

# Transaction Searches with SuiteScript 2.1

written by [Eric T Grubaugh](#)

*part of the ["SuiteScript by Example" ↗](#) series*

published by [Stoic Software, LLC](#)

# Transaction Searches with SuiteScript 2.1

by [Eric T Grubaugh](#)

Copyright (c) 2017- [Stoic Software, LLC](#). All rights reserved.

Published by Stoic Software, LLC, PO Box 129, Wellington, CO 80549.

NetSuite and SuiteScript are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Neither the author nor the publisher have any affiliation with Oracle Corporation or NetSuite, Inc. This product is neither endorsed nor sponsored by Oracle Corporation or NetSuite, Inc.

## Using Code Samples

This book is here to help you learn. In general, you may use the code presented herein in your own code. You do not need to contact me unless you are reproducing or redistributing large portions of the code.

I appreciate, but do not require, attribution. An attribution usually includes the title, author, and publisher:

*"Transaction Searches with SuiteScript 2.1, by Eric T Grubaugh (Stoic Software, LLC).  
Copyright 2017 Stoic Software, LLC."*

# Introduction

This SuiteScript cookbook is intended to provide you with practical examples for creating transaction searches with the SuiteScript API.

In *Transaction Searches in SuiteScript*, you'll see examples of:

- What are the totals on the 10 most recent Sales Orders?
- How many of each Item were ordered last month?
- What are the *Most Recent* Sales Orders?
- What are the *Oldest* Return Authorizations?
- What are the *Most Recent* Sales Orders, *with the Highest Amounts First*?
- What is the Amount of the Most Recent Sales Order by Customer?
- Which Sales Rep had the Top Transaction in each Month?
- Who is the Contact for the Most Recent Case Filed by a Customer?
- What are all the unapproved Return Authorizations from last month?
- How much On Hand Inventory do I have for each Item?
- What is my Inventory breakdown at a specific Location?

## Patterns in this Book

All code examples are written in *SuiteScript 2.1*.

All code examples in this book use the `require` function for defining modules. This allows you to copy and paste the snippets directly into the debugger or your browser's developer console and run them.

The `N/search` module is always imported as `s`.

`console.log` is used for writing output to the browser console. If desired, you can replace these with calls to the [N/log module](#) for writing to the Execution Log for the debugger.

For more on how to test SuiteScript in your browser's console, watch my [How-To video](#).

# Understanding `mainline`

NetSuite's data model consists of Records, which are split into "body" fields and "sublist" fields. In searches, NetSuite delineates the two with a concept called ["Main Line"](#). Main Line is a Search Filter which allows you to control whether your Search Results contain data from only the body, only the sublists, or both.

Mastering this Main Line concept is absolutely *critical* to mastering transaction searches within NetSuite.

For a detailed walkthrough of how Main Line works, watch [this video](#) (13 minutes).

## What are the totals on the 10 most recent Sales Orders?

```
/**
 * Retrieves the 10 most recent Sales Orders with only body-level
 * data in the Results.
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/search'], (s) => {
  const printOrder = (result) => {
    const transactionNumber = result.getValue({ name: 'tranid' })
    const transactionDate = result.getValue({ name: 'trandate' })
    const transactionTotal = result.getValue({ name: 'total' })

    console.log(`${transactionNumber} on ${transactionDate} for ${transactionTotal}`)

    return true
  }

  s.create({
    type: s.Type.SALES_ORDER,
    filters: [
      ['mainline', s.Operator.IS, true]
    ],
    columns: [
      'total',
      { name: 'trandate', sort: s.Sort.DESC },
      'tranid'
    ]
  }).run().getRange({ start: 0, end: 10 }).forEach(printOrder)
})
```

This example retrieves the Date, Transaction Number, and Total for the ``10`` most recent Sales Orders. These are all body-level fields.


To avoid including any sublist lines in our Results, we leverage the ``mainline`` Filter, which behaves like any other checkbox Filter.

```
filters: [  
  ['mainline', s.Operator.IS, true]  
],
```

When ``mainline`` is ``true``, we get one result per transaction, and only body fields can be included in the ``columns``.

When ``mainline`` is ``false``, we get one result per transaction *line*. Both body- and line-level fields can be included in the ``columns``. The body fields will have their value repeated on each result.

When ``mainline`` is not specified at all, we get one result per transaction *and* one result per transaction *line*. If we don't properly understand ``mainline``, our transaction searches will yield extremely misleading and incorrect Results.



Return To Criteria

Save This Search

+ FILTERS

EDIT   VIEW		INTERNAL ID	*	TRANSACTION NUMBER	DATE ▲	AMOUNT
Edit   View	BODY RESULT  SAME ORDER	18	*	SLS00000101	6-Jan-2015	1,065.20
Edit   View		18		SLS00000101	6-Jan-2015	999.00
Edit   View		18	SUBLIST RESULTS	SLS00000101	6-Jan-2015	12.95
Edit   View		18		SLS00000101	6-Jan-2015	-53.25
Edit   View		26	*	SLS00000102	10-Jan-2015	12,481.90
Edit   View		26		SLS00000102	10-Jan-2015	449.95
Edit   View		26		SLS00000102	10-Jan-2015	27.95
Edit   View		26		SLS00000102	10-Jan-2015	11,996.00
Edit   View		26		SLS00000102	10-Jan-2015	-8.00

Experiment with the ``mainline`` Filter in your Searches to see how it affects the Results.

It is important to note that the ``mainline`` Filter is not supported on Journal Entry searches.

**How many of each Item were ordered last month?**

Conversely, we can leverage the ``mainline`` Filter to avoid matching any *body-level* data in our Results as well so that we *only* get sublist data, but there is a further wrinkle: NetSuite Transactions actually contain *multiple* sublists, not only the Items sublist.

We are often only interested in the Items sublist, though. In order to filter out the other sublists, we need to provide additional Filters. This is shown in the following example, which calculates how many of each Item were ordered last month:

```
/**
 * Retrieves the quantity ordered by Item from Sales Orders created last month.
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/search'], (s) => {
  const printOrder = (result) => {
    const quantity = result.getValue({
      name: 'quantity',
      summary: s.Summary.SUM
    })
    const item = result.getText({ name: 'item', summary: s.Summary.GROUP })

    console.log(`${quantity} ${item} ordered last month.`)

    return true
  }

  s.create({
    type: s.Type.SALES_ORDER,
    filters: [
      ['mainline', s.Operator.IS, false], 'and',
      ['cogs', s.Operator.IS, false], 'and',
      ['shipping', s.Operator.IS, false], 'and',
      ['taxline', s.Operator.IS, false], 'and',
      ['trandate', s.Operator.WITHIN, 'lastMonth']
    ],
    columns: [
      { name: 'item', summary: s.Summary.GROUP },
      { name: 'quantity', summary: s.Summary.SUM }
    ]
  }).run().getRange({ start: 0, end: 1000 }).forEach(printOrder)
})
```

Unfortunately, there is no single Filter that says "Only show me the Item lines". Instead, we have to *explicitly exclude* all *other* sublists (Shipping, Tax, and COGS) by setting their respective Filters to ``false``.

```
filters: [  
  ['mainline', s.Operator.IS, false], 'and',  
  ['cogs', s.Operator.IS, false], 'and',  
  ['shipping', s.Operator.IS, false], 'and',  
  ['taxline', s.Operator.IS, false], 'and',  
  // ...  
],
```

We can also use these same Filters in various combinations when we *are* interested in the data from these sublists. In this way, we can focus in specifically on Shipping, Tax, or COGS for highly targeted reporting and analysis.

# Sorting Search Results

We can sort our Search Results from SuiteScript, just like we can in the UI. This is not limited to Transaction Searches in any way.

We define the sort order of our Results using the [sort property ↗](#) in the `Column` definition. The `N/search` module provides us with a [Sort enumeration ↗](#) for the direction of the sort.

## What are the *Most Recent* Sales Orders?

Sorting can be done in descending order using the `DESC` value from the `Sort` enumeration:

```
/**
 * Retrieves the 10 most recent Sales Orders
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/search'], (s) => {
  const printOrder = (result) => {
    console.log(`${result.id} on ${result.getValue({ name: 'trandate' })}`)
    return true
  }

  s.create({
    type: s.Type.SALES_ORDER,
    filters: [
      ['mainline', s.Operator.IS, true]
    ],
    columns: [
      { name: 'trandate', sort: s.Sort.DESC }
    ]
  }).run().getRange({ start: 0, end: 10 }).forEach(printOrder)
})
```

To find the most recent transaction, we sort by the Transaction Date in descending order:

```
columns: [
  { name: 'trandate', sort: s.Sort.DESC }
]
```

We limit the number of results by specifying a smaller `start` and `end` range to `getRange`:



```
getRange({ start: 0, end: 10 })
```

## What are the *Oldest* Return Authorizations?

Sorting can be done in ascending order using the `ASC` value from the `Sort` enumeration:

```
/**
 * Retrieves the 5 oldest Return Authorizations
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/search'], (s) => {
  const printOrder = (result) => {
    console.log(`${result.id} on ${result.getValue({ name: 'trandate' })}`)
    return true
  }

  s.create({
    type: s.Type.RETURN_AUTHORIZATION,
    filters: [
      ['mainline', s.Operator.IS, true]
    ],
    columns: [
      { name: 'trandate', sort: s.Sort.ASC }
    ]
  }).run().getRange({ start: 0, end: 5 }).forEach(printOrder)
})
```

To find the oldest transaction, we sort by the Transaction Date in ascending order:

```
columns: [
  { name: 'trandate', sort: s.Sort.ASC }
]
```

## What are the *Most Recent* Sales Orders, with the Highest Amounts First?

We can also apply sorting to multiple Columns, and our Results will be sorted by those Columns *in the order in which they are defined*.

```

/**
 * Retrieves the 10 most recent Sales Orders with the highest amounts first
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/search'], function (s) {
  const printOrder = (result) => {
    const date = result.getValue({ name: 'trandate' })
    const total = result.getValue({ name: 'total' })

    console.log(`${result.id} on ${date} for ${total}`)

    return true
  }

  s.create({
    type: s.Type.SALES_ORDER,
    filters: [
      ['mainline', s.Operator.IS, true]
    ],
    columns: [
      { name: 'trandate', sort: s.Sort.DESC },
      { name: 'total', sort: s.Sort.DESC }
    ]
  }).run().getRange({ start: 0, end: 10 }).forEach(printOrder)
})

```

Here we sort by both the `trandate` and the `total` column, so our Results will be ordered first by the date, and then by the total:

```

columns: [
  { name: 'trandate', sort: s.Sort.DESC },
  { name: 'total', sort: s.Sort.DESC }
]

```

If we instead wanted to see the Results sorted first by the highest amount, then by the Date of the order, we would flip the order of the Columns:

```

columns: [
  { name: 'total', sort: s.Sort.DESC },
  { name: 'trandate', sort: s.Sort.DESC }
]

```

# Aggregating Transaction Results Based on Minimal/Maximal Values

There are times when we only care about the values on a Record where a certain field is minimal or maximal. For instance, perhaps we want the Totals of Sales Orders by Customer, but we only care about the *most recent* orders - in other words, where the Date field is maximal.

For situations like this, NetSuite provides us with the [When Ordered By](#) feature of Search Columns.

According to NetSuite Help:

*“The When Ordered By Field option provides search results that return the value for a field when the value for another field is minimal or maximal.”*

We can leverage *When Ordered By* from both the UI and SuiteScript.

If you happen to be familiar with Oracle SQL, *When Ordered By* is the same as ``keep_dense_rank``.

Let's examine several examples of using *When Ordered By*.

## What is the Amount of the Most Recent Sales Order, by Customer?

```

/**
 * Retrieves the Total on the most recent Sales Order for each Customer
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/search'], (s) => {
  const printOrder = (result) => {
    const customer = result.getText({
      name: 'entity',
      summary: s.Summary.GROUP
    })
    const total = result.getValue({
      name: 'totalamount',
      summary: s.Summary.MAX
    })

    console.log(`${customer}'s most recent order: ${total}`)

    return true
  }

  s.create({
    type: s.Type.SALES_ORDER,
    filters: [
      ['mainline', s.Operator.IS, true]
    ],
    columns: [
      { name: 'entity', summary: s.Summary.GROUP },
      s.createColumn({
        name: 'totalamount',
        summary: s.Summary.MAX
      }).setWhenOrderedBy({ name: 'trandate', join: 'x' })
    ]
  }).run().getRange({ start: 0, end: 10 }).forEach(printOrder)
})

```

To add a When Ordered By clause to one of our columns, we use the Column's [setWhenOrderedBy](#) method. Because we need a `Column` instance in order to invoke `setWhenOrderedBy`, we call `s.createColumn()` instead of using one of the shorthand syntaxes.

We want the `totalamount` where `trandate` is most recent (maximal), so we place a `MAX` summary on the `Column` and pass `trandate` to `setWhenOrderedBy`:

```

s.createColumn({
  name: 'totalamount',
  summary: s.Summary.MAX
}).setWhenOrderedBy({ name: 'trandate', join: 'x' })

```

Note the use of `join: 'x'` in `setWhenOrderedBy`; at the time of this writing (NetSuite v2024.1), `join` is a *required* value in `setWhenOrderedBy`, even if your search does not require a join here. I used `'x'` as a nonsensical join name, so it would not be confused with an actual join. It is my hope that this is found to be a bug in the `setWhenOrderedBy` API and is fixed by making `join` optional.

## Which Sales Rep had the Top Transaction in each Month?

```
/**
 * Retrieves the Sales Rep with the largest (by Total) Sales Order
 * in each Period
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/search'], (s) => {
  const printOrder = (result) => {
    const rep = result.getValue({ name: 'salesrep', summary: s.Summary.MAX })
    const period = result.getText({
      name: 'postingperiod',
      summary: s.Summary.GROUP
    })

    console.log(`${rep} had the largest order in ${period}`)

    return true
  }

  s.create({
    type: s.Type.SALES_ORDER,
    filters: [
      ['mainline', s.Operator.IS, true], 'and',
      ['salesrep', s.Operator.NONEOF, '@NONE@'], 'and',
      ['salesrep.isinactive', s.Operator.IS, false]
    ],
    columns: [
      { name: 'postingperiod', summary: s.Summary.GROUP },
      s.createColumn({
        name: 'salesrep',
        summary: s.Summary.MAX
      }).setWhenOrderedBy({ name: 'totalamount', join: 'x' })
    ]
  }).run().getRange({ start: 0, end: 10 }).forEach(printOrder)
})
```

## Who is the Contact for the Most Recent Case Filed by a Customer?

*When Ordered By* is not limited to Transaction searches; we can leverage it in any other Search type as well.

```

/**
 * Retrieves the Contact on the oldest Case filed by each Customer
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/search'], (s) => {
  const printOrder = (result) => {
    const contact = result.getValue({ name: 'contact', summary: s.Summary.MIN })
    const company = result.getValue({
      name: 'company',
      summary: s.Summary.GROUP
    })

    console.log(`${contact} is the Contact on ${company}'s oldest Case.`)

    return true
  }

  s.create({
    type: s.Type.SUPPORT_CASE,
    filters: [
      ['contact', s.Operator.ISNOTEMPTY, '']
    ],
    columns: [
      { name: 'company', summary: s.Summary.GROUP },
      s.createColumn({
        name: 'contact',
        summary: s.Summary.MIN
      }).setWhenOrderedBy({ name: 'createddate', join: 'x' })
    ]
  }).run().getRange({ start: 0, end: 10 }).forEach(printOrder)
})

```

Recall we can use *When Ordered By* for the minimal value of a field as well. To do that, we use a `MIN` Summary on our `Column` instead of `MAX`:

```

s.createColumn({
  name: 'contact',
  summary: s.Summary.MIN
}).setWhenOrderedBy({ name: 'createddate', join: 'x' })

```

When retrieving `Column` values, always remember that the options of your `getValue` call must match the options used to create that `Column`:

```

const contact = result.getValue({ name: 'contact', summary: s.Summary.MIN })

```

# Transaction Status in Searches

Often we need to retrieve or filter by the Status of a Transaction, but Statuses are *notoriously* confusing to work with in SuiteScript. Status fields behave uniquely when compared to any other Select-type field in NetSuite.

Normally, a Select field has a text value that you see in the UI, and a numeric internal ID that you use in SuiteScript. Statuses, however, do not have this numeric ID. Instead, they have a letter identifier, sometimes coupled with the record type.

The examples below will demonstrate how to work with Transaction Statuses in both Search Filters and Columns.

## What are all the unapproved Return Authorizations from last month?

```
/**
 * Retrieves the 10 oldest, unapproved Return Authorizations
 * from last month
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/search'], (s) => {
  const printOrder = (result) => {
    console.group(result.id)
    console.log(`Status Value: ${result.getValue({ name: 'status' })}`)
    console.log(`Status Text: ${result.getText({ name: 'status' })}`)
    console.groupEnd()

    return true
  }

  s.create({
    type: s.Type.RETURN_AUTHORIZATION,
    filters: [
      ['mainline', s.Operator.IS, true], 'and',
      ['trandate', s.Operator.WITHIN, 'lastMonth'], 'and',
      ['status', s.Operator.ANYOF, ['RtnAuth:A']]
    ],
    columns: [
      'status',
      { name: 'trandate', sort: s.Sort.ASC }
    ]
  }).run().getRange({ start: 0, end: 10 }).forEach(printOrder)
})
```

To filter on the unapproved status, we use a Filter on the ``status`` field with a value of ``RtnAuth:A``:

```
[ 'status', s.Operator.ANYOF, ['RtnAuth:A'] ]
```

In the Results, the ``value`` of ``status`` is ``pendingApproval``, while the ``text`` is ``Pending Approval``.

```
result.getValue({ name: 'status' }) // => pendingApproval  
result.getText({ name: 'status' }) // => Pending Approval
```

These same values *do not* work when used as the Filter value; you *must* use the ``RecordType:Letter`` format for the Filter value. Unfortunately, there is no *official* documentation from Oracle or NetSuite that shows the correct identifiers of the various Statuses.

There are several public sources for these values, but note they are all unofficial:

1. [this blog post ↗](#) by DreamXTream from 2011
2. [this blog post ↗](#) by netsuiteerp.com from 2018
3. [this GitHub gist ↗](#) by [W3BGUY ↗](#) from 2020



# Location-Based Inventory Searches

While not technically Transaction Searches, *Inventory* Searches are usually closely tied to Transactions.

In NetSuite, we retrieve Inventory values via Searches on the `Item` record, which has a number of Inventory-specific Filters and Columns.

## How much On Hand Inventory do I have for each Item?

First, NetSuite Items offer a number of columns for determining the total Inventory you have for an Item across all Locations.

```
/**
 * Retrieves the On Hand Inventory for the 10 Inventory Items
 * with the most On Hand quantity
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/search'], (s) => {
  const printOrder = (result) => {
    const item = result.getValue({ name: 'displayname' })
    const onHand = result.getValue({ name: 'quantityonhand' })

    console.log(`${item} on hand: ${onHand}`)

    return true
  }

  s.create({
    type: s.Type.INVENTORY_ITEM,
    filters: [
      ['type', s.Operator.IS, 'InvPart'], 'and',
      ['quantityonhand', s.Operator.ISNOTEMPTY, '']
    ],
    columns: [
      { name: 'displayname' },
      { name: 'quantityonhand', sort: s.Sort.DESC }
    ]
  }).run().getRange({ start: 0, end: 10 }).forEach(printOrder)
})
```

The Search Filters and Columns for total Inventory all start with `quantity*`, e.g. `quantityonhand`, `quantityonorder`, etc.

As always, see the Item page in the [Records Browser](#) for the complete list.

## What is my Inventory breakdown at a specific Location?

Next, NetSuite Items also allow you to retrieve Inventory quantity values for a specific Location using a separate set of Filters and Columns.

```
/**
 * Retrieves the On Hand Inventory for the 10 Inventory Items
 * with the most On Hand quantity
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/search'], (s) => {
  const printOrder = (result) => {
    const item = result.getValue({ name: 'displayname' })
    const onHand = result.getValue({ name: 'locationquantityonhand' })
    const onOrder = result.getValue({ name: 'locationquantityonorder' })
    const backOrder = result.getValue({ name: 'locationquantitybackordered' })

    console.group(item)
    console.log(`On Hand: ${onHand}`)
    console.log(`On Order: ${onOrder}`)
    console.log(`Backordered: ${backOrder}`)
    console.groupEnd()

    return true
  }

  s.create({
    type: s.Type.INVENTORY_ITEM,
    filters: [
      ['type', s.Operator.IS, 'InvtPart'], 'and',
      ['inventorylocation', s.Operator.ANYOF, ['4']], 'and',
      ['locationquantityonhand', s.Operator.ISNOTEMPTY, '']
    ],
    columns: [
      'displayname',
      'locationquantitybackordered',
      'locationquantityonhand',
      'locationquantityonorder'
    ]
  }).run().getRange({ start: 0, end: 10 }).forEach(printOrder)
})
```

The Search Filters and Columns for Location-based Inventory all start with ``locationquantity*``, e.g. ``locationquantityonhand``, ``locationquantityonorder``, etc. See the [Records Browser](#) for the complete list.

In order to use these ``locationquantity*`` Columns, you *must* specify the ``inventorylocation`` Filter in your Search.

# Frequently Asked Questions

## How do I find the details on NetSuite's SQL formulas?

The Help page titled [SQL Expressions ↗](#) contains all the reference material for the supported SQL functions you can utilize.

## How do I find the name/ID for a specific Filter?

Find your Record Type in the [Records Browser ↗](#), and explore the "Search Filters" section. The value in the *Internal ID* column is what you'll use as your Filter name.

## How do I find the name/ID for a specific Column?

Find your Record Type in the [Records Browser ↗](#), and explore the "Search Columns" section. The value in the *Internal ID* column is what you'll use as your Column name.

## How do I find the name/ID for a specific Join?

Find your Record Type in the [Records Browser ↗](#), and explore the "Search Joins" section. The value in the *Join ID* column is what you'll use as your Join name.

## There's a Records Browser, a Schema Browser, and a Connect Browser. What's the difference?

- *Records Browser* - Used for accessing Record data via *SuiteScript*
- *Schema Browser* - Used for accessing Record data via *SuiteTalk*
- *Connect Browser* - Used for accessing Record data via *ODBC*

When you're writing SuiteScript, you can safely focus only on the *Records Browser*.

# Recommendations and Resources

## NetSuite Help

NetSuite Help is the most definitive reference for the `N/search`` module and all of its capabilities. I recommend studying the following articles and any related sub-articles:

- [N/search Module ↗](#)
- [N/search Module Script Samples ↗](#)
- [search.Type ↗](#)
- [search.Summary ↗](#)
- [SuiteScript 2.x Search Operators ↗](#)
- [search.Filter ↗](#)
- [search.filterExpression ↗](#)
- [search.Column ↗](#)
- [search.Operator ↗](#)
- [Summary Type Descriptions ↗](#)
- [Search Date Filters ↗](#)
- [Search.filters ↗](#)
- [search.filterExpression ↗](#)
- [search.Column ↗](#)
- [SQL Expressions ↗](#)

## Use the Search UI

A great way to both learn about and verify your SuiteScript searches is to actually build the search in the UI first, then translate it into SuiteScript.

By doing this, you can quickly verify that the Filters and Columns you're specifying actually give you the correct results before you even start writing code.

## NetSuite Search Export Chrome Plugin

There is a helpful Chrome Plugin called "NetSuite Search Export" built by [in David Smith](#). The Plugin will automatically generate SuiteScript for any Saved Search in your account. You can find it [on the Chrome Plugin Store ↗](#)

To use this plugin:

1. Create a Saved Search in the UI
2. Save the search
3. Click the "Export to Script" link near the top right

## Searching with SuiteScript Playlist

I have a [📺 playlist on YouTube](#) containing several videos and examples of searching in SuiteScript.

## The Records Browser

The [Records Browser ↗](#) is an absolutely crucial tool for creating effective searches. There is a new version of the Records Browser for every version of NetSuite. The 2023.2 version can be found [in NetSuite Help ↗](#).

If you are unfamiliar with the Records Browser, see [SuiteScript Records Browser ↗](#) in the Help documentation and [my tutorial](#).

## Mozilla Developer Network

SuiteScript is a library on top of JavaScript, and the best JavaScript reference manual is the [Mozilla Developer Network ↗](#).

While not related specifically to NetSuite, this site is an excellent source of JavaScript reference material, examples, and tutorials.

# About the Author



My name is [in Eric T Grubaugh](#). I run the *Sustainable SuiteScript* community for NetSuite developers. I founded Stoic Software in 2016 to help others lead successful, sustainable careers as NetSuite developers.

## The "Sustainable SuiteScript" Community

We are a small community of NetSuite developers who want to deepen their technical skills, expand their professional network, and raise the bar for SuiteScript development. [Join us](#) today.

## Questions, Comments, Corrections

If you have any questions, comments, or corrections on this document, please email them to me at [eric+cookbooks@stoic.software](mailto:eric+cookbooks@stoic.software).

## Get in Touch

The best way to keep in regular contact with me is to join the [Sustainable SuiteScript](#) mailing list. I read and respond to all emails I receive there.

I create SuiteScript videos on [YouTube](#).

You can also connect with me on [LinkedIn](#).

