

# Managing Files with SuiteScript 2.1

written by [Eric T Grubaugh](#)

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

published by [Stoic Software, LLC](#)

# Managing Files 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:

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

# Introduction

*Managing Files with SuiteScript 2.1* is intended to provide you with practical examples for interacting with the File Cabinet in your SuiteScript.

In this SuiteScript Cookbook, you'll see examples of:

- Loading an existing file
- Creating a new file
- Relocating a file
- Deleting a file
- Stream data into a file
- Read contents of a file line by line

## Patterns in this Book

All code examples are written in *SuiteScript 2.1*.

The `~/file` module is always imported as `f`.

# Initial Setup

The ``N/file`` module allows us to interact with the contents of NetSuite's File Cabinet.

Before we can get to working with those files, there's a little setup work to do first.

``N/file`` can *only be used in server-side scripts*, meaning we can't drop code in the browser console and expect it to work.

Instead, we're going to build a Suitelet that will interact with our files for us.

We'll start by building a custom module file. This is where we'll be adding and changing the code from the upcoming examples:

```
/**
 * Custom module for executing N/file cookbook examples
 *
 * @NApiVersion 2.1
 * @NModuleScope SameAccount
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
define(['N/file'], (f) => {
  // This is where our example code will go
})
```

We'll walk through this code in detail shortly.

1. Create a folder in the File Cabinet at ``/SuiteScripts/file-cookbook/``
2. Upload the above source code into the new folder in a file named ``file-cookbook.js``.
3. From now on, I will refer to this file as the "``file-cookbook`` module".

```

/**
 * Suitelet for testing File interactions
 *
 * @NApiVersion 2.1
 * @NModuleScope SameAccount
 * @NScriptType Suitelet
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
define(['./file-cookbook', 'N/https'], function (fileCookbook, https) {
  /** @see https://docs.oracle.com/en/cloud/saas/netsuite/ns-online-help/section_4407.html# */
  const onRequest = (context) => {
    log.audit({ title: context.request.method + ' request received' })

    // Ignore POST requests
    if (context.request.method !== https.Method.GET) {
      return
    }

    fileCookbook.readFile(context.response)

    log.audit({ title: 'Request complete.' })
  }

  return { onRequest }
})

```

1. Use the above source code to create a second file in the same folder as before.
2. Use this second file to [create a new `Suitelet`](#) named *File Interaction*.
3. [Create a Deployment](#) for the Suitelet; leave it in **Testing** status.
4. On the Deployment, [add a new `Link`](#) in the **Links** sublist. Locate it somewhere accessible to the Role you'll be using to test the examples in this book. For me, using the Administrator Role, I chose **Classic Center > Setup > Custom > File Interaction**.

# Script Deployment

[Edit](#)[Back](#)[Actions ▾](#)

SCRIPT

Render a PDF

STATUS

Testing

TITLE

Render a PDF

EVENT TYPE

ID

customdeploy\_sl\_render

LOG LEVEL

Debug



DEPLOYED

EXECUTE AS ROLE

Current Role



AVAILABLE WITHOUT LOGIN

URL

</app/site/hosting/scriptlet.nl?script=682&deploy=1>[Audience •](#) [Links •](#) [Execution Log](#) [System Notes](#)

CENTER

SECTION

CATEGORY

LABEL

Classic Center

Setup

Custom

Render a PDF

“⚠ If either of these code files is named differently or is not in the same folder, you will likely receive `MODULE_NOT_FOUND` errors when attempting to access the Suitelet.”

This Suitelet is our test runner for working with files. Whenever we need to test one of our code examples, we access the link for the Suitelet in NetSuite's main navigation, and we can monitor the resulting [Execution Logs ↗](#) on the Suitelet record.

For the remainder of the cookbook, you should not need to make any modifications to the Suitelet or its source code.

# Load contents of an existing File

We'll start by reading the contents of existing files. Add a new function named `readFile` to the `file-cookbook` module:

```
/**
 * Custom module for executing N/file cookbook examples
 *
 * @NApiVersion 2.1
 * @NModuleScope SameAccount
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
define(['N/file'], (f) => {
  const readFile = (response) => {
    const text = [
      f.load({ id: 7825 }).getContents(),
      f.load({ id: 'Cookbook Files/lost.txt' }).getContents(),
      f.load({ id: './help.txt' }).getContents()
    ].join('\r\n')

    response.write({ output: text })
  }

  return { readFile }
})
```

The `N/file` module provides a `load()` method for retrieving existing `File` instances. I've made three different sample files in my File Cabinet to showcase that we can load a file using three different means:

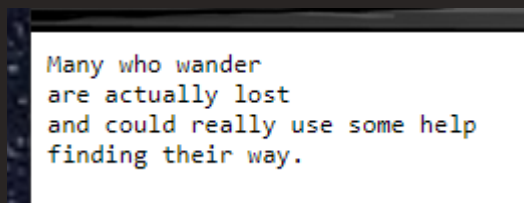
1. by its internal ID: `f.load({id: 7825})`
2. by its absolute path, relative to the File Cabinet root: `f.load({id: 'Cookbook Files/lost.txt'})`
3. by its path relative to the currently executing script: `f.load({id: './help.txt'})`

*"You will need to adjust the IDs and paths for files that exist in your File Cabinet."*

`File` instances have a `getContents()` [method](#) which will return the contents of the `File` as a `string`.

Here we load three `File` instances and place their contents in an Array, then we [join\(\)](#) the Array into a single `string`.

Once you've updated this code with files that exist in your File Cabinet, navigate to the Suitelet via the Link you created on the Deployment. You should see the contents of all three files concatenated and displayed in your browser.



## Other File Properties

Once a `File` has been loaded, there are a few other pieces of information we can retrieve in addition to the contents.

Adjust the `readFile` function like so:

```
const readFile = (response) => {
  const file = f.load({ id: './help.txt' })

  const output = `
    <p>File Size (bytes): ${file.size}</p>
    <p>File Path: ${file.path}</p>
    <p>File URL: ${file.url}</p>
    <p>Is Text Type? ${file.isText}</p>
  `

  response.write({ output })
}
```

Once the `file-cookbook` module is updated, refresh the Suitelet page, and you should see something like:

File Size (bytes): 50

File Path: SuiteScripts/file-cookbook/help.txt

File URL: /core/media/media.nl?id=7827&c=TSTDREV15

Is Text Type? true

See the `File` [object members documentation](#) for details on the properties and methods of the `File` instance.



# Create a new File

We're not limited to working with existing files; we can also create them via script:

```
const readFile = (response) => {
  f.create({
    fileType: f.Type.PLAINTEXT,
    name: 'alliteration.txt',
    folder: -15,
    contents: 'How much wood would a woodchuck chuck'
  }).save()

  response.write({ output: 'File created.' })
}
```

The method for creating files is `file.create()`. The only required parameters are `fileType` and `name`. Note that the `name` should include the file extension explicitly.

In addition, we can directly provide the `folder` where the file should be saved. You must provide the numeric ID of the folder; it cannot be a path.

```
f.create({
  fileType: f.Type.PLAINTEXT,
  name: 'alliteration.txt',
  folder: -15,
  contents: 'How much wood would a woodchuck chuck'
})
```

Once we've created the `File` instance with `create()`, we then invoke its [`save\(\)` method](#) to store it in the File Cabinet.

```
f.create({
  // ...
}).save()
```

## Other Options

There are a few additional parameters we can specify at creation time:

```
const readFile = (response) => {
  f.create({
    fileType: f.Type.PLAINTEXT,
    name: 'alliteration-again.txt',
    folder: -15,
    contents: 'How much wood would a woodchuck chuck',
    description: 'If a woodchuck could chuck wood?',
    encoding: f.Encoding.ISO_8859_1,
    isInactive: false,
    isOnline: true
  }).save()

  response.write({ output: 'File created again.' })
}
```

- ``description`` sets the Description field on the File record when viewing it in the UI
- ``encoding`` sets the Character Encoding of the File using the [`file.Encoding` enumeration ↗](#)
- ``isInactive`` enables (``true``) or disables (``false``) the File's ``inactive`` flag
- ``isOnline`` enables (``true``) or disables (``false``) the ``Available without Login`` setting on the File

See the [`file.create\(\)` documentation ↗](#) for details.

# Move a File

To relocate an existing file to another folder; we set the `File` instance's `folder` [property](#):

```
const readFile = (response) => {  
  const wander = f.load({ id: 7825 })  
  wander.folder = -15  
  wander.save()  
  
  response.write({ output: 'File moved.' })  
}
```

`-15` is the ID for the `SuiteScripts/` folder in all accounts; once this executes, the File `7825` will be relocated to the `SuiteScripts/` directory.

Note you can *only* set a numeric internal ID here; you cannot set the `folder` with a path.

# Delete a File

To delete an existing file, we use the `delete()` [method](#):

```
const readFile = (response) => {  
  f.delete({ id: 7825 })  
  
  response.write({ output: 'File deleted.' })  
}
```

We delete a file using its numeric internal ID.

Note you can *only* specify a numeric internal ID here; you cannot delete a file using a path. (Are you sensing a pattern here? I feel it's a little unfortunate that most of these File APIs do not support paths.)

# Stream data into a File

With the basic mechanics of File operations covered, let's turn to a slightly different way of generating File contents. Say we want to generate a CSV from some data we've retrieved from elsewhere. It could be from a search or an external system or any other source.

```
/**
 * Custom module for executing N/file cookbook examples
 *
 * @NApiVersion 2.1
 * @NModuleScope SameAccount
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
define(['N/file'], (f) => {
  const WeatherData = [
    { date: '01/01/2020', high: 51, low: 19 },
    { date: '01/02/2020', high: 45, low: 27 },
    { date: '01/03/2020', high: 43, low: 20 },
    { date: '01/04/2020', high: 55, low: 22 },
    { date: '01/05/2020', high: 41, low: 26 },
    { date: '01/06/2020', high: 43, low: 30 },
    { date: '01/07/2020', high: 57, low: 31 },
    { date: '01/08/2020', high: 55, low: 23 },
    { date: '01/09/2020', high: 42, low: 26 },
    { date: '01/10/2020', high: 31, low: 13 }
  ]

  const readFile = (response) => {
    const weatherFile = f.create({
      name: 'weather.csv',
      fileType: f.Type.CSV,
      description: 'Stream data to a file, line by line',
      folder: -15
    })

    const lines = WeatherData.map((w) =>
      [w.date, w.low, w.high].join(',')
    )

    lines.forEach((w) => {
      weatherFile.appendLine({ value: w })
    })

    weatherFile.save()

    response.write({ output: weatherFile.getContents() })
  }

  return { readFile }
})
```

## Create the File

We create the `File` almost the same as we did before, except we do not specify any `contents`:

```
const weatherFile = f.create({
  name: 'weather.csv',
  fileType: f.Type.CSV,
  description: 'Stream data to a file, line by line',
  folder: -15
})
```

## Prepare contents

As stated previously, our data could be from any imaginable source; for this example I've created it statically within our script.

```
const WeatherData = [
  { date: '01/01/2020', high: 51, low: 19 },
  { date: '01/02/2020', high: 45, low: 27 },
  { date: '01/03/2020', high: 43, low: 20 },
  { date: '01/04/2020', high: 55, low: 22 },
  { date: '01/05/2020', high: 41, low: 26 },
  { date: '01/06/2020', high: 43, low: 30 },
  { date: '01/07/2020', high: 57, low: 31 },
  { date: '01/08/2020', high: 55, low: 23 },
  { date: '01/09/2020', high: 42, low: 26 },
  { date: '01/10/2020', high: 31, low: 13 }
]
```

From there, we `map` over the Objects, turning each into a comma-separated string of `date,low,high`.

```
const lines = WeatherData.map((w) =>
  [w.date, w.low, w.high].join(',')
)
```

This gives us all the lines of our CSV in an Array, each element of the Array represents one line of the file.

## Stream contents to file, line-by-line

`File` instances have an `appendLine` [method](#) for writing data into them one line at a time.

We already have our Array of `lines`, so we iterate over the Array with `forEach`, and invoke `appendLine()` for each element in the Array.

```
lines.forEach((w) => {  
    weatherFile.appendLine({ value: w })  
})
```

*“Note that `appendLine()` can only be used on Text or CSV file types, and each line can be no more than 10MB. It is also not limited to new files. We can use this method to load an existing file and append data to the end of it without disturbing the original contents.”*

# Read File line by line

Just as we can *write* to a file line by line, we can also *read* line by line:

```
const readFile = (response) => {
  const weatherData = []
  const weatherFile = f.load({ id: 'SuiteScripts/weather.csv' })

  weatherFile.lines.iterator().each((line) => {
    const [date, low, high] = line.value.split(',')
    weatherData.push({ date, low, high })
    return true
  })

  response.write({ output: JSON.stringify(weatherData) })
}
```

Here we load the same CSV file as we created in the previous example:

```
const weatherFile = f.load({ id: 'SuiteScripts/weather.csv' })
```

`File` instances provide us with a [`lines` iterator](#) which we can use to walk the lines of a File one by one.

```
weatherFile.lines.iterator()
```

The Iterator provides an `each()` method which will loop through the lines, passing each line to the callback function we provide.

```
weatherFile.lines.iterator().each((line) => {
  const [date, low, high] = line.value.split(',')
  weatherData.push({ date, low, high })
  return true
})
```

The `line` passed in is an Object, and you can retrieve the contents of the line from its `value` property. Here we do a [naive `split\(\)`](#) on all commas where we assume none of our column values contain commas.

```
const [date, low, high] = line.value.split(',')
```



We use [Array destructuring ↗](#) to assign each element from the Array to a variable. This allows us to use the [shorthand syntax ↗](#) for specifying Object properties next.

```
weatherData.push({ date, low, high })  
return true
```

We then reconstruct the Objects we used initially when we defined `weatherData` and `push()` [the Objects ↗](#) onto the `weatherData` Array.

Your callback function can return `false` to stop or `true` to continue, similar to the way the `each()` iterator works on Search Results. Returning nothing is the same as returning `false`. If you find that your script is only processing the first line of the file, it's likely because you forgot to return `true` here.

Note that the `lines` iterator can only be used on Text or CSV file types, and each line can be no more than `10MB`.

## Why Stream?

It might not be immediately obvious why you would read a file this way. Why not get the entire contents and get to work?

Files can be large; extremely large. Loading the entire contents of a huge file would immediately consume the entire memory limit for your script. Further, you are presumably going to do something interesting with each line of a CSV, and almost anything interesting uses governance. Trying to process a massive file all at once is almost guaranteed to run you into the governance limit for your script.

Instead, you can use this approach to process the file without pushing up against either of these limits, allowing you to stop and check your governance threshold, store your progress for next time, and react accordingly.

You can also use this as a preparatory step to processing large files, chunking them out into smaller, more manageable file sizes.

# Recommendations and Resources

## NetSuite Help

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

- [N/file Module ↗](#)
- [N/file examples ↗](#)
- [Iterators ↗](#)

## 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 Grubbaugh](#). 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 [in LinkedIn](#).

