and mobile (developer.veevacrm.com).
- **Picklist Values:** If filtering by a picklist, use the exact API name of the value (e.g. `Face_to_Face_vod`). Case must match exactly; otherwise queries may fail (developer.veevacrm.com).
- **IDs:** In browser queries, use full SFDC IDs inside quotes; on mobile, IDs must be exactly 15 or 18 characters (no dynamic automatic conversion) (developer.veevacrm.com) (crmhelp.veeva.com).
- **IN Clauses:** The `ds.getInStatement(ids)` method should be used to build an ('id1', 'id2') string for the WHERE clause if matching against a list (developer.veevacrmqa.com). The helper formats the ID list for the query.
- **Operators:** Only basic operators (`=`, `!=`, `<`, `>`, `<=`, `>=`, `IN` maybe) are supported on mobile. Complex SOQL conditions may need to be simplified.

Given these, developers often test both in local simulator and in device. Veeva's help advises to "preview often" to catch mismatches (crmhelp.veeva.com) (crmhelp.veeva.com).

Working with Large Data

For heavy data loads, `runQuery` has similar limits as SOQL; large results may require pagination or where clauses. On mobile, large result sets can slower. The `querySalesData` and `ds.request` notes imply:

- `querySalesData` can return up to many records. Docs suggest using `limit` for practicality.
- External requests using `ds.request` may run into timeouts or memory issues if the payload is very large (developer.veevacrm.com).

MyInsights Studio and Data Elements

Beyond raw code, MyInsights offers a **Studio** UI for building pages. While coding gives full flexibility, Studio allows admins to drag-and-drop Data Elements which generate calls to the underlying API. For example:

- *Get CRM Data*: Equivalent to a `runQuery` configured via UI (select object, fields, filter).
- *Get Sales Data*: Similar to `querySalesData`.
- *Filter List* and *Sort List* correspond to applying `where` or `sort` after data retrieval.
- *Get Veeva Messages* uses `getVeevaMessagesWithDefault`.
- *Get Aligned Territories* is likely using `getAvailableObjects` or a specialized service.

Studio abstracts the API, but the queries it generates must follow the same rules discussed above (crmhelp.veeva.com). For example, the “**Get CRM Data**” element’s `where` clause is written in the same way as for `runQuery`, and the help page warns about quoting and ID formats (crmhelp.veeva.com).

Data elements also introduce *calculate* and *filter* components (e.g., *Filter List*), which internally handle things like `joinQueries` and `calculate` functions. These behave similarly to what a developer would code manually, but in a no-code interface.

Data Analysis: Usage and Impact

While the primary focus here is API details, it is worth noting evidence of MyInsights’ impact, illustrating why mastering the API can be important:

- **Time Savings**: The GSK example reported an average **30-minute per day productivity gain** for field reps using a custom MyInsights engagement planning page (www.veeva.com). This aligns with the goal stated by Veeva that “field teams get the right insights in context” to improve execution (www.veeva.com).
- **Broader Adoption**: Veeva’s 2023 blog describes success stories from GSK and implies wide usage: “since implementing MyInsights three years ago, we’ve made a number of impactful customizations” (www.veeva.com). Dozens of global life sciences companies and their partner consultants have developed MyInsights solutions, signifying industry buy-in.
- **Expert Opinions**: Numerous Veeva partners and customers have praised MyInsights. The 2017 Veeva press release quotes executives saying custom data visualizations allow meeting customer’s specific data needs (www.veeva.com) (www.veeva.com). For instance, AbelsonTaylor noted MyInsights lets them “provide immediate access to key data” and “personalize the experience” (www.veeva.com). This consensus among industry voices (Aktena, Accenture, etc.) highlights MyInsights’ role in improving data-driven decisions.
- **Case Studies**: The IntuitionLabs/Scilex case states the implementation delivered “daily automated updates” and “Unified Data View” via Nitro-powered MyInsights dashboards (intuitionlabs.ai). While IntuitionLabs’ site doesn’t quantify in text, it emphasizes solving “Fragmented Data” issues

(intuitionlabs.ai).

In future sections of this report, we will revisit these themes: for example, how a certain API (like `runQuery`) feeds the metrics in these dashboards, and how efficiency in usage (like quick nav with `viewRecord`) contributes to time savings.

Case Studies and Real-World Examples

To understand how MyInsights API capabilities translate into business value, we consider a few illustrative implementations and scenarios.

Scilex (IntuitionLabs Case Study)

In 2025, Scilex (a pharmaceutical company) engaged IntuitionLabs to build custom MyInsights dashboards integrated with Veeva Nitro. Key points from their case (intuitionlabs.ai) (intuitionlabs.ai):

- **Challenge:** Scilex's field reps faced "fragmented data" across systems, delayed updates (weekly instead of daily), and lacked unified views of prescriber data (intuitionlabs.ai).
- **Solution:** IntuitionLabs built X-Pages and MyInsights dashboards with "custom analytics powered by Veeva Nitro" (intuitionlabs.ai). These dashboards provided real-time (daily) updates and integrated multiple data sources (prescription, claims, CRM data).
- **Benefits:** They reported "significant reduction in pre-call planning time" and unified all data in one place (intuitionlabs.ai).
- While exact figures are not given in the snippet, this narrative underscores how MyInsights (in combination with Nitro) solved real business problems. To deliver this, developers likely relied on the Nitro data queries (e.g. `querySalesData`) and extensive `runQuery` calls in the dashboards. The API enabled easily joining CRM data (via `joinQueries`) with Nitro results to present combined charts and tables.

GSK (Veeva Blog)

In 2023, GSK described how they built various MyInsights pages in Veeva CRM, including an **engagement planning page** (www.veeva.com). Highlights:

- **Implementation:** GSK created a single-view page that "brings together data from several different places in Veeva CRM" for each HCP call (www.veeva.com).
- **Impact:** Field teams reported saving an average of 30 minutes per day in engagement planning with the MyInsights page (www.veeva.com).
- **Development:** The initial page was built with Veeva's Technical Services (likely using `runQuery` on accounts, calls, programs, etc.). Then GSK also built additional pages (HCP affiliations UI, sample inventory display) using MyInsights Studio, emphasizing "didn't need any coding" for some content (www.veeva.com).
- **Usage Tracking:** GSK captured usage analytics from their MyInsights pages – tracking clicks and interactions (presumably via JavaScript calls to Nitro or custom logging) – to iteratively improve content (www.veeva.com).

From a technical viewpoint, GSK's experience highlights:

- How multiple `runQuery` calls feed one UI (pulling, e.g., account plans, call history, sample info).
- Integration of service calls (they likely used `ds.request` to log usage or `ds.getVeevaMessages` for translations).
- Enabling business users (even without coding) to create effective pages (suggesting strong alignment between Studio elements and underlying API calls).
- The imperative of measuring outcomes: they specifically quantified time savings.

We cite GSK's time-saving stat to emphasize MyInsights' ROI (www.veeva.com).

Other Examples

- **Pre-call dashboards:** A common use is a "Pre-call Planner" that queries the account's top products (via `runQuery` and product fields), top HCPs (possibly using partner arrays), recent orders, etc. APIs like `getObjectMetadata` might populate a dynamic filter, `joinQueries` might combine products with prescription data.
- **KOL Profile:** For Key Opinion Leaders, one could use `runQuery` to fetch HCP profile fields, `joinQueries` to combine with call summaries (Action Plans), and then `viewRecord` to allow launching underlying CRM records.
- **Sample Inventory:** As GSK did, showing remaining sample counts. Likely uses `runQuery` or Nitro data to compute available vs given.

In each scenario, we see multiple API calls building the final picture. The MyInsights documentation and samples (many of which we cited) provide building blocks for these patterns.

Technical and Platform Considerations

While MyInsights offers flexibility, developers must consider performance, limits, and best practices:

- **Asynchronous Programming:** `getInStatement(ids)` returns a string for use in a query WHERE clause; use its result directly. Apply `.then()` only to methods documented as returning promises.
- **Handling Errors:** Always include error callbacks. For example, `.then(successFn, errorFn)`. The documentation for each method lists possible error codes (like the session ID errors 72/73 (developer.veevacrm.com)). Logging errors to the console (or displaying) helps debug.
- **Query Limits:** MyInsights uses SOQL under the covers. If a query exceeds the Salesforce limit (e.g. 50k records, or too many fields), it may fail. Use `limit`, filters, or multiple queries. Consider `joinQueries` if needed to combine results without hitting single-query size.
- **Offline vs Online Data:** Understand that iPads use a local data store. As noted, some queries (especially large searches, or unsupported operators) may fail on mobile. Always test on an actual device. Veeva's help suggests "[e]nsure queries are mobile-compatible" (crmhelp.veeva.com).
- **Caching and State:** Data from `runQuery` is fetched fresh (or from local sync). For stability, don't assume results persist across sessions. Re-run queries if needed. For performance, minimize unnecessary calls (e.g., if the same data is needed in multiple charts, store it in a JS variable).

- **Translations and Localization:** If supporting multiple languages, use `getVeevaMessagesWithDefault` for static labels, and perhaps store content in custom metadata that can be translated. The API supports `localeKeys`.
- **Security:** The user's permissions apply. E.g. `getAvailableObjects` only shows what the user can read. Querying for data the user lacks access to will fail – so design UI only to show allowed objects/fields.
- **Versioning:** MyInsights library versions change over time. Content created for v1.0 may break under v2.0 if not updated. Veeva provided the MyInsights v2.0 library and docs (developer.veevacrm.com). Always test against the intended version (the library file should match or specify compatibility).
- **Performance:** Large JSON can slow rendering. Compress layouts, paginate results (`limit` or `Filter List` in Studio), and consider using charts image libraries (`chart.js`, etc.) only after data is ready.

Example: One known quirk is that on the iPad, using `console.log()` is not visible, so developers often need to append errors to the DOM or use alerts. Also, since offline storage can be limited, syncing Nitro data must be targeted (the Nitro sync docs recommend using the current user's ID as a partition key to limit data (nitrohelp.veeva.com)).

Implications and Future Directions

MyInsights has become a key component of the Veeva ecosystem. Looking ahead:

- **Integration with Nexus (Vault CRM):** Veeva is converging CRM and Vault platforms. The GSK blog even renamed MyInsights to *Vault CRM X-Pages* (www.veeva.com), hinting that these embedded analytics will live within the broader Vault technology stack.
- **AI and Advanced Analytics:** The ability to call external services via `ds.request` opens doors for machine learning. For instance, a custom MyInsights widget might call an AI inference API with rep or HCP data and display predictions.
- **Wider Ecosystem Adoption:** The 2017 push to certify partners (www.veeva.com) suggests a growing third-party ecosystem. Specialized providers (like IntuitionLabs, Accenture) continue to build templates and extensions. This can drive innovation (like better chart components, or cross-platform widgets).
- **Competition and Standardization:** Embedded analytics is a trend (e.g. Salesforce Lightning/Einstein). Veeva continues to invest to differentiate (Teams at Pfizer or Novartis may evaluate bench projects). Veeva's emphasis on no-code tools (Studio, certification) suggests a direction toward citizen development, which can accelerate adoption but also means admin oversight is key.
- **Mobile-First Analytics:** MyInsights is designed for iPad use. As tablets become ubiquitous in field sales, expectations rise. Future directions may include more offline optimization (delta sync, incremental queries), or leveraging device capabilities (GPS, camera for receipts integration through the API's external requests).
- **Standards and Governance:** With increasing custom content, companies need governance. Best practices will emerge (naming conventions, performance benchmarks, usage analytics). MyInsights' tracking (as GSK did (www.veeva.com)) will likely become standard – to prioritize active reports and retire stale ones.

In technical terms, we may see new methods or enhancements in the API in future releases (post-2026). While relying on official sources up to the final cut-off, developers should watch Veeva's release notes and developer portal for updates. As of January 2026, the API is robust and well-documented, but the life sciences requirements (compliance, security) may impose future constraints (e.g. encryption, offline only evolving).

Conclusion

Veeva MyInsights's JavaScript API is a comprehensive toolkit that bridges Salesforce/Veeva CRM data with rich, custom client-side experiences. Our deep dive has covered the full spectrum of its capabilities: querying CRM and Nitro data (`runQuery`, `querySalesData`), retrieving labels and metadata (`getFieldLabels`, `getObjectLabels`, etc.), integrating with SSO and external APIs (`getSFDCSessionID`, `request`), and even controlling CRM UI navigation from within a dashboard (`viewRecord`, `getAvailableObjects`). Each function plays a critical role in translating raw data into actionable insights on the field device.

Throughout this report, we have cited Veeva's own technical documentation and related resources to ensure accuracy (developer.veevacrmqa.com) (developer.veevacrmqa.com) (developer.veevacrmqa.com). We have shown how these API calls underpin real-world dashboards that save time and bring together disparate data (www.veeva.com) (intuitionlabs.ai). For practitioners, understanding these API methods in depth enables the creation of powerful tools – from dynamic charts to entire workflow screens – that drive business impact.

Looking forward, MyInsights will likely continue as a strategic element in life sciences cloud solutions. Its API flexibility suggests it will adapt to new data sources (e.g., connected healthcare data, IoT devices) and analytics paradigms (AI/ML). Meanwhile, best practices around query efficiency, offline capability, and user-centric design will remain essential. We recommend that developers stay engaged with Veeva's evolving documentation, participate in the MyInsights developer community, and rigorously test multi-platform scenarios.

In conclusion, the MyInsights JavaScript API transforms Veeva CRM into a platform where “insights [are] at your fingertips” (as Veeva advertises (www.veeva.com)). By adhering to the detailed guidance and examples presented here, technical teams can fully leverage MyInsights to build next-generation analytics experiences for field teams, cementing its role as a cornerstone of data-driven decision-making in the life sciences.

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. <https://www.veeva.com/ap/resources/veeva-crm-myinsights-empowers-field-teams-with-actionable-insights-for-better-execution/>
2. <https://www.veeva.com/blog/how-gsks-field-reps-save-30-minutes-a-day-with-veeva-crm-myinsights/>
3. <https://developer.veevacrmqa.com/doc/Content/myinsights-v2.htm>
4. https://crmhelp.veeva.com/doc/Content/CRM_topics/MyInsightsStudio/AddData.htm
5. <https://intuitionlabs.ai/articles/veeva-nitro-dashboards-data-architecture>
6. <https://www.veeva.com/resources/more-than-125-customers-worldwide-live-on-vault-crm-as-veeva-accelerates-the-industrys-agentic-transformation/>

7. <https://intuitionlabs.ai/articles/veeva-x-pages-dashboard-tutorial>
8. <https://intuitionlabs.ai/case-studies/veeva-myinsights-pharma-analytics>
9. <https://github.com/kriskowal/q>
10. https://developer.veevacrm.com/doc/Content/CRM_topics/Veeva/myinsights.htm
11. https://developer.veevacrm.com/doc/Content/CRM_topics/Veeva/myinsights-veeva.htm
12. <https://nitrohelp.veeva.com/doc/Content/Nitro/MyInsights/ConfigVeevaCRM.htm>
13. https://crmhelp.veeva.com/doc/Content/CRM_topics/MyInsights/MyInsightsAdvFunct/QuerySalesData.htm

THE PUBLISHER

About IntuitionLabs

IntuitionLabs is an AI consulting, custom software development and data engineering firm serving pharmaceutical, biotechnology, medical-device and other life-science organizations. We work with clinical, regulatory, medical-affairs, commercial, quality and IT teams to connect technology decisions with the work people need to accomplish.

AI consulting and adoption

Our AI enablement services cover readiness assessments, use-case selection, governance and policies, team workshops, adoption measurement and ongoing advisory support. We help organizations structure the information layer behind AI: source material, context, permissions and maintained knowledge that make generated answers useful and reviewable. Private LLM inference and hosted AI options support teams evaluating how to operate AI with appropriate control over their data and infrastructure.

Software, data and life-science workflows

IntuitionLabs develops custom software for pharma and biotech, integrates enterprise systems, and builds data engineering and business intelligence solutions. Areas of focus include AI agents, regulatory research, medical writing, medical affairs, CMC information, competitive intelligence and clinical-document workflows. Our eTMF intelligence work includes cross-system reconciliation and inspection-readiness support.

Enterprise platforms and regulated delivery

We provide Veeva services, application support, managed services, integrations and custom applications, alongside enterprise content work involving platforms such as Egnyte. For regulated workflows, our services include GxP enablement, computer-system validation and software development addressing 21 CFR Part 11 requirements. The applicable controls, validation responsibilities and acceptance criteria are defined for each engagement.

Work with IntuitionLabs

Explore [AI enablement](#), [pharma and biotech software development](#), [data engineering and BI](#), and [Veeva services](#). [Contact IntuitionLabs](#) to discuss your workflow, information sources and implementation needs.

IntuitionLabs publishes educational research to help life-science teams make informed technology decisions. Coverage of a product or organization does not imply a client relationship, endorsement or partnership.

Website: <https://intuitionlabs.ai>

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.