Improve Angular solid application handling with multiple values

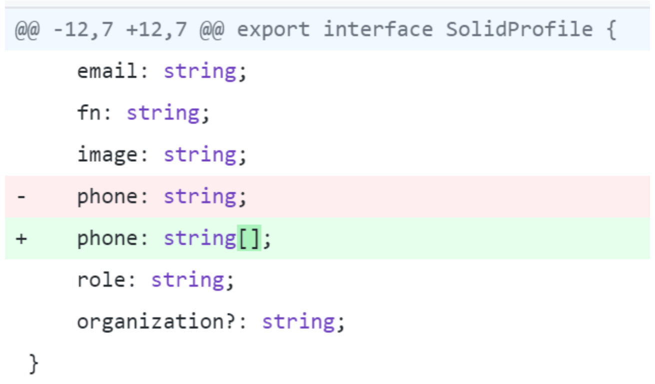
2018/11/13

Xinglong Jia

As I mentioned in yesterday interview, in Solid Docs, I find some problems on the demo project Writing solid applications with angular . The most important or uncomplete function in the application is this application doesn't handle fields with multiple values. In this document, I will introduce my simple approach to solve this problem to you. Hope this can have some effects to improve the solid angular framework.

In all picture, red lines mean deleted codes, and green lines new codes.

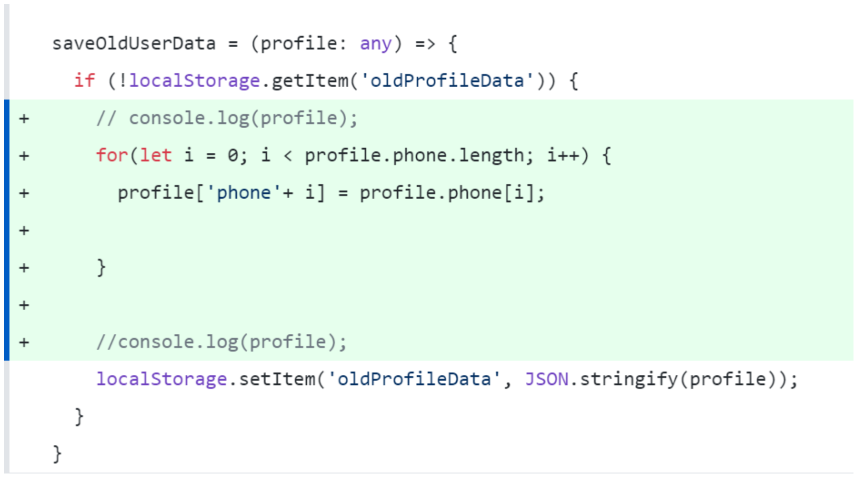
To begin with this approach, the first thing is to modify solid profile model. Change the previous



```
@@ -12,7 +12,7 @@ export interface SolidProfile {
      email: string;
      fn: string;
      image: string;
-     phone: string;
+     phone: string[];
      role: string;
      organization?: string;
  }
```

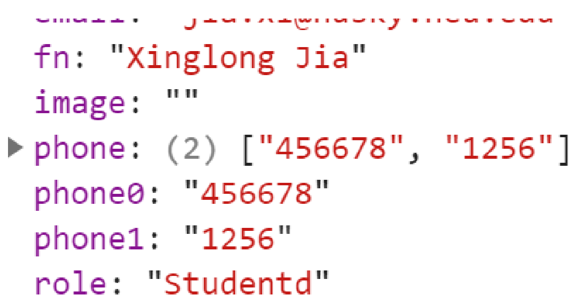
phone property type from string to string array, which can help me to store multiple phone numbers from the WebID.

To better compare the previous information with the new information, the application use local storage to store the old profile as 'oldProfileData'. As the phone property become a string array, here I divide the array into different property, names as phone0, phone1….



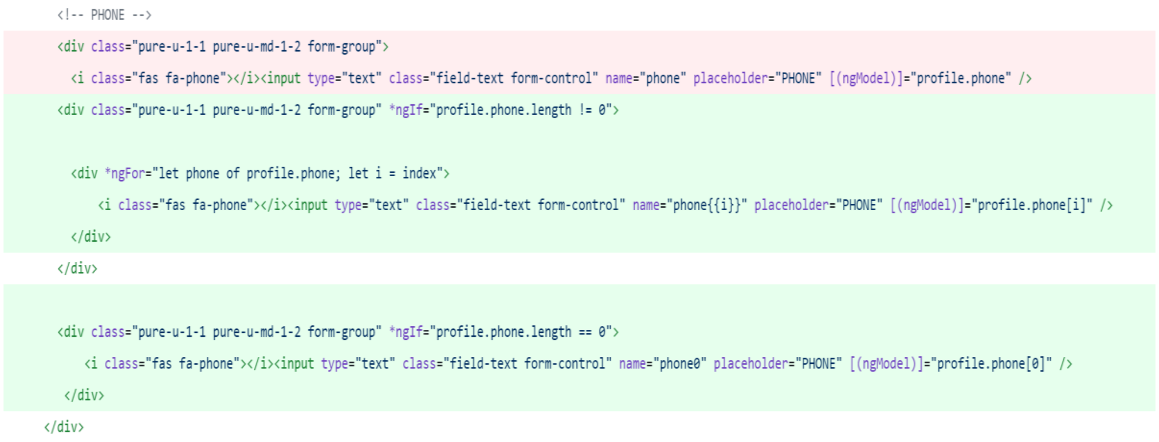
```
saveOldUserData = (profile: any) => {
  if (!localStorage.getItem('oldProfileData')) {
+     // console.log(profile);
+     for(let i = 0; i < profile.phone.length; i++) {
+       profile['phone'+ i] = profile.phone[i];
+
+     }
+
+     //console.log(profile);
      localStorage.setItem('oldProfileData', JSON.stringify(profile));
  }
}
```

And when log out the saved old profile data in local storage, there will be different elements for every phone number.

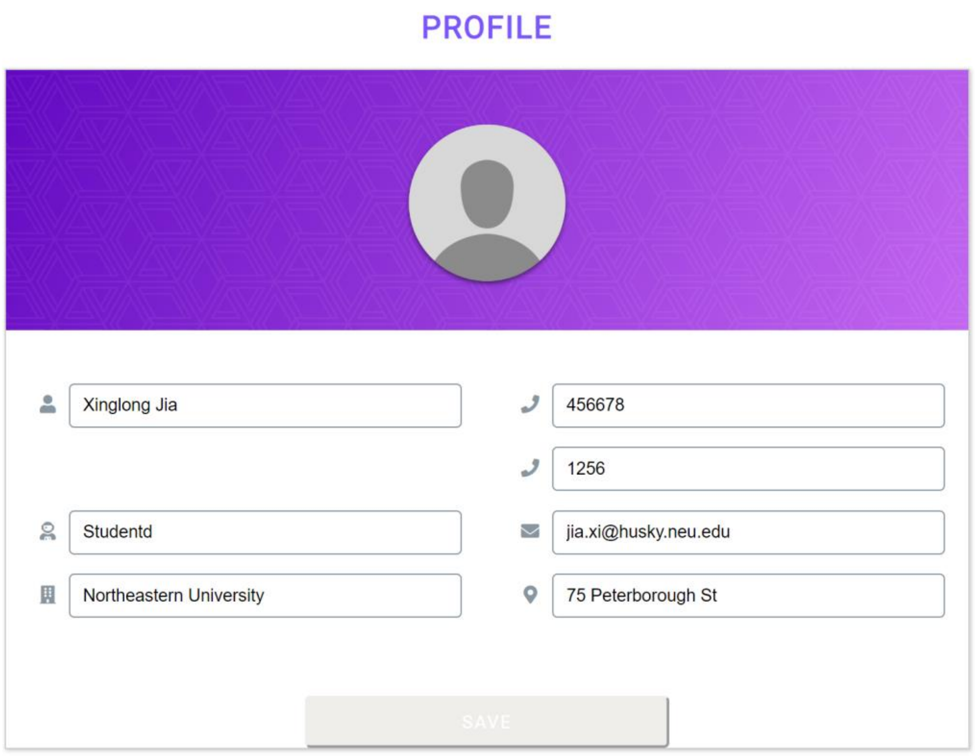


```
fn: "Xinglong Jia"
image: ""
▶ phone: (2) ["456678", "1256"]
phone0: "456678"
phone1: "1256"
role: "Studentd"
```

In card component, since profile.phone is string array, I use ng for to iterate through all the phone, and use ng if to check if the array doesn’t have any phone. If the phone is empty, there will be a blank phone input and the name will be phone0 as default.



```
<!-- PHONE -->
<div class="pure-u-1-1 pure-u-md-1-2 form-group">
  <i class="fas fa-phone"></i><input type="text" class="field-text form-control" name="phone" placeholder="PHONE" [(ngModel)]="profile.phone" />
<div class="pure-u-1-1 pure-u-md-1-2 form-group" *ngIf="profile.phone.length != 0">


  <div *ngFor="let phone of profile.phone; let i = index">
      <i class="fas fa-phone"></i><input type="text" class="field-text form-control" name="phone{{i}}" placeholder="PHONE" [(ngModel)]="profile.phone[i]" />
  </div>
</div>


<div class="pure-u-1-1 pure-u-md-1-2 form-group" *ngIf="profile.phone.length == 0">
    <i class="fas fa-phone"></i><input type="text" class="field-text form-control" name="phone0" placeholder="PHONE" [(ngModel)]="profile.phone[0]" />
 </div>
</div>
```


PROFILE

Xinglong Jia	456678
	1256
Studentd	jia.xi@husky.neu.edu
Northeastern University	75 Peterborough St

Next will be the most important part. After extracting the data from solid, an interaction with solid profile is needed. First, the previous application use funcion getValueFromVcard based on the getValueFromNamespace

```
getValueFromVcard = (node: string, webId?: string): string | any => {
  return this.getValueFromNamespace(node, VCARD, webId);
};
```

```
private getValueFromNamespace(node: string, namespace: any, webId?: string): string | any {
  const store = this.store.any($rdf.sym(webId || this.session.webId), namespace(node));
  if (store) {
    return store.value;
  }
  return '';
}
```

But this approach can only handle single value, in order to handle multiple values like phone array, I create another approach to handle multiple values. Instead of using store.any(), I use store.each() to get all values as array.

```
    return this.getValueFromNamespace(node, VCARD, webId);
  };

+  getValuesFromVcard = (node: string, webId?: string): string[] | any => {
+    return this.getValuesFromNamespace(node, VCARD, webId);
+  };
+
```

```
private getValuesFromNamespace(node: string, namespace: any, webId?: string): string[] | any {
  const store = this.store.each($rdf.sym(webId || this.session.webId), namespace(node));
  if (store) {
    return store;
  }
  return '';
}
```

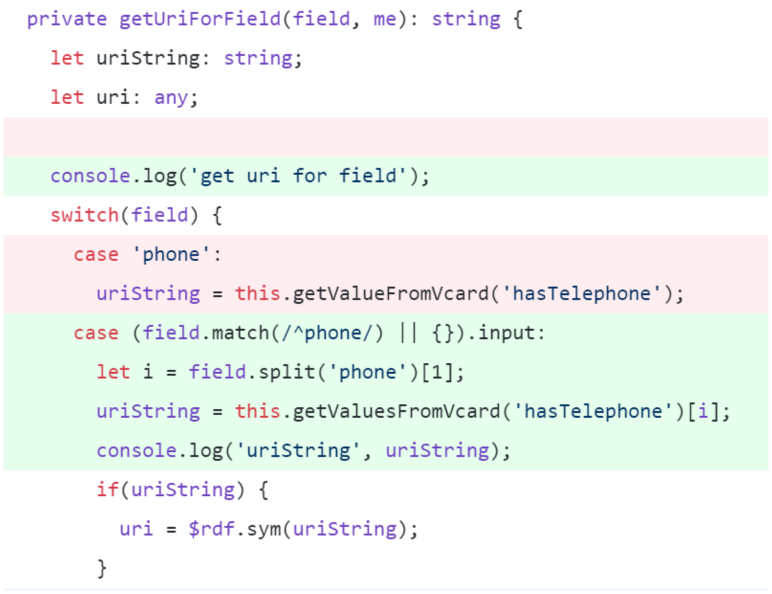
To use the multi values approach, take the phone numbers as example, extract all phone ids in the VCARD: hasTelephone predicate, and then use each phone id to get the value, the exact phone number, and put them in a string array.

```
//Function to get phone number. This returns only the first phone number, which is temporary. It also ignores the type.
getPhone = () => {
  const linkedUri = this.getValueFromVcard('hasTelephone');

  const linkedUri = this.getValuesFromVcard('hasTelephone');
  //this.store.each($rdf.sym(this.session.webId), VCARD('hasTelephone'));
  //this.getValueFromVcard('hasTelephone');
  console.log('LinkedUri', linkedUri);
  if(linkedUri) {
    return this.getValueFromVcard('value', linkedUri).split('tel:+')[1];
    const phones = [];
    linkedUri.forEach(uri => {
      phones.push(this.getValueFromVcard('value', uri).split('tel:')[1]);
    });
    console.log('phones', phones);
    return phones;
    //return this.getValueFromVcard('value', linkedUri[1]).split('tel:')[1];
  }
};
```

To handle profile update, create or delete, there is a transformDataForm function which collect all profile modifications and update the solid by creating RDF statement(subject, predicate, object, why) and using updateManager to ‘patch’ data.

```
transformDataForm = (form: NgForm, me: any, doc: any) => {
  const insertions = [];
  const deletions = [];
  const fields = Object.keys(form.value);
  const oldProfileData = JSON.parse(localStorage.getItem('oldProfileData')) || {};

  // We need to split out into three code paths here:
  // 1. There is an old value and a new value. This is the update path
  // 2. There is no old value and a new value. This is the insert path
  // 3. There is an old value and no new value. Ths is the delete path
  // These are separate codepaths because the system needs to know what to do in each case
  fields.map(field => {

    let predicate = VCARD(this.getFieldName(field));
    let subject = this.getUriForField(field, me);
    let why = doc;

    let fieldValue = this.getFieldValue(form, field);
    let oldFieldValue = this.getOldFieldValue(field, oldProfileData);
```

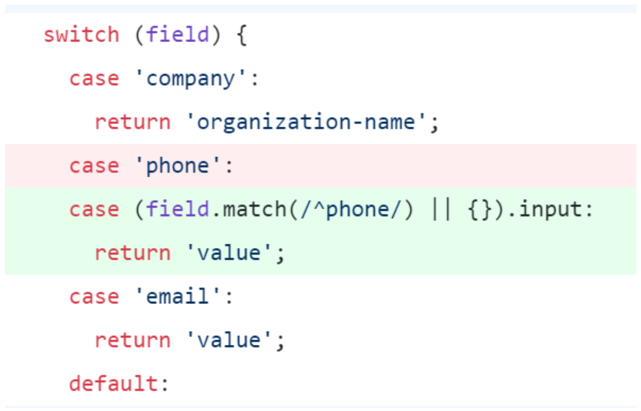
1. Subject



```
private getUriForField(field, me): string {
  let uriString: string;
  let uri: any;

  console.log('get uri for field');
  switch(field) {
    case 'phone':
      uriString = this.getValueFromVcard('hasTelephone');
    case (field.match(/^phone/) || {}).input:
      let i = field.split('phone')[1];
      uriString = this.getValuesFromVcard('hasTelephone')[i];
      console.log('uriString', uriString);
      if(uriString) {
        uri = $rdf.sym(uriString);
      }
```

Function getUriForField to extract subject from the from, here use regular expression to filter the phone0, phone1,… and use split to get the index of the phone element

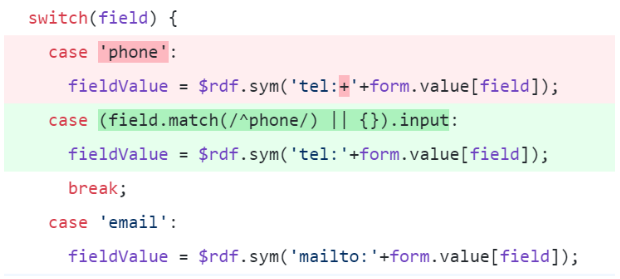
2. Predicate



```
switch (field) {
  case 'company':
    return 'organization-name';
  case 'phone':
  case (field.match(/^phone/) || {}).input:
    return 'value';
  case 'email':
    return 'value';
  default:
```

Function getFieldName to assign the link type needed for predicate

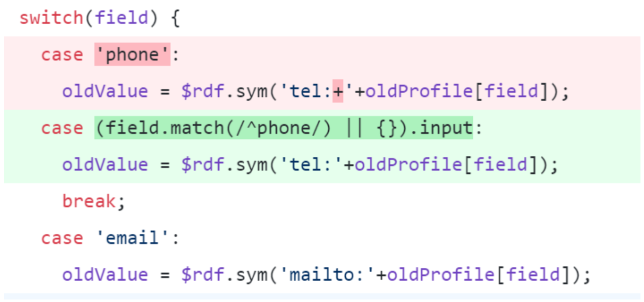
3. Object
   a. fieldValue



```
switch(field) {
  case 'phone':
    fieldValue = $rdf.sym('tel:+'+form.value[field]);
  case (field.match(/^phone/) || {}).input:
    fieldValue = $rdf.sym('tel:'+form.value[field]);
    break;
  case 'email':
    fieldValue = $rdf.sym('mailto:'+form.value[field]);
```

The new value will be added into the solid profile, the object of statements in the insertions.

   b. oldFieldValue



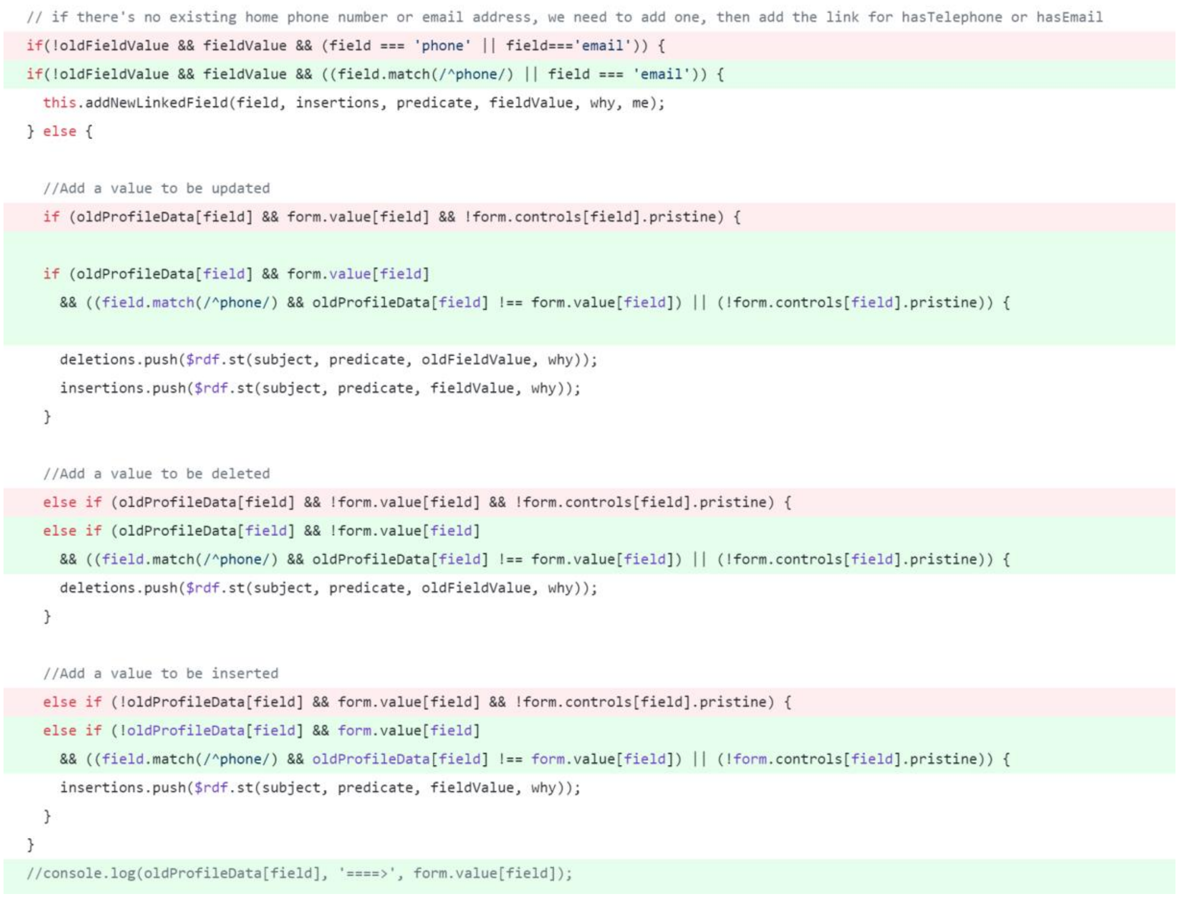
```
switch(field) {
  case 'phone':
    oldValue = $rdf.sym('tel:+'+oldProfile[field]);
  case (field.match(/^phone/) || {}).input:
    oldValue = $rdf.sym('tel:'+oldProfile[field]);
    break;
  case 'email':
    oldValue = $rdf.sym('mailto:'+oldProfile[field]);
```

The old value will be deleted from the solid profile, the object of statements in the deletions.

4. Why

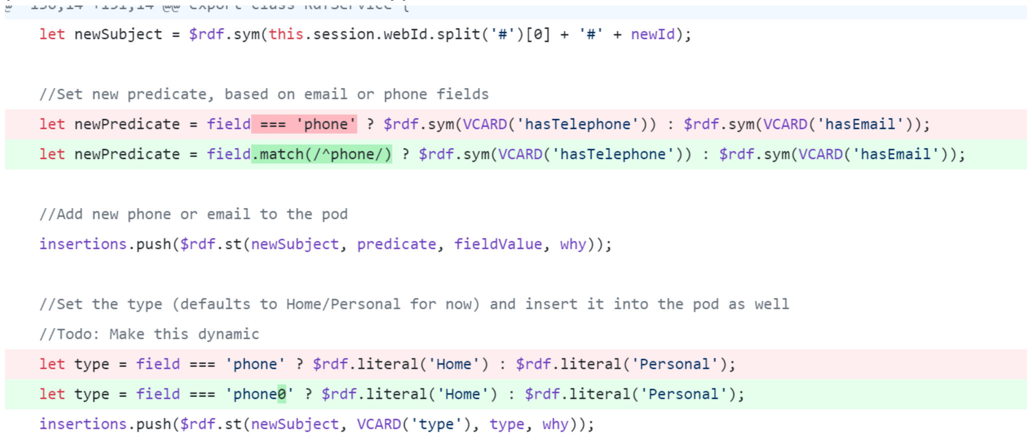
   Since all the update events happen in same place, no need to change.

Now, after getting all the elements in a RDF statement, need to divide separate code paths since the system needs to know that to do in different cases.



```
// if there's no existing home phone number or email address, we need to add one, then add the link for hasTelephone or hasEmail
if(!oldFieldValue && fieldValue && (field === 'phone' || field==='email')) {
if(!oldFieldValue && fieldValue && ((field.match(/^phone/) || field === 'email')) {
  this.addNewLinkedField(field, insertions, predicate, fieldValue, why, me);
} else {

  //Add a value to be updated
  if (oldProfileData[field] && form.value[field] && !form.controls[field].pristine) {

  if (oldProfileData[field] && form.value[field]
    && ((field.match(/^phone/) && oldProfileData[field] !== form.value[field]) || (!form.controls[field].pristine)) {


    deletions.push($rdf.st(subject, predicate, oldFieldValue, why));
    insertions.push($rdf.st(subject, predicate, fieldValue, why));
  }

  //Add a value to be deleted
  else if (oldProfileData[field] && !form.value[field] && !form.controls[field].pristine) {
  else if (oldProfileData[field] && !form.value[field]
    && ((field.match(/^phone/) && oldProfileData[field] !== form.value[field]) || (!form.controls[field].pristine)) {
    deletions.push($rdf.st(subject, predicate, oldFieldValue, why));
  }

  //Add a value to be inserted
  else if (!oldProfileData[field] && form.value[field] && !form.controls[field].pristine) {
  else if (!oldProfileData[field] && form.value[field]
    && ((field.match(/^phone/) && oldProfileData[field] !== form.value[field]) || (!form.controls[field].pristine)) {
    insertions.push($rdf.st(subject, predicate, fieldValue, why));
  }
}
//console.log(oldProfileData[field], '====>', form.value[field]);
```

1. If there is no existing phone number in the past, need to add a new phone id for thenew phone number, set the default type as home

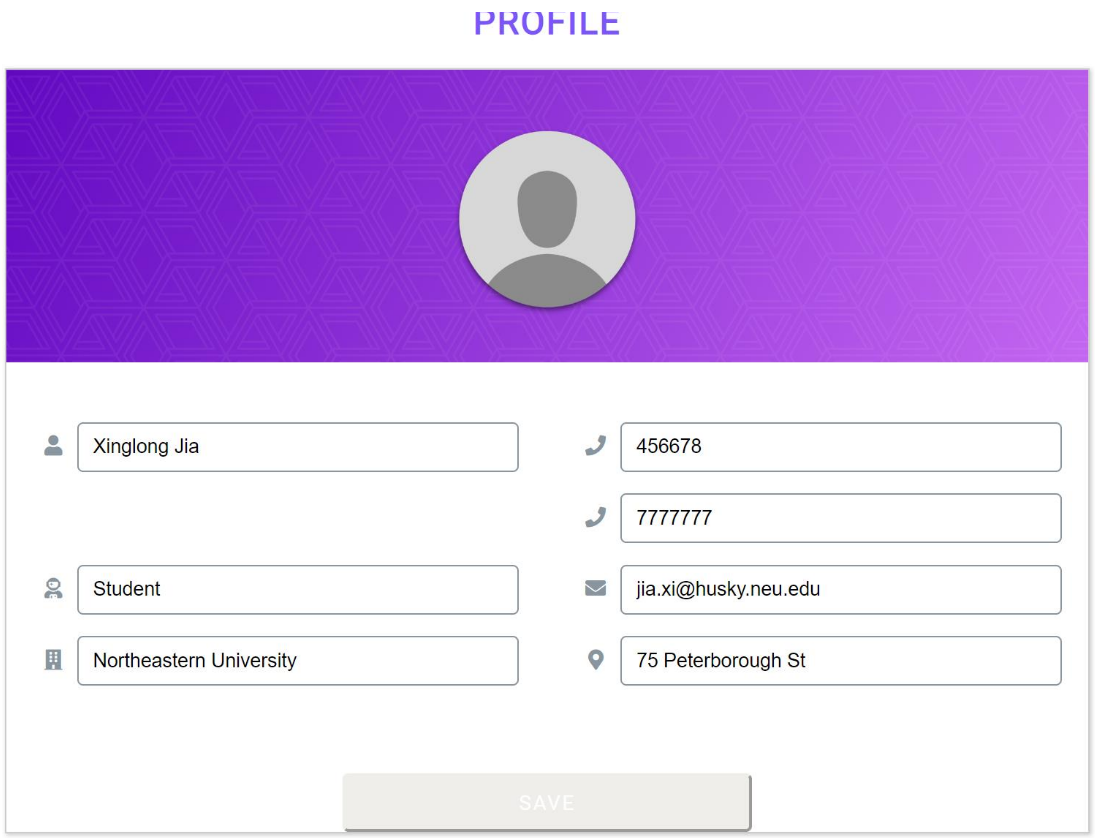


```
  let newSubject = $rdf.sym(this.session.webId.split('#')[0] + '#' + newId);

  //Set new predicate, based on email or phone fields
  let newPredicate = field === 'phone' ? $rdf.sym(VCARD('hasTelephone')) : $rdf.sym(VCARD('hasEmail'));
  let newPredicate = field.match(/^phone/) ? $rdf.sym(VCARD('hasTelephone')) : $rdf.sym(VCARD('hasEmail'));

  //Add new phone or email to the pod
  insertions.push($rdf.st(newSubject, predicate, fieldValue, why));

  //Set the type (defaults to Home/Personal for now) and insert it into the pod as well
  //Todo: Make this dynamic
  let type = field === 'phone' ? $rdf.literal('Home') : $rdf.literal('Personal');
  let type = field === 'phone0' ? $rdf.literal('Home') : $rdf.literal('Personal');
  insertions.push($rdf.st(newSubject, VCARD('type'), type, why));
```

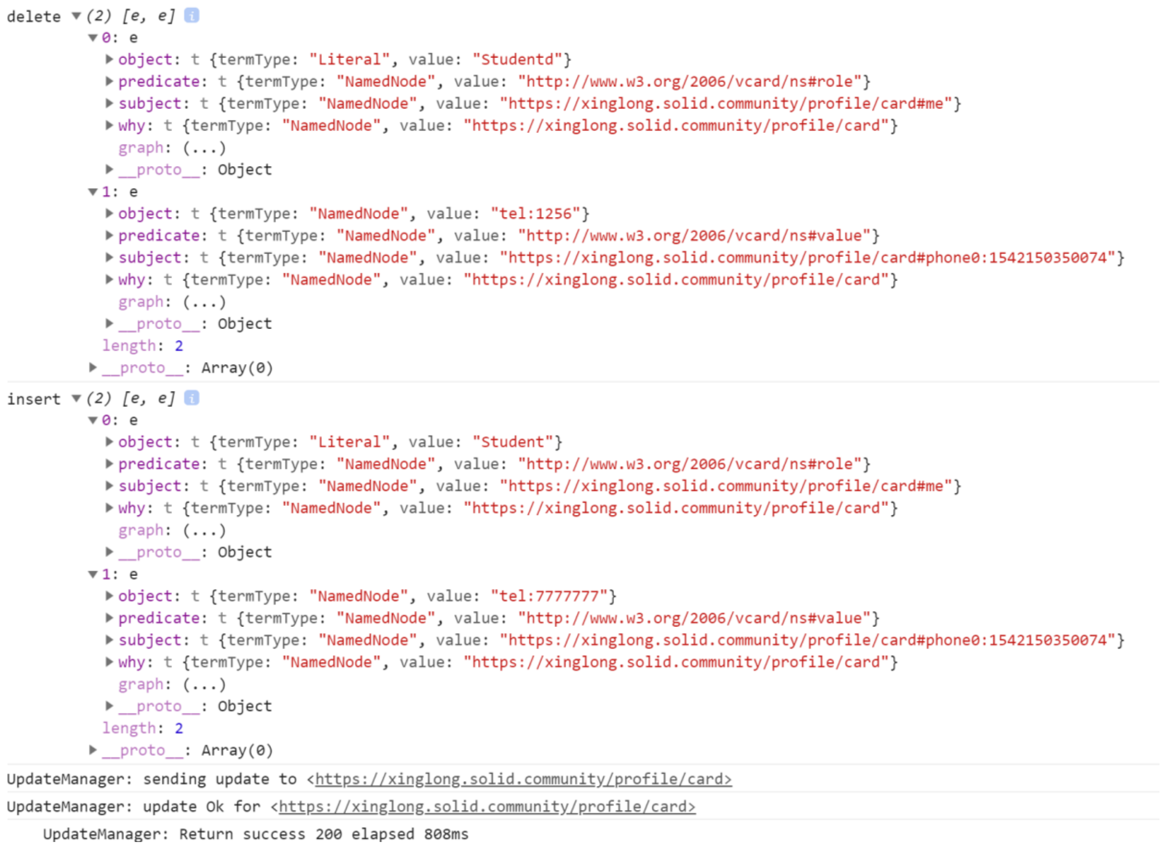
2. Add a value to be updated
    a. Check if old phone number equals to input phone number
    b. Add statements for deletions and insertions seperately, since the previous number need to be deleted and then add new number to the same location
3. Add a value to be deleted
    a. Just add statement to deletions
4. Add a value to be inserted

a. Just add statement to insertions

Final, make a test! Change pofile role and second phone as the same time.

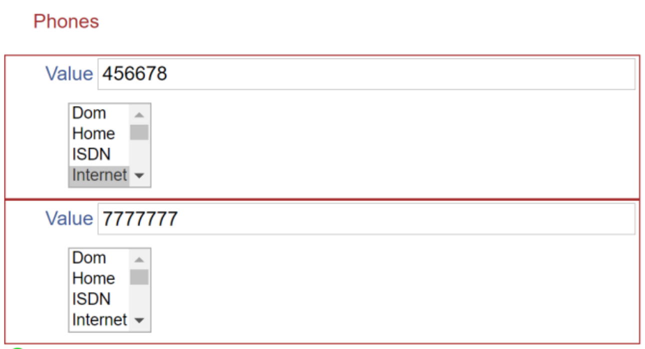


Xinglong Jia	456678
	7777777
Student	jia.xi@husky.neu.edu
Northeastern University	75 Peterborough St



```
delete ▼(2) [e, e] 
        ▼0: e
          ▶object: t {termType: "Literal", value: "Studentd"}
          ▶predicate: t {termType: "NamedNode", value: "http://www.w3.org/2006/vcard/ns#role"}
          ▶subject: t {termType: "NamedNode", value: "https://xinglong.solid.community/profile/card#me"}
          ▶why: t {termType: "NamedNode", value: "https://xinglong.solid.community/profile/card"}
           graph: (...)
          ▶__proto__: Object
        ▼1: e
          ▶object: t {termType: "NamedNode", value: "tel:1256"}
          ▶predicate: t {termType: "NamedNode", value: "http://www.w3.org/2006/vcard/ns#value"}
          ▶subject: t {termType: "NamedNode", value: "https://xinglong.solid.community/profile/card#phone0:1542150350074"}
          ▶why: t {termType: "NamedNode", value: "https://xinglong.solid.community/profile/card"}
           graph: (...)
          ▶__proto__: Object
         length: 2
        ▶__proto__: Array(0)
insert ▼(2) [e, e] 
        ▼0: e
          ▶object: t {termType: "Literal", value: "Student"}
          ▶predicate: t {termType: "NamedNode", value: "http://www.w3.org/2006/vcard/ns#role"}
          ▶subject: t {termType: "NamedNode", value: "https://xinglong.solid.community/profile/card#me"}
          ▶why: t {termType: "NamedNode", value: "https://xinglong.solid.community/profile/card"}
           graph: (...)
          ▶__proto__: Object
        ▼1: e
          ▶object: t {termType: "NamedNode", value: "tel:7777777"}
          ▶predicate: t {termType: "NamedNode", value: "http://www.w3.org/2006/vcard/ns#value"}
          ▶subject: t {termType: "NamedNode", value: "https://xinglong.solid.community/profile/card#phone0:1542150350074"}
          ▶why: t {termType: "NamedNode", value: "https://xinglong.solid.community/profile/card"}
           graph: (...)
          ▶__proto__: Object
         length: 2
        ▶__proto__: Array(0)
UpdateManager: sending update to <https://xinglong.solid.community/profile/card>
UpdateManager: update Ok for <https://xinglong.solid.community/profile/card>
    UpdateManager: Return success 200 elapsed 808ms
```

Log out the deletions and insertions, there are two statements for each array, because we update two values at the same time!



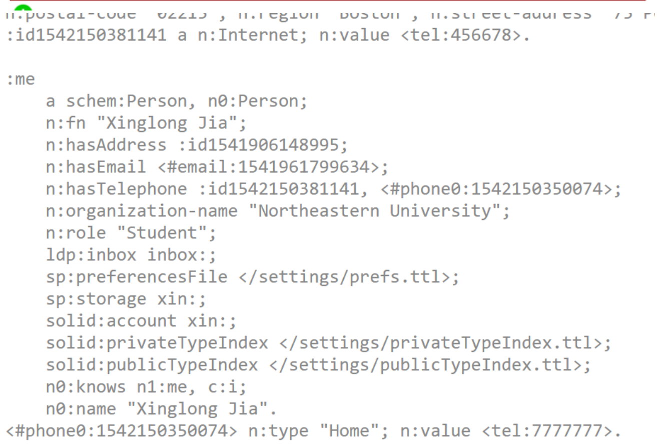
Phones

Value 456678

Dom
Home
ISDN
Internet

Value 7777777

Dom
Home
ISDN
Internet



```
n:postal-code "02215"; n:region "Boston"; n:street-address "75 F
:id1542150381141 a n:Internet; n:value <tel:456678>.

:me
    a schem:Person, n0:Person;
    n:fn "Xinglong Jia";
    n:hasAddress :id1541906148995;
    n:hasEmail <#email:1541961799634>;
    n:hasTelephone :id1542150381141, <#phone0:1542150350074>;
    n:organization-name "Northeastern University";
    n:role "Student";
    ldp:inbox inbox:;
    sp:preferencesFile </settings/prefs.ttl>;
    sp:storage xin:;
    solid:account xin:;
    solid:privateTypeIndex </settings/privateTypeIndex.ttl>;
    solid:publicTypeIndex </settings/publicTypeIndex.ttl>;
    n0:knows n1:me, c:i;
    n0:name "Xinglong Jia".
<#phone0:1542150350074> n:type "Home"; n:value <tel:7777777>.
```

<#phone0:1542150350074> are created from the empty phone number from the angular solid application, so the type is home, and it's name has phone0, these are all default parameters and we can change by ourselves.

While creating this approach to handle multi values, I understand solid better. This approach combines solid with practical web application. After this, I believe I totally understand solid structures and specifications, it gives me the ability to handle difficult solid data management problems and connect with other applications.

By the way, this approach can be widely used to handle other multi values, like email, address and etc. I really hope my effort will make solid better and better, and I am quite keen on working on this! If you have any question, please let me know. Thank you so much.