

Sending Email with SuiteScript 2.1

written by [Eric T Grubaugh](#)

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

published by [Stoic Software, LLC](#)

Sending Email 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:

"Sending Email with SuiteScript 2.1, by Eric T Grubaugh (Stoic Software, LLC). Copyright 2017 Stoic Software, LLC."

Introduction

Sending Email with SuiteScript 2.1 is intended to provide you with practical examples for sending communication via email using SuiteScript's ``N/email`` module.

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

- Sending a simple email
- Sending an email to multiple recipients, including ``cc`` and ``bcc`` functionality
- Sending an HTML email
- Sending an email using a template
- Attaching files to an email
- Linking an email to a NetSuite record

WARNING

This Cookbook contains example code for sending emails - *real, live emails*. If you will be practicing the examples in a Production environment, any email you send from the browser console will actually be delivered to all recipients. *Do not* be the developer that emails your customers an Invoice by mistake.

If at all possible, practice with these examples in a Sandbox or a Preview account.

Sandbox Email Handling

Sandboxes and Release Preview accounts handle email differently so that - similarly to the above warning - test emails *do not* go out to real users or clients. You can control the configuration of email in these accounts by going to ``Setup > Company > Email Preferences > Email Options subtab``.

Your options are:

1. ``SEND EMAIL TO``: All emails, regardless of who you set as recipients/CCs/BCCs, will go to the email address(es) listed here.
2. ``SEND EMAIL TO LOGGED IN USER``: Self-explanatory, but all emails will go to the user in context during the execution of the script. If you're testing in the browser console,

they should come to your inbox.

3. ``DO NOT SEND EMAILS``: No matter what you do, your emails will not show up in *anyone's* inbox. Emails will, however, continue to be linked on the Sender's ``Communications`` subtab.

Work with your Account Administrator to decide the appropriate setting as you test and adapt these examples.

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.

Module aliases:

- The ``N/email`` module is always imported as ``email``.
- The ``N/render`` module is always imported as ``render``.
- The ``N/file`` module is always imported as ``file``.

``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](#).

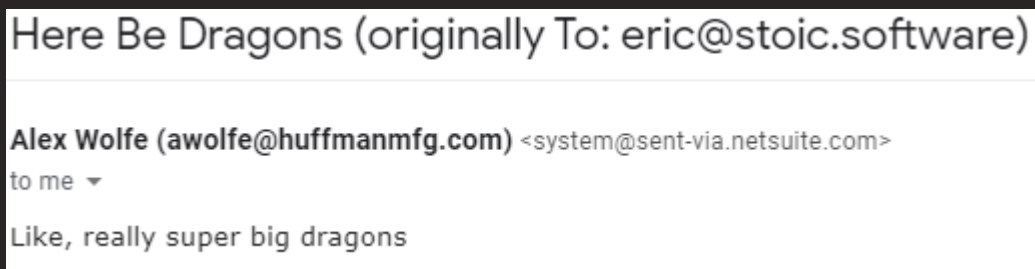
Sending a simple email

The `N/email` module provides the functionality you'll need to send email communications in and out of NetSuite via SuiteScript. All standard email features like sender, recipients, CC, BCC, and file attachments are supported.

Let's start by sending an email with the bare minimum of details. To do so, we use the [send method](#) of the `N/email` module:

```
/**
 * Send a bare minimum email
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/email'], (email) => {
  email.send({
    author: -5,
    recipients: 'eric@stoic.software',
    subject: 'Here Be Dragons',
    body: 'Seriously. Really super big dragons.'
  })
})
```

This results in an email that looks something like this:



To view an email you've sent within NetSuite, navigate to the Employee record of the `author`, and locate the email within the `Communication > Messages` subtab.

`email.send()`

The `send()` method provides everything you'll need to send non-bulk email from SuiteScript. The `send()` method uses `20` governance units and is supported in both Client and Server scripts.

You can see here that four basic options are provided to `send()`:

1. `author`: The sender of the email; this *must* be the internal ID of an active Employee record. You cannot send email out of NetSuite from arbitrary email addresses.
2. `recipients`: The receiver(s) of the email; this can be the internal ID of any Entity record (Lead, Prospect, Employee, etc) or any valid email address as a `string`. You can send to multiple recipients, and you can mix-and-match addresses and internal IDs. We'll look at how we do that in the next example.
3. `subject`: As the name implies, this is the email's subject line.
4. `body`: The text of the email as a `string`. Both plain text and HTML are supported here; we'll look at an HTML example later, and we'll also see how we can merge records into templates and generate the email body.

When you use an internal ID for any recipient, make sure the primary Email Address is populated on the corresponding record; if it is not, the `send()` method will throw an unhelpful `UNEXPECTED ERROR`.

These four parameters are the minimum required values to send an email with the `send()` method. As we continue exploring further uses of the `send()` method, we'll see the other parameter options it provides.

`email.send.promise()`

Do note when you use `N/email` from a Client Script, there is a `Promise` version of the `send()` method: [email.send.promise\(\).](#) While Promises are beyond the scope of this Cookbook, you can find more info [in Help](#) and other links in the *Resources* chapter.

Additional Miscellaneous Options

There are two minor options to `send()` that are straightforward enough that we will not explore them in detail, but only describe them here:

- `isInternalOnly`: A boolean flag that indicates whether the `Message` created by the email will be visible to external Entities like Customers. Defaults to `false`. It may be a good idea to set this to `true` while you are testing.
- `replyTo`: Sets the email's `reply-to` header value. Must be a valid email address.

Sending email to multiple recipients

What good is an email you can only send to one person? How else would you catch people accidentally hitting "Reply All"? The `send()` method supports multiple recipients as well as `cc` and `Bcc` functionality:

```
/**
 * Send an email to multiple recipients, CCs, and BCCs
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/email'], (email) => {
  email.send({
    author: -5,
    recipients: ['eric@stoic.software', 'aragorn@kingme.mid'],
    cc: [17, 'frodo@fly.fool'],
    bcc: ['s@iseeyou.mordor', 21, 'notthegrey@eaglefriends.mid'],
    subject: 'Here Be Dragons',
    body: 'Like, really super big dragons'
  })
})
```

results in an email with a list of recipients:

Here Be Dragons (originally To: eric@stoic.software, aragorn@kingme.middleearth) (originally Cc: Krista Barton <kim@ramsey.com>, frodo@fly.fool) (originally Bcc: s@iseeyou.mordor, Andy Andrews <asnow@ramsey.com>, notthegrey@eaglefriends.middleearth) Inbox x

Multiple recipients

```
recipients: ['eric@stoic.software', 'aragorn@kingme.mid'],
```

The `recipients` property supports the following values:

- a single internal ID of an Entity record (Customer, Employee, Vendor, etc) as a `number`
- a single email address as a `string`
- an Array containing any combination of the above options to send to multiple people.

Here we have an Array containing two email addresses, thus our ``to`` line on the email will have these two recipient addresses.

``cc`` and ``bcc``

The ``send()`` method also provides the ``cc`` and ``bcc`` options to provide the corresponding functionality. These two properties behave identically to the ``recipients`` property in that they accept a single ID or address as well as an Array containing either or both.

```
cc: [17, 'frodo@fly.fool'],
```

will CC both ``frodo@fly.fool`` as well as the Entity record with internal ID ``17``. Similarly,

```
bcc: ['s@iseeyou.mordor', 21, 'notthegrey@eaglefriends.mid'],
```

will BCC three different email addresses: ``s@iseeyou.mordor``, ``notthegrey@eaglefriends.mid``, and the Entity record with internal ID ``21``.

Recipient Limit

Note that the ``send()`` method is limited to a maximum of ``10`` recipients across all of ``recipients``, ``cc``, and ``bcc``.

If you need to send bulk emails to larger audiences, such as with Marketing Campaigns, there are other methods within ``N/email`` for that purpose. This cookbook does not cover those methods, but researching them on your own is a useful exercise.

The methods you'll be interested in researching are:

- [email.sendBulk\(\). ↗](#)
- [email.sendCampaignEvent\(\). ↗](#)

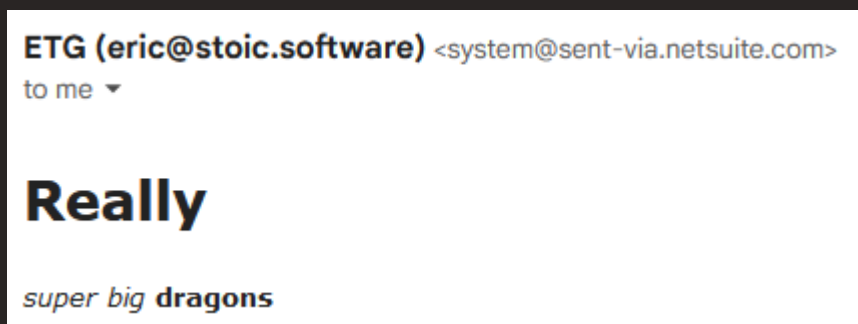
Sending HTML emails

As previously mentioned, we can do quite a lot more with the `body` of our email. For starters, the `body` parameter supports HTML markup:

```
/**
 * Send an email with HTML in the body
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/email'], (email) => {
  const html = `
    <h1>Really</h1>
    <p><em>super big</em> <strong>dragons</strong></p>
  `

  email.send({
    author: -5,
    recipients: 'eric@stoic.software',
    subject: 'Here Be Dragons',
    body: html
  })
})
```

results in a more emphatic email:



That's all there is to it, but know there are better ways to get HTML into your emails aside from writing raw HTML in your code. Instead, leverage Templates to generate the HTML for you.

Sending Email using a Template

Most of the time when we're sending out business email, we want a consistent look and feel along with branding and other official information. We typically use company [Email Templates](#) to accomplish this, and thankfully we can leverage these same Templates in SuiteScript.

The `N/render` [module](#) provides enables us to work with Templates. We import the module and use its `mergeEmail()` [method](#) to generate the subject and body of our email.

```
/**
 * Sends an email using an Email Template
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/email', 'N/render'], (email, render) => {
  const template = render.mergeEmail({
    templateId: 17,
    entity: {
      type: 'employee',
      id: 19
    }
  })

  email.send({
    author: -5,
    recipients: [19],
    subject: template.subject,
    body: template.body
  })
})
```

Usable Email Templates are listed under `Documents > Templates > Email Templates`.

Rendering Email Templates

`render.mergeEmail()` merges specific record data into an existing Email Template to generate the `subject` and `body` of an email.

Template `17` used above has this snippet in it:

```
<p><b>It's Playoffs Time!</b></p>
<p><b>NetSuite invites ${entity.entityId}</b></p>
```

``mergeEmail()`` substitutes data from Employee ``19`` into the email body like this:

A white rectangular box containing two lines of black text. The first line reads "It's Playoffs Time!" and the second line reads "NetSuite invites E0006 Mark Grogan and a guest".

It's Playoffs Time!
NetSuite invites E0006 Mark Grogan and a guest

Note you must use the numeric internal ID of the Template; the more friendly text ID that begins with ``custemailtmp1_`` is unfortunately not supported.

``mergeEmail()`` also supports merging in data from a Support Case (``supportCaseId``), a Transaction (``transactionId``), a Custom Record (``customRecord``), and a Recipient (``recipient``). See the ``render.mergeEmail()`` [documentation ↗](#) for more details.

Sending an Email with an Attachment

What would email be without the ability to attach files? How else would you send the funniest GIFs on the internet to Great Aunt Netta?

Here's how we attach files from the File Cabinet to our emails via SuiteScript:

```
/**
 * Send an email with a file attached
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/email', 'N/file'], (email, file) => {
  email.send({
    author: -5,
    recipients: 'eric@stoic.software',
    subject: 'Here Be Dragons',
    body: 'Really super big dragons',
    attachments: [file.load({ id: 'User Documents/lotr.txt' })]
  })
})
```

“⚠ Note that the `attachments` parameter and the `N/file` module are only supported in server-side scripts. You will need to use this code in the debugger or otherwise adapt it into a server-side script. ⚠”

Two approaches I use to do this are:

1. a `beforeLoad` User Event in Testing mode on a seldom-used record type
2. an Admin- or developer-only Suitelet specifically designed as a harness for testing server-side code

Attaching Files

The `attachments` parameter accepts an Array of `File` instances, so we're free to use any methods available to us to generate a `File` and attach it to our email.

Here we see the simplest method where we retrieve an existing file from the File Cabinet under `/User Documents/lotr.txt` using `file.load()`.

An individual attachment cannot exceed `10MB`; the entire message cannot exceed `20MB`.

Attaching Multiple Files

`attachments` accepts an *Array* of `File` instances:

```
/**
 * Send an email with multiple attachments
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/email', 'N/file'], (email, file) => {
  const createNewFile = () =>
    file.create({
      name: 'unexpected-party.txt',
      fileType: file.Type.PLAINTEXT,
      contents: 'In a hole ...',
      folder: -20
    })

  email.send({
    author: -5,
    recipients: 'eric@stoic.software',
    subject: 'Here Be Dragons',
    body: 'Really super big dragons',
    attachments: [
      file.load({ id: 'User Documents/lotr.txt' }),
      createNewFile()
    ]
  })
})
```

We attach multiple files to our email by adding more elements to the Array:

```
attachments: [
  file.load({ id: 'User Documents/lotr.txt' }),
  createNewFile()
]
```

Observe that we're not just limited to existing files; we can generate a new `File` instance on the fly using `file.create()`, and still attach that to the email without saving to the File Cabinet:

```
const createNewFile = () =>
  file.create({
    name: 'unexpected-party.txt',
    fileType: file.Type.PLAINTEXT,
    contents: 'In a hole ...',
    folder: -20
  })
```

Resulting in an email like:

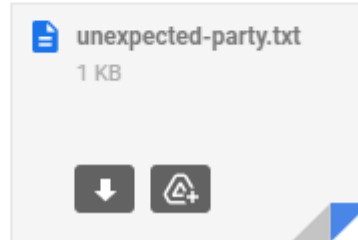
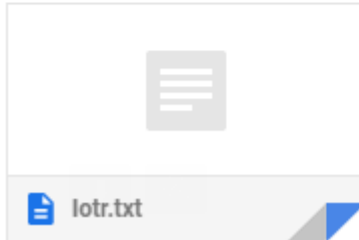
Here Be Dragons (originally To: eric@stoic.software)

Alex Wolfe (awolfe@huffmanmfg.com) <system@sent-via.netsuite.com>

to me ▾

Like, really super big dragons

2 Attachments



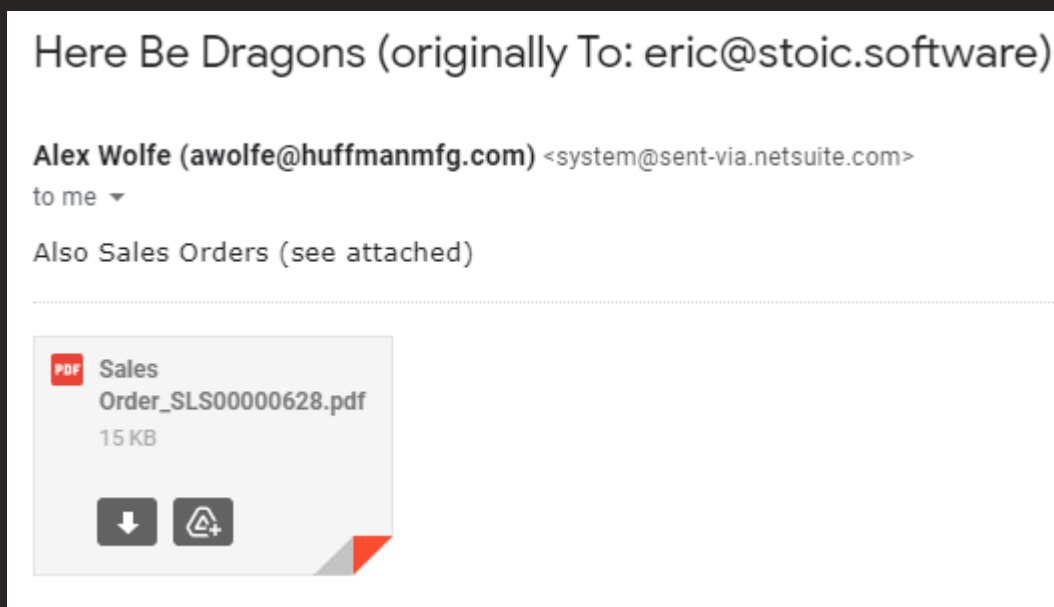
Attaching Records instead of Files

As previously stated, the `attachments` parameter of `email.send()` accepts *any* `File` instance, and in addition to files from the File Cabinet, that can also mean Records. Our old friend the `N/render` module provides us with several methods for turning Records into `File`'s using Advanced PDF/HTML Templates in our account.

```
/**
 * Send an email with the PDF of a Record attached
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/email', 'N/render'], (email, render) => {
  const transactionFile = render.transaction({
    entityId: 8432,
    formId: 75
  })

  email.send({
    author: -5,
    recipients: 'eric@stoic.software',
    subject: 'Here Be Dragons',
    body: 'Also Sales Orders (see attached)',
    attachments: [transactionFile]
  })
})
```

results in an email like:



Rendering Record Attachments

`N/render` provides several methods, like the `transaction()` [method](#) used here, for transforming Records into `File` instances via Advanced PDF/HTML Templates. When we invoke `transaction()`, we provide the internal ID of a Transaction record as the `entityId`.

```
render.transaction({
  entityId: 8432,
  formId: 75
})
```

Which Template NetSuite uses to render the Transaction depends on the Templates defined by the Custom Form we specify for the `formId`. Note here again we need to supply the numeric ID of the Custom Form:

Custom Transaction Form

Save ▼ Cancel Reset | Save & Move Elements

NAME *
Custom Online Order - Invoice

ID

TYPE
Sales Order

EMAIL TEMPLATE
HM Sales order Form ▼

EMAIL MESSAGE TEMPLATE
Default Email Template ▼

☐ ALLOW ADD MULTIPLE

☐ INACTIVE

The `Email Template` field defines which Advanced PDF Template is used to render the Transaction Attachment. The `Email Message Template` field defines which Email Template is used to render the body of the email.

Because `transaction()` returns a `File` instance, we can directly use its output in the `attachments` Array for our email. See the `N/render` [module documentation](#) for the list of additional supported methods and their details.

Linking Emails to Records

When you send an email from a Record in NetSuite's UI, the message is automatically linked to that Record's `Communications` tab for reference later. However, when you send an email via SuiteScript, we have to be a bit more explicit in order to link our email to specific Records.

`email.send()` will automatically link the message to the `author` Employee record as well as any `recipients` specified by Internal ID. If you specify a Recipient with an Email Address, NetSuite will not try to match them up to a Record, and thus will not link the message to that Recipient. In order to link our message to certain Records, `email.send()` provides us with the `relatedRecords` parameter.

```
/**
 * Link an email to multiple Records without attaching those Records
 *
 * @author Eric T Grubaugh <eric@stoic.software> (https://stoic.software/)
 */
require(['N/email', 'N/render'], (email, render) => {
  const salesOrderId = 8432
  const transactionFile = render.transaction({
    entityId: salesOrderId,
    formId: 75
  })

  email.send({
    author: -5,
    recipients: 'eric@stoic.software',
    subject: 'Here Be Dragons',
    body: 'Also Sales Orders (see attached)',
    attachments: [transactionFile],
    relatedRecords: {
      transactionId: salesOrderId,
      customRecord: {
        recordType: 'customrecord_order_notifications',
        id: 17723
      }
    }
  })
})
```

This results in a link on the `Communications` tab of each specified Record instance:

Messages •
Tasks
Phone Calls
Events
User Notes
Files

☐ TO BE PRINTED
☐ TO BE E-MAILED test@tester.com
☐ TO BE FAXED

CUSTOMER MESSAGE
The author and publisher disclaim any warranties (express or implied), merchantability, or fitness for any particular purpose. The author and publisher shall in no event be held liable to any party for any direct, indirect, punitive, special, incidental or other consequential damages arising directly or indirectly from any use of this material, which is provided "as is", and without warranties.

VIEW *
Default

Email Attach Letter PDF Fax Refresh View History Customize View

#	VIEW	DATE ▼	AUTHOR	PRIMARY RECIPIENT	SUBJECT	TYPE	FILES	ATTACHMENTS	INTERNAL ONLY	REMOVE
1	View	2020/10/31 15:40	E0001 Eric T Grubaugh	eric@stoic.software	Here Be Dragons	Email	Yes	Sales Order_SL50000062...	No	Remove

`relatedRecords`

`relatedRecords` is an `Object` structure which provides us the means to link our message to several different types of Records. Here we see an example of linking our message to both the Sales Order we've rendered and attached, as well as linking our message to an instance of a Custom Record we have in our account:

```
relatedRecords: {
  transactionId: salesOrderId,
  customRecord: {
    recordType: "customrecord_order_notifications",
    id: 17723
  }
}
```

See the [`relatedRecords` documentation](#) for the details of our other options for linking to records.

Be aware that for each record type we can link to, we can only link to a single instance of that record type. For instance, since our example already links to a Sales Order, we cannot also link it to another Transaction, like an Invoice. We can only link to one instance of each record type.

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/email Module ↗](#)
- [N/email Examples ↗](#)
- [email.send\(options\).↗](#)
- [email.send.promise\(options\).↗](#)
- [Transaction System Information and Communication Subtabs ↗](#)
- [Promise Object ↗](#)

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.

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](#).

