# Title: Setting up remote choices in Freshsales

### Problem statement:

We have use cases where customer are expecting 50K+ choices for certain custom dropdown fields. As of now, we only support 300 choices for dropdown fields from Product UI.

### Solution:

When such requests come, we handle it from backend offline scripts  for exclusive customers.What we do is create a separate secondary form with this custom field  and have choices only as part of that secondary form.  
Even primary form also will have this field with a reference to secondary form and field. We will be referring secondary form when we need choices for that field using that reference.

Reason why we need to store this field in secondary form is to avoid making primary form heavy as we will accessing Primary form in many places in the application and it will increase the form loading time. It's like side loading or on demand loading kind of approach.

This doc lists out steps to setup remote choices in freshsales with the remote choices being stored in Formservice.

refer this doc [Support for filtering of choices in Formserv](https://confluence.freshworks.com/pages/viewpage.action?pageId=219350855) for the API architecture.

### steps to create remote choices multiselect field

1. find entity form service form id from sequel pro eg it will be in lead form, contact form table etc. copy that service-form-id
2. open [formserv-staging.freshworksapi.com](https://formserv-staging.freshworksapi.com/)  
   3.open Forms tab and in Get tag: retrieve the existing form by pasting the copied service-form-id.
3. Copy the response of the existing form
4. taking that response as base, create a secondary form with a sample multiselect field
5. update that new field with 80000 choices
6. no go to the first form field and update its properties  
   link: /search/auto\_suggest/get\_field\_choices/{secondary\_form\_doc\_id}/{field\_id}  
   field\_options: { ..., "is\_formserv\_link": "true" }

### Things to note

* choices text should be trimmed
* remote choice URL ie link attribute should start with ''/..."
* is\_formserv\_link: should be "true" as a string in field\_options when storing in formserv
* remove all choices from the selected field when setting up the remote choices

### Before running scripts:

1. please ensure you read the README.md before starting
2. change the configurations available at the beginning of each script as needed.

**Scripts:**

* Step 1 - temporarily increase the max allowed choices

**Increase choice limit** Expand source

def increase\_formula\_limit(account\_id)  
 Sharding.select\_shard\_of(account\_id) do  
 account\_record = FdMultitenant::Account.find(account\_id).make\_current  
 acc\_sett = account\_record.account\_setting  
 puts acc\_sett.configs  
 acc\_sett.configs['form\_meta'] = (acc\_sett.configs['form\_meta'] || {}).merge(max\_choices\_allowed\_per\_field: 1000)  
 acc\_sett.save  
 puts acc\_sett.configs  
 end  
end

* Step 2 - create secondary form and copy existing choices over, mark choices as empty in primary form

**create\_secondary\_form** Expand source

def create\_secondary\_form(account\_id, primary\_form\_class, field\_name)  
 Sharding.select\_shard\_of(account\_id) do  
 account = FdMultitenant::Account.find(account\_id).make\_current  
 is\_sales\_360 = account.sales\_360\_account || account.has\_feature?(:sales\_360)  
 puts "is\_sales\_360 #{is\_sales\_360}"  
 form\_key = is\_sales\_360 && ['Contact', 'SalesAccount'].include?(primary\_form\_class) ? FORMSERV\_CONFIG['api\_tokens']['sales\_360'] : FORMSERV\_CONFIG['api\_tokens']['default']  
 puts "form\_key #{form\_key}"  
   
 primary\_form = account.cached\_default\_form\_for(primary\_form\_class)  
 primary\_form\_field = primary\_form.find\_field\_by(:name, field\_name)  
 puts primary\_form\_field.name  
   
 field\_body = ['name', 'column\_name', 'label', 'position', 'type',  
 'custom', 'editable', 'visible', 'deleted',  
 'validatable', 'builder', 'internal', 'field\_class',  
 'placeholder', 'default\_value', 'hint'].each\_with\_object({}) do |property, field\_hash|  
 field\_hash[property] = primary\_form\_field.send(property.to\_sym)  
 end  
   
 field\_body['account\_id'] = account\_id.to\_s  
 field\_body['form\_id'] = '1'  
 # https://github.com/freshdesk/formserv-gem/blob/master/lib/formserv/constants.rb  
 field\_body['type'] = '18'  
 field\_body['choices'] = primary\_form\_field.super\_choices.each\_with\_object([]) do |super\_choice, choices\_list|  
 choices\_list << ['id', 'value', 'position', 'editable', 'custom'].each\_with\_object({}) do |property, choice\_hash|  
 choice\_hash[property] = super\_choice.send(property.to\_sym)  
 end.merge({ 'account\_id' => account\_id.to\_s })  
 end  
   
 secondary\_form\_body = {  
 "account\_id" => account\_id.to\_s,  
 "form\_id" => '1',  
 "coll\_id" => "#{primary\_form\_class}Form",  
 "prod\_id" => "Freshsales",  
 "name" => "#{primary\_form\_class} secondary Form",  
 "title" => "#{primary\_form\_class} secondary Form",  
 "description" => "#{primary\_form\_class} secondary Form",  
 "active" => true,  
 "fields" => [field\_body]  
 }  
   
 secondary\_form\_post\_args = {  
 headers: {  
 'Content-Type' => 'application/json',  
 'Accept' => 'application/json',  
 'Authorization' => form\_key  
 },  
 path: FORMSERV\_CONFIG['formserv\_url'],  
 type: :post,  
 body: secondary\_form\_body  
 }  
 puts "secondary\_form\_post\_args :: #{secondary\_form\_post\_args}"  
 secondary\_form\_response = UtilityServices::Request.new(secondary\_form\_post\_args).send  
   
 if secondary\_form\_response.success?  
 secondary\_form\_response\_body = JSON.parse(secondary\_form\_response.body)  
 secondary\_form\_doc\_id = secondary\_form\_response\_body['docId']  
 secondary\_form\_field\_id = secondary\_form\_response\_body['fields'].find { |fld| fld['name'] == primary\_form\_field.name }['id']  
   
 puts "Secondary Form doc ID :: #{secondary\_form\_doc\_id}"  
 puts "Secondary Field ID :: #{secondary\_form\_field\_id}"  
   
 primary\_field\_get\_args = {  
 headers: {  
 'Content-Type' => 'application/json',  
 'Accept' => 'application/json',  
 'Authorization' => form\_key  
 },  
 path: FORMSERV\_CONFIG['formserv\_url'] + "/#{primary\_form.service\_form\_id}/fields/#{primary\_form\_field.id}",  
 type: :get,  
 body: {}  
 }  
   
 primary\_field\_get\_response = UtilityServices::Request.new(primary\_field\_get\_args).send  
   
 if primary\_field\_get\_response.success?  
 primary\_field\_response\_body = JSON.parse(primary\_field\_get\_response.body)  
 puts "primary\_field\_response\_body :: #{primary\_field\_response\_body}"  
 # Add link  
 primary\_field\_response\_body['link'] = "/search/auto\_suggest/get\_field\_choices/#{primary\_form\_class}/#{secondary\_form\_doc\_id}/#{secondary\_form\_field\_id}"  
 # Remove choices  
 primary\_field\_response\_body['choices'] = []  
 # Alter field options  
 primary\_field\_response\_body['field\_options'] = primary\_field\_response\_body['field\_options'].merge('is\_formserv\_link' => 'true')  
  
 primary\_field\_response\_body['field\_options'].merge('reference' => true)  
   
 primary\_field\_put\_args = {  
 headers: {  
 'Content-Type' => 'application/json',  
 'Accept' => 'application/json',  
 'Authorization' => form\_key  
 },  
 path: FORMSERV\_CONFIG['formserv\_url'] + "/#{primary\_form.service\_form\_id}/fields/#{primary\_form\_field.id}",  
 type: :put,  
 body: primary\_field\_response\_body  
 }  
 puts "primary\_field\_put\_args ::: #{primary\_field\_put\_args}"  
 primary\_field\_put\_response = UtilityServices::Request.new(primary\_field\_put\_args).send  
   
 if primary\_field\_put\_response.success?  
 account.reset\_forms\_cache\_for(primary\_form\_class)  
 account.reset\_forms\_cache\_for(primary\_form\_class.underscore.downcase)  
 puts "SUCCESS!! DONE WITH SECONDARY FORM CREATION AND PRIMARY FIELD UPDATION"  
 else  
 puts "Primary Field PUT failed"  
 end  
 else  
 puts "Primary Field GET failed"  
 end  
 else  
 puts "Secondary Form POST failed"  
 end  
 end  
end

* Step 3 - populate [new script - update using form.update\_field method]

**populate\_choices\_new** Expand source

require 'csv'  
ACCOUNT\_ID = 1651745403  
IMPORT\_FILE = 'file.csv'  
FIELD\_MAPPINGS = {  
 'Country': {  
 'primary\_form': true,  
 'form\_class': 'Contact',  
 'mapped\_field': 'cf\_country\_4',  
 },  
 'State': {  
 'primary\_form': true,  
 'form\_class': 'Contact',  
 'mapped\_field': 'cf\_state\_4',  
 },  
 'City': {  
 'primary\_form': true,  
 'form\_class': 'Contact',  
 'mapped\_field': 'cf\_city\_4',  
 },  
 'Zipcode': {  
 'primary\_form': false,  
 'form\_class': 'Contact',  
 'mapped\_field': 'cf\_zipcode\_4',  
 'form\_doc\_id': '6f98d911-4795-4b40-a500-595b42dfc02d',  
 'form\_field\_id': '95ff8d4b-2dcd-46de-b881-c96b2c631a5c',  
 },  
}  
  
  
  
  
def read\_import\_file(file\_name)  
 csv\_text = File.read(file\_name)  
 csv = CSV.parse(csv\_text, :headers => true)  
 import\_hash = []  
 csv.each do |row|  
 import\_hash << row.to\_hash  
 end  
 import\_hash  
end  
  
  
def write\_json(file\_name, data)  
 File.open(file\_name, 'w') do |f|  
 f.write(data.to\_json)  
 end  
end  
  
  
def filter\_unique\_values(key, field)  
 values = []  
 @items.each do |item|  
 values << item[key.to\_s] if !item[key.to\_s].nil?  
 end  
 values.uniq  
end  
  
  
def get\_primary\_form\_field(field)  
 form = @account.cached\_default\_form\_for(field[:form\_class])  
 return nil if form.blank? || form.nil?  
 form.find\_field\_by(:name, field[:mapped\_field])  
end  
  
  
  
def get\_form\_field(field)  
 if field[:primary\_form]  
 # !\* - fetch form field from primary form  
 form = @account.cached\_default\_form\_for(field[:form\_class])  
 return nil if form.blank? || form.nil?  
 form.find\_field\_by(:name, field[:mapped\_field])  
 else  
 # \* - fetch form field from secondary form  
 is\_sales\_360 = @account.sales\_360\_account || @account.has\_feature?(:sales\_360)  
 form\_key = is\_sales\_360 && ['SalesAccount', 'Contact'].include?(field[:form\_class]) ? FORMSERV\_CONFIG['api\_tokens']['sales\_360'] : FORMSERV\_CONFIG['api\_tokens']['default']  
 headers = { 'Content-Type' => 'application/json', 'Accept' => 'application/json', 'Authorization' => form\_key }  
 req = {  
 headers: headers, type: :get,  
 path: FORMSERV\_CONFIG['formserv\_url'] + "/#{field[:form\_doc\_id]}/fields/#{field[:form\_field\_id]}"  
 }  
 resp = UtilityServices::Request.new(req).send  
 JSON.parse(resp.body)  
 end  
end  
  
  
def save\_form\_choices(mapped\_key, field)  
 mapped\_field = field[:mapped\_field]  
 form\_field = get\_form\_field(field)  
 choice\_ids = {}  
 if field[:primary\_form]  
 form\_field.choices.each do |ch|  
 choice\_ids[ch[:value]] = ch[:id]  
 end  
 else  
 form\_field['choices'].each do |ch|  
 choice\_ids[ch['value']] = ch['id']  
 end  
 end  
 write\_json("#{mapped\_field}.json", choice\_ids)  
end  
  
  
def insert\_into\_form(mapped\_key, field)  
 prefix = '[inserting into form] '  
 mapped\_field = field[:mapped\_field]  
 puts "#{prefix} populating choices for field #{mapped\_key} :: #{mapped\_field}"  
 form\_field = get\_primary\_form\_field(field)  
 return if form\_field.blank?  
  
 values = filter\_unique\_values(mapped\_key, mapped\_field)  
 puts "#{prefix} total new unique choices #{values.length}"  
 return if values.length <= 0  
  
 last\_postion = form\_field.choices.present? ? form\_field.choices.last[:position] || 0 : 0  
 choices\_to\_add = values.map { |ch | { :id => nil, :value => ch, :position => last\_postion += 1 } }  
 # puts "#{prefix} choices\_to\_add #{choices\_to\_add}"  
  
 form = @account.cached\_default\_form\_for(field[:form\_class])  
 form.update\_field(form\_field.id, choices: choices\_to\_add, skip\_callbacks: true)  
 @account.reset\_forms\_cache\_for(field[:form\_class])  
 @account.reset\_forms\_cache\_for(field[:form\_class].underscore.downcase)  
end  
  
  
  
def run\_script()  
 Sharding.select\_shard\_of(ACCOUNT\_ID) do  
 @account = FdMultitenant::Account.find\_by\_id(ACCOUNT\_ID)  
 if @account.blank?  
 puts "cannot find requested account"  
 return;  
 end  
 @account.make\_current  
  
 puts "starting to populate choices ..."  
 @items = read\_import\_file(IMPORT\_FILE)  
 puts "total items in csv file #{@items.length}"  
  
 FIELD\_MAPPINGS.each do |mapped\_key, field|  
 puts "\*" \* 30  
 insert\_into\_form(mapped\_key, field)  
 save\_form\_choices(mapped\_key, field)  
 puts "\*" \* 30  
 end  
  
 puts "completed choices population"  
 end  
end  
  
run\_script()

* Step 4 - update dependencies

**update\_dependencies** Expand source

require 'csv'  
  
ACCOUNT\_ID = 1651745403  
IMPORT\_FILE = 'file.csv'  
FIELD\_MAPPINGS = {  
 'Country': {  
 'form\_class': 'Contact',  
 'mapped\_field': 'cf\_country\_4',  
 'primary\_form': true,  
 'depends': {}  
 },  
 'State': {  
 'form\_class': 'Contact',  
 'mapped\_field': 'cf\_state\_4',  
 'primary\_form': true,  
 'depends': {  
 'csv\_field': 'Country',  
 'form\_class': 'Contact',  
 'mapped\_field': 'cf\_country\_4',  
 'primary\_form': true  
 }  
 },  
 'City': {  
 'form\_class': 'Contact',  
 'mapped\_field': 'cf\_city\_4',  
 'primary\_form': true,  
 'depends': {  
 'csv\_field': 'State',  
 'form\_class': 'Contact',  
 'mapped\_field': 'cf\_state\_4',  
 'primary\_form': true  
 }  
 },  
 'Zipcode': {  
 'form\_class': 'Contact',  
 'mapped\_field': 'cf\_zipcode\_4',  
 'primary\_form': false,  
 'form\_doc\_id': '6f98d911-4795-4b40-a500-595b42dfc02d',  
 'form\_field\_id': '95ff8d4b-2dcd-46de-b881-c96b2c631a5c',  
 'depends': {  
 'csv\_field': 'City',  
 'form\_class': 'Contact',  
 'mapped\_field': 'cf\_city\_4',  
 'primary\_form': true  
 }  
 },  
}  
  
  
def read\_import\_file(file\_name)  
 csv\_text = File.read(file\_name)  
 csv = CSV.parse(csv\_text, :headers => true)  
 import\_hash = []  
 csv.each do |row|  
 import\_hash << row.to\_hash  
 end  
 import\_hash  
end  
  
  
def write\_json(file\_name, data)  
 File.open(file\_name, 'w') do |f|  
 f.write(data.to\_json)  
 end  
end  
  
def read\_json(file\_name)  
 JSON.parse(File.read(file\_name))  
end  
  
  
def filter\_unique\_values(key, field)  
 values = []  
 @items.each do |item|  
 values << item[key.to\_s] if !item[key.to\_s].nil?  
 end  
 values.uniq  
end  
  
# ! controlling field is in primary form  
def generate\_field\_dependency(mapped\_key, field)  
 prefix = '[primary form] '  
 depends = field[:depends]  
 mapped\_field = field[:mapped\_field]  
 if depends.nil? || depends.empty?  
 puts "#{prefix} dependency update not required for #{mapped\_key} :: #{mapped\_field}"  
 return  
 end  
 puts "#{prefix} updating dependency for #{mapped\_key} :: #{mapped\_field}"  
  
 choice\_ids = read\_json("#{field[:mapped\_field]}.json")  
 # puts "#{prefix} choice\_ids #{choice\_ids}"  
  
 csv\_field = depends[:csv\_field]  
 parent\_mapped\_field = depends[:mapped\_field]  
 puts "#{prefix} parent field :: #{csv\_field} :: #{parent\_mapped\_field}"  
  
 results = {}  
 @items.each do |item|  
 if results[item[csv\_field.to\_s]].nil?  
 results[item[csv\_field.to\_s]] = [item[mapped\_key.to\_s]]  
 else  
 results[item[csv\_field.to\_s]] << item[mapped\_key.to\_s]  
 end  
 end  
 results.each do |c\_field, d\_choices|  
 results[c\_field] = d\_choices.map { |ch| choice\_ids[ch] }  
 results[c\_field] = results[c\_field].uniq  
 end  
 # puts "#{prefix} dependency results #{results}"  
  
 if depends[:primary\_form]  
 puts "#{prefix} controlling field is in the primary form"  
  
 form = @account.cached\_default\_form\_for(field[:form\_class])  
 form\_field = form.find\_field\_by(:name, field[:mapped\_field])  
 parent\_field = form.find\_field\_by(:name, depends[:mapped\_field])  
 puts "#{prefix} parent field :: #{parent\_field.id}"  
 choices = []  
 parent\_field.choices.map do |ch|  
 if results[ch[:value]].present?  
 ch[:dependent\_ids] = {  
 choice: results[ch[:value]],  
 field: []  
 }  
 end  
 choices << ch  
 end  
  
 request = {  
 field\_dependency\_configuration: {  
 dependent\_field: {  
 id: form\_field.id  
 },  
 controlling\_field: {  
 id: parent\_field.id,  
 choices: choices,  
 field\_options: {  
 dependent\_choice\_ids\_for\_none: '',  
 dependent\_field\_ids\_for\_none: '',  
 dependent\_required\_field\_ids\_for\_none: '',  
 unique: 'false'  
 },  
 },  
 }  
 }  
 # puts "request #{mapped\_field}\_request.json :: #{request}"  
 write\_json("#{mapped\_field}\_request.json", request)  
  
 config = @account.field\_dependency\_configurations.new  
 config.attributes = {  
 controlling\_field\_id: parent\_field.id,  
 dependent\_field\_id: form\_field.id,  
 form\_class: field[:form\_class],  
 form\_id: form.id  
 }  
 puts "creating dependency now .."  
 # puts "choices : #{choices}"  
 res = config.save\_and\_create\_mapping\_in\_form(field\_options: request[:field\_dependency\_configuration][:controlling\_field][:field\_options],  
 cf\_choices: request[:field\_dependency\_configuration][:controlling\_field][:choices])  
 puts "dependency result : #{res}"  
 @account.reset\_forms\_cache\_for(field[:form\_class])  
 @account.reset\_forms\_cache\_for(field[:form\_class].underscore.downcase)  
 else  
 puts "#{prefix} controlling field is in the secondary form"  
 end  
end  
  
  
  
def run\_script()  
 Sharding.select\_shard\_of(ACCOUNT\_ID) do  
 @account = FdMultitenant::Account.find\_by\_id(ACCOUNT\_ID)  
 if @account.blank?  
 puts "cannot find requested account"  
 return;  
 end  
 @account.make\_current  
 @admin\_user = @account.user\_with\_higesh\_privilege  
 if @admin\_user.blank?  
 puts "cannot find admin user"  
 return;  
 end  
 @admin\_user.make\_current  
  
 puts "starting to updated dependencies ..."  
 @items = read\_import\_file(IMPORT\_FILE)  
 puts "total items in csv file #{@items.length}"  
  
 FIELD\_MAPPINGS.each do |mapped\_key, field|  
 puts "\*" \* 30  
 generate\_field\_dependency(mapped\_key, field)  
 puts "\*" \* 30  
 end  
  
 puts "completed updating dependencies"  
 end  
end  
  
run\_script()

* Step 5 - manually update zipcode choice ids

**update\_zip\_ids** Expand source

def update\_choices(account\_id, model, field\_name, choices)  
 Sharding.select\_shard\_of(account\_id) do  
 account = FdMultitenant::Account.find(account\_id)  
 account.make\_current  
 form = FdMultitenant::Account.current.cached\_default\_form\_for(model)  
 field = form.find\_field\_by(:name, field\_name)  
 return if field.blank?  
  
 form.update\_field(field.id, choices: choices, skip\_callbacks: true)  
 puts "updated field choices"  
 account.reset\_forms\_cache\_for(model)  
 account.reset\_forms\_cache\_for(model.underscore.downcase)  
 end  
end  
  
  
def read\_json(file\_name)  
 JSON.parse(File.read(file\_name))  
end  
  
choices = read\_json('cf\_zipcode\_1\_request.json')  
  
update\_choices(1648123656, 'SalesAccount', 'cf\_city\_1', choices['field\_dependency\_configuration']['controlling\_field']['choices'])

* Step 6 - add reference option to field

**add\_reference\_option** Expand source

def update\_field(account\_id, model, field\_name)  
 Sharding.select\_shard\_of(account\_id) do  
 account = FdMultitenant::Account.find(account\_id)  
 account.make\_current  
 form = account.cached\_default\_form\_for(model)  
 field = form.find\_field\_by(:name, field\_name)  
 return if field.blank?  
  
 puts "field\_options #{field.field\_options}"  
  
 options = field.field\_options  
 options['reference'] = true  
 res = form.update\_field(field.id, field\_options: options)  
 puts "result #{res}"  
 account.reset\_forms\_cache\_for(model)  
 account.reset\_forms\_cache\_for(model.underscore.downcase)  
 end  
end  
  
update\_field(1648123656, 'SalesAccount', 'cf\_zipcode\_1')

* README.md

**README.md** Expand source

0. create required fields through SSO  
  
1. increase the formula limit for the customer using increase\_formula\_limit.rb script  
  
2. run the set\_remote\_choices.rb on zipcodes field  
  
3. change the configs in the populate.rb and run  
  
4. change the configs in dependency.rb and run  
  
5. copy the request.json name for zipcodes and run the update\_ids.rb script  
  
6. add reference: true in the field options using add\_reference\_option.rb script  
  
  
Atom BE things to comment out before running scripts  
  
lib/form\_service/common\_field\_methods.rb:19  
 if field.custom && (field.dom\_type.in?(%i[dropdown checkbox radio]) || (field.field\_class == 'Product' && field.type.to\_sym.in?(%i[dropdown checkbox radio]))) && !(field.respond\_to?(:has\_remote\_choices?) && field.has\_remote\_choices?) # fields with remote choices not supported in field-dep by formserv for now!  
  
  
lib/form\_service/common\_field\_methods.rb:35  
 return false if !field.editable || field.default\_dropdown\_as\_autocomplete? || (field.respond\_to?(:has\_remote\_choices?) && field.has\_remote\_choices?) # fields with remote choices not supported in field-dep by formserv for now!

[file.csv](/display/freshsales/Setting+up+remote+choices+in+Freshsales?preview=%2F223786168%2F362491365%2Ffile.csv)