Contents

BOE Pro ESI Configuration Guide

Definition

The External System Integration (ESI) feature in BOE Pro enables seamless data retrieval, configuration, and display within BOE text sections. This integration allows administrators to set up and manage external system connections, define search criteria fields for data retrieval, and display retrieved results as new actuals within BOE text sections.

Before continuing with the procedures in this guide, ensure that you have installed the newest version of BOE Pro.

Implement OpenAPI for BOE Pro ESI

The BOE Pro ESI endpoint requirements are defined by OpenAPI documentation at https://boeproesidoc.propricer.com.

It can be implemented in any language.

1. Set Up Your Development Environment

Choose your preferred programming language and set up your development environment. Ensure you have the necessary tools and libraries for building a web server and handling HTTP requests.

2. Create a New Project

Initialize a new project. This might involve creating a new directory and setting up a project structure. For example, in Node.js, you might use npm init to create a new project.

3. Install Required Dependencies

Install any required dependencies for your project. For example, if you're using Node.js, you might install Express for handling HTTP requests.

4. Define the API Endpoint

Create a new file for your API endpoint. Define the /search endpoint as specified in the OpenAPI definition. This endpoint should accept a POST request.

5. Handle Request Headers

Extract the api_key and api_version headers from the incoming request. Ensure that the api_version header is present and valid.

6. Parse the Request Body

Parse the JSON request body to extract the searchCriteria array. This array contains the search criteria specified by the client.

7. Implement Search Logic

Implement the logic to perform the search based on the provided criteria. This might involve querying a database or performing some calculations. The search logic will vary depending on your specific requirements.

8. Return the Response

Construct the response object, including the api_version header and the search results. Return the response to the client with a status code of 200 OK.

9. Test the Endpoint

Test your endpoint to ensure it works as expected. You can use tools like Postman or curl to send requests to your API and verify the responses.

Example Implementation Outline

The following high-level outline shows what the implementation might look like in a generic programming language.

1. Initialize project and set up dependencies.
2. Define the `/search` endpoint.
3. Extract headers (`api_key`, `api_version`).
4. Validate `api_version` header.
5. Parse request body to extract `searchCriteria`.
6. Implement search logic based on `searchCriteria`.
7. Construct response object with `api_version` and results.
8. Return response with status code `200` (200 OK).
9. Test the endpoint.

Example Pseudocode

The information in this guide should be able to help you implement the endpoint in any programming language of your choice.

```
function handleSearchRequest(request, response) {
  // Extract headers
  apiKey = request.headers['api_key'];
  apiVersion = request.headers['api_version'];

  // Validate api_version header
  if (apiVersion == null || apiVersion === '') {
    return response.status(400).send('API version is required.');
  }

  // Parse request body
  searchCriteria = request.body.searchCriteria;

  // Perform search logic
  results = performSearch(searchCriteria);

  // Construct response
  responseObject = {
    api_version: apiVersion,
    results: results
  };

  // Return response
  return response.status(200).json(responseObject);
}

function performSearch(searchCriteria) {
  // Implement search logic here
  // Return sample results for demonstration
  return [
    { WBS: "1.1.1", RateType: "Hours", Date: "2020-01-01", Results: 123.4 },
    { WBS: "1.1.1", RateType: "Hours", Date: "2020-02-01", Results: 122.5 },
    { WBS: "1.1.1", RateType: "Hours", Date: "2020-03-01", Results: 126.1 }
  ];
}
```

Implementing OpenAPI in ASP.NET Code using C#

The following example shows a sample implementation using ASP.NET Core. This code defines a SearchController with a Search endpoint that accepts the headers and request body as specified in your OpenAPI definition. The PerformSearch method is a placeholder where you can implement your actual search logic.

```
using Microsoft.AspNetCore.Mvc;

using System;
using System.Collections.Generic;

namespace ESI_API.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class SearchController : ControllerBase
    {
        [HttpPost]
        public IActionResult Search(
            [FromHeader(Name = "api_key")] string apiKey,
            [FromHeader(Name = "api_version")] string apiVersion,
            [FromBody] SearchRequest request)
        {
            // Validate the api_version header
            if (string.IsNullOrEmpty(apiVersion))
            {
                return BadRequest("API version is required.");
            }

            // Perform the search logic here
            var results = PerformSearch(request.SearchCriteria);

            // Return the results
            return Ok(new
            {
                api_version = apiVersion,
                results = results
            });
        }

        private List<Item> PerformSearch(List<SearchCriteria> searchCriteria)
        {
            // Implement your search logic here
            // This is just a sample implementation
            var results = new List<Item>
            {
                new Item { WBS = "1.1.1", RateType = "Hours", Date = "2020-01-01",
Results = 123.4 },
                new Item { WBS = "1.1.1", RateType = "Hours", Date = "2020-02-01",
Results = 122.5 },
                new Item { WBS = "1.1.1", RateType = "Hours", Date = "2020-03-01",
Results = 126.1 }
            };
            return results;
```

```
        }
    }

    public class SearchRequest
    {
        public List<SearchCriteria> SearchCriteria { get; set; }
    }

    public class SearchCriteria
    {
        public string Name { get; set; }
        public string Value { get; set; }
        public string StartValue { get; set; }
        public string EndValue { get; set; }
    }

    public class Item
    {
        public string WBS { get; set; }
        public string RateType { get; set; }
        public string Date { get; set; }
        public double Results { get; set; }
    }
}
```

Implementing OpenAPI in Node.js using Express

This code sets up an Express server with a /search endpoint that accepts the headers and request body as specified in your OpenAPI definition. The performSearch function is a placeholder where you can implement your actual search logic.

```
const express = require('express');
const app = express();
app.use(express.json());

app.post('/search', (req, res) => {
  const apiKey = req.header('api_key');
  const apiVersion = req.header('api_version');
  const searchCriteria = req.body.searchCriteria;

  // Validate the api_version header
  if (!apiVersion) {
    return res.status(400).send('API version is required.');
  }

  // Perform the search logic here
  const results = performSearch(searchCriteria);

  // Return the results
  res.status(200).json({
    api_version: apiVersion,
    results: results
  });
});

function performSearch(searchCriteria) {
  // Implement your search logic here
  // This is just a sample implementation
  return [
    { WBS: "1.1.1", RateType: "Hours", Date: "2020-01-01", Results: 123.4 },
    { WBS: "1.1.1", RateType: "Hours", Date: "2020-02-01", Results: 122.5 },
    { WBS: "1.1.1", RateType: "Hours", Date: "2020-03-01", Results: 126.1 }
  ];
}

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

Configuration

You must add ESI connections in the BOE Pro WebAPI **appsettings.json** file. This can be done manually or using the Manager tool. The Manager tool is recommended.

Configure with the Manager Tool

1. Open a command prompt dialog.
2. Go to the folder where **BOEProWebAPIManager.exe** is extracted (in the WebAPI folder).
3. At the command prompt, enter: `BOEProWebAPIManger.exe configesi`
4. Enter the information requested.
5. Repeat steps 4 and 5 for as many connections as needed.

Configure Manually

Open the BOE Pro WebAPI **appsettings.json** file in the WebAPI site folder and edit the `ESIConfiguration` section as shown in the following examples.

ApiKey: The API key used to authenticate requests to the API. This field is optional but highly recommended. It should be replaced with your real API key.

- For example, `"ApiKey": "YOUR_API_KEY_HERE"`

Connections: A list of connections to different services or API endpoints.

Name: The name of the connection. This field is a string that identifies the connection.

- For example, `"Name": "ActualRatesConnection1"`

Description: A brief description of the connection. This field provides additional information about the connection and its purpose.

- For example, `"Description": "Connection to the API for retrieving actual hourly rates for project A."`

URL: The URL of the API endpoint for this connection. This should be replaced with the real service URL.

- For example, `"URL": "https://api.projectdata.com/v1/rates/projectA"`

ApiVersion: The version of the API being used for this connection. It is an integer indicating the version. Currently, the version is `1`. This should not be changed until the API changes for BOE Pro.

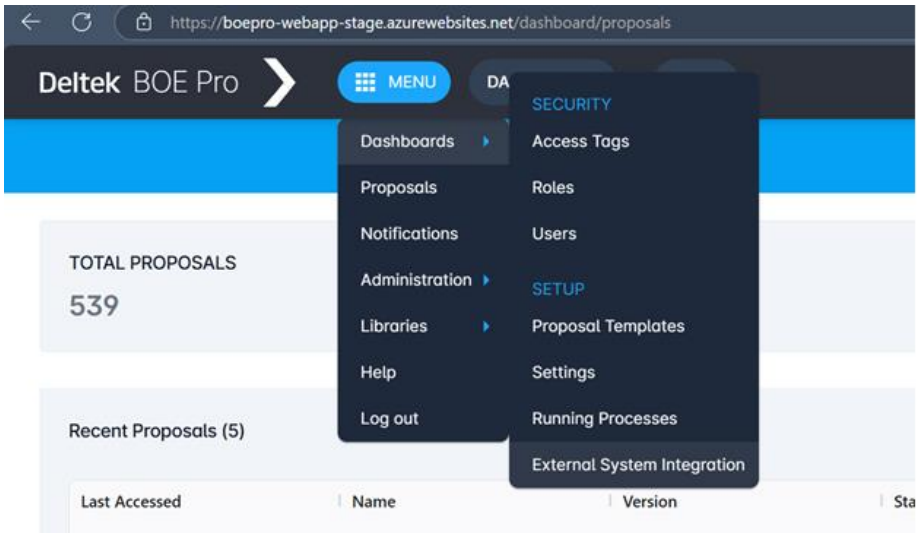
- For example, `"ApiVersion": 1`

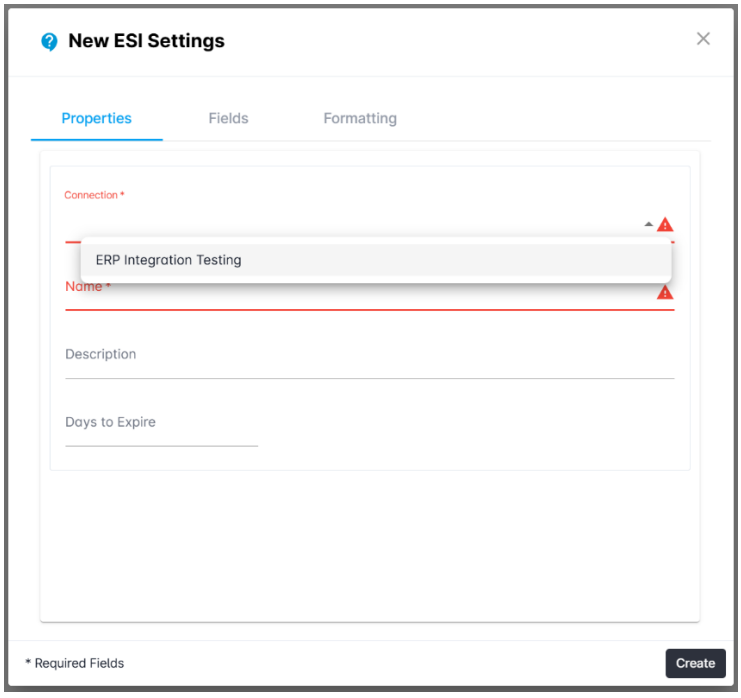
Complete Example of ESIConfiguration Section

```
{
  "ESIConfiguration": {
    "ApiKey": "YOUR_API_KEY_HERE",
    "Connections": [
      {
        "Name": "ActualRatesConnection1",
        "Description": "Connection to retrieve actual hourly rates for project A.",
        "URL": "https://api.actualrates.com/rates/projectA",
        "ApiVersion": 1
      },
      {
        "Name": "ActualRatesConnection2",
        "Description": "Connection to retrieve actual material costs for project B.",
        "URL": "https://api.actualrates.com/costs/projectB",
        "ApiVersion": 1
      }
    ]
  }
}
```

BOE Pro ESI Setup

When you add your desired connections to the **appsettings.json** file, the connections become visible on the Properties tab of the ESI settings in BOE Pro. After completing the settings on the Properties tab and creating an ESI, you can access the settings on the Fields and Formatting tabs.





ESI Properties

Connections A menu showing all connection names specified in the BOE Pro WebAPI **appsettings.json** file within the `ESIConfiguration:Connections` section.

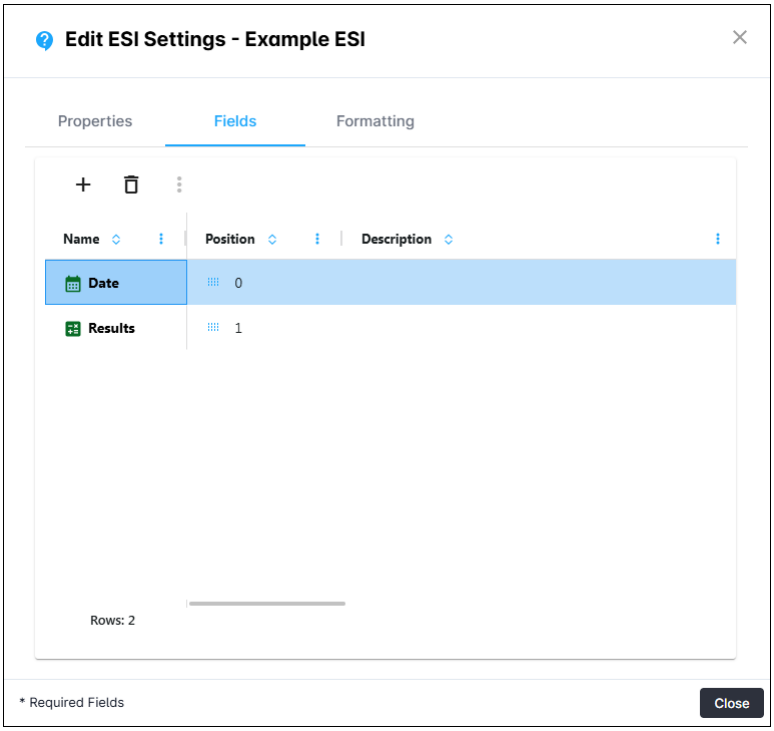
Name The name of the ESI in BOE Pro. It doesn't need to match the selected connection name.

Description Additional information about the ESI.

Days to Expire A numeric value greater than zero that defines how many days an actual from the external system remains valid after it is inserted into the text of a BOE section.

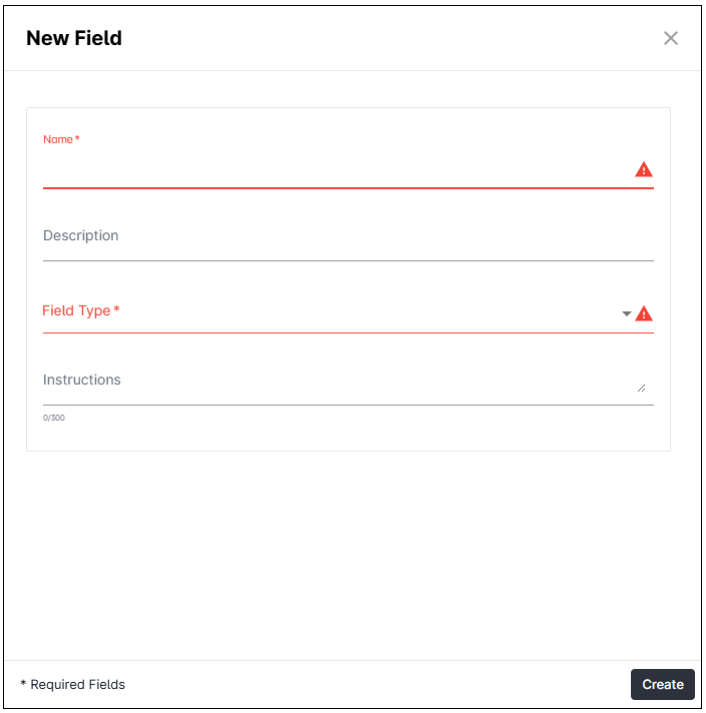
ESI Fields

The Fields tab has two built-in fields: **Date** and **Results**. You can also create more fields. The fields on this tab serve as search criteria fields for requests sent to the external system.



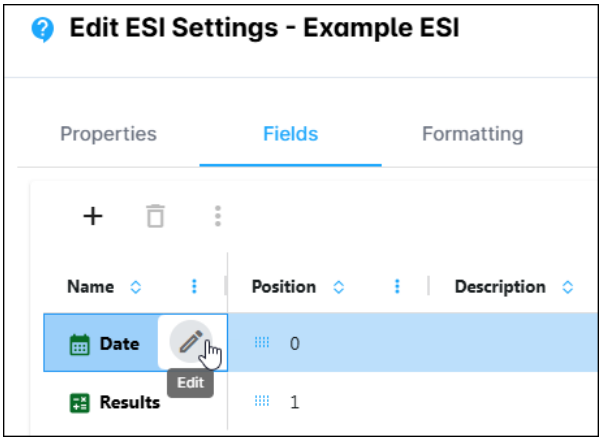
Note: The Date and Results fields are fixed. They can be repositioned but not deleted.

In addition to the **Date** and **Results** fields, you can also create custom search criteria fields.



Edit ESI Fields

You can edit the built-in search criteria fields and any custom fields you create.



Unlike other search criteria fields, the Results field limits what you can edit.

Depending on the type of field, the following information is available for editing:

Name The name of the search criteria field. The field name should match a valid field in the external system you are connecting to.

Description Additional information about the search criteria field.

Field Type

- Text type fields accept simple text values.
- List type fields appear as a dropdown list that only accepts the values defined for it.
- Date type fields allow users to enter or select a calendar date value. Along with the built-in Date field, you can create additional date type fields.

Instructions

Instructions can help people understand how to use the search criteria field in a proposal. Instructions appear as a tooltip when a user hovers over the field. Your instructions can be up to 300 characters long.

Options

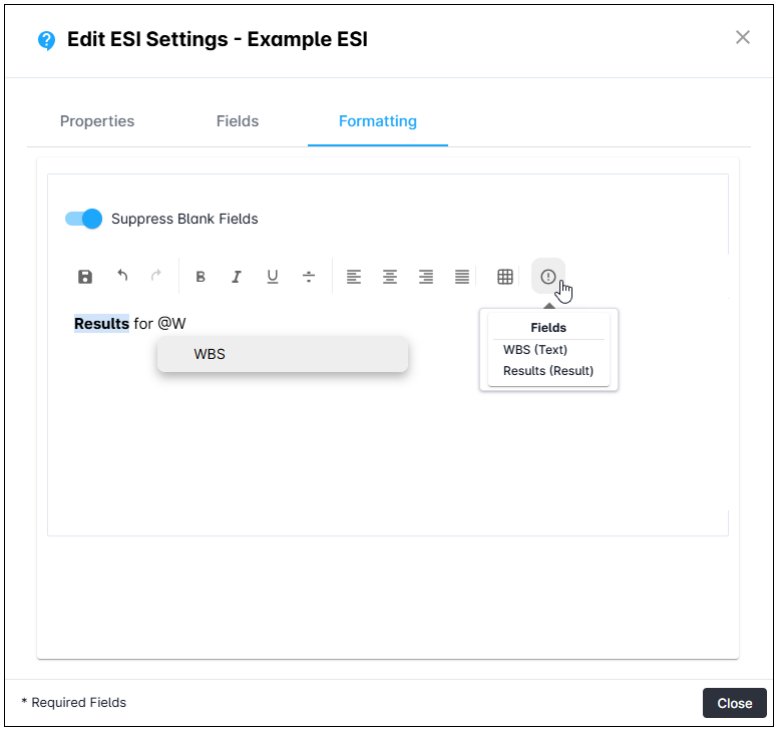
- **Required** determines whether or not users can leave the field empty. A search criteria field that is not required and has no value is sent with a *Null* value to the external system.
- **Queryable** means the field is used as a search criteria field, and it also appears in the results. Disabling **Queryable** prevents the field from being used as a search criteria field, but it is still included in the results data.
- **Display** means the field is used as a search criteria field, and it also appears in the results. Disabling **Display** excludes the field from the results data, but it is still used as a search criteria field.
- **Allow multiple** lets users enter or paste multiple values in a search criteria field at once instead of entering each value one by one.
- **Include in skill mix** means that the search criteria field is included in the skill mix table that users can insert in the text of a BOE section.

ESI Formatting

On the Formatting tab, the **Results** field is available by default. However, you can reference any search criteria field with the **@** character in your text to apply your desired formatting to it.

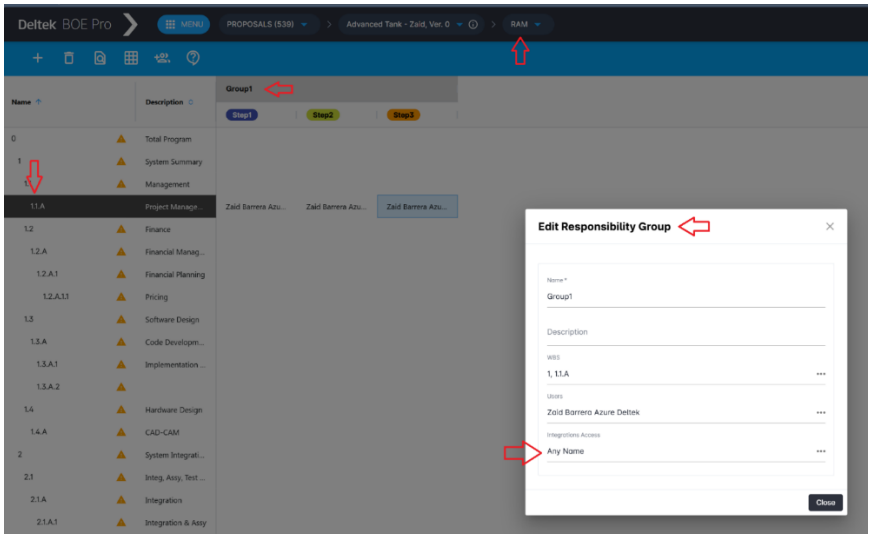
The **Suppress Blank Fields** option hides search criteria fields in a BOE when the fields have no results. Disabling the option allows empty fields to appear in a BOE.

The **Information** button shows which field names you can reference with the **@** character. Any formatting applied is inserted into the BOE text with the final values in place of the criteria fields referenced.

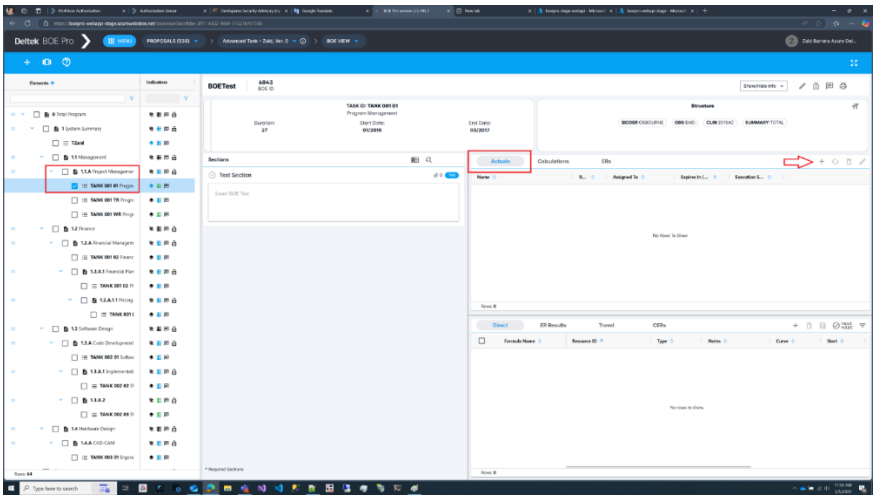


Set Up the RAM

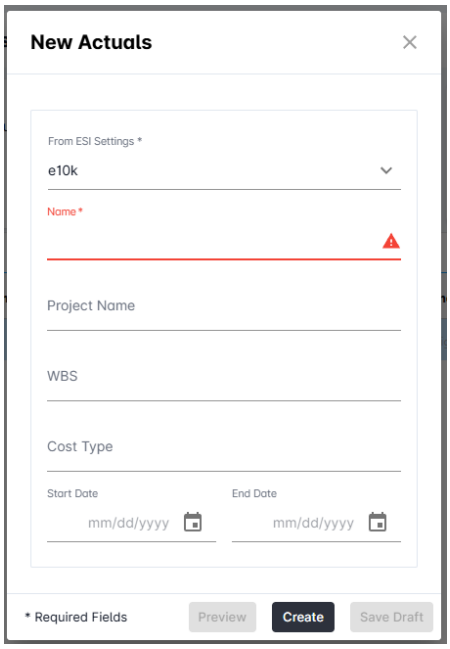
After completing your ESI settings in BOE Pro, you need a well-configured BOE Pro proposal to use the connections. In the proposal, you must edit a Responsibility Assignment Matrix (RAM) group to make connections available at the BOE level.



When at least one connection is assigned to a specific RAM group, a new Actuals tab becomes available in BOE View for that proposal breakdown structure (PBS) element. This is where users can create new lookups by clicking the right-side **+** button.



In this example, the user is creating a lookup from an ESI named *e10k*.



The *e10k* ESI has the following search criteria fields:

- Project Name
- WBS
- Cost Type
- Date

Name identifies the lookup within the BOE. For example, this could be named *e10k-1.1-Labor*, assuming *1.1* is the value for **WBS**, and *Labor* is the value for **Cost Type**.

Once a lookup is created, the user can insert it into the text of a BOE section.