noltic

Libphonenumber

User Guide

The library for parsing and validating international phone numbers in Salesforce. Ported by Noltic

What is it?

This is a famous Google's library ported to Apex and adapted to the Salesforce best practices. Libphonenumber is a well-known Java, C++ and JavaScript library for parsing, formatting, and validating international phone numbers.

Frequently Asked Questions

Does this library make callouts (calls outside of Salesforce) to validate, parse or for any other activities?

No. Until version 1.7 we have used the method PageReference.getContent() to retrieve data from files, and as you can see from documentation it is treated as callout. We have since stopped using this method. This would be acceptable for clients who have requirements against external calls.

Does this library require coding skills?

Yes. This library is a port from Google's library and provides you with an interface to parse and validate numbers. At this time we do not provide a no-code solution bundled with the library to use it without coding. We will be assessing options to provide ready solutions in future. At this time we have documentation and there's extensive documentation of Google's library. Salesforce agencies should have no problem creating specific solutions for your needs.

Is this library integrated into default objects to parse/validate numbers?

No. Due to a number of different requirements for different clients we have not made any concrete implementations, just provide the library to allow doing so. We will be assessing options to provide ready solutions in future. At this time Salesforce agencies should have no problem creating specific solutions for your needs.

We would like to use the phone number validator for multiple objects. How can this be done?

Use an example that we have created for Contact object, but create similar triggers for each object that you'd like to use this for.

Do you offer ready solutions for Process Builder/Flow?

We don't offer ready solutions for flow due to limitations on error handling, but we are looking to provide this in the near future.

Can you use parse number without supplying defaultRegion to parse?

Yes. defaultRegion is only used if the number being parsed is not written in international format. The country_code for the number in this case would be stored as that of the default region supplied. If the number is guaranteed to start with a '+' followed by the country calling code, then null can be supplied.

Releases

The library will be maintained based on releases of the Google library. Any further updates from Google or bug fixes will be ported to Apex too. Libphonenumber for Salesforce will be also updated with useful specific features based on the requests from the clients.

Troubleshooting / reporting an issue

- Libphonenumber is a third-party port of Google's library. ReadMe by Google's library might answer some of your questions.

- In case there are issues in using the product you can get support from one of Noltic's Salesforce experts. Contact us via libphone@noltic.com.

Features & functionality

- PhoneNumberUtil
- PhoneNumberToCarrierMapper
- PhoneNumberToTimeZonesMapper
- Quick Examples

PhoneNumberUtil

Namespace: `noltic_libphone`
Class: `PhoneNumberUtil`

Methods:

- `format(number, numberFormat)` - formats a phone number in the specified format using default rules.
- `formatInOriginalFormat(number, regionCallingFrom)` - formats a phone number using the original phone number format that the number is parsed from.
- `formatOutOfCountryCallingNumber(number, regionCallingFrom)` - formats a phone number for out-of-country dialing purposes. If no regionCallingFrom is supplied, we format the number in its INTERNATIONAL format. If the country calling code is the same as that of the region where the number is from, then NATIONAL formatting will be applied.
- `getNumberType` - gets the type of the number based on the number itself; able to distinguish Fixed-line, Mobile, Toll-free, Premium Rate, Shared Cost, VoIP, Personal Numbers, UAN, Pager, and Voicemail (whenever feasible).
- `getRegionCodeForNumber(number)` - returns the region where a phone number is from. This could be used for geocoding at the region level. Only guarantees correct results for valid, full numbers (not short-codes, or invalid numbers).
- `isNumberMatch` - gets a confidence level on whether two numbers could be the same.
- `getExampleNumber` and `getExampleNumberForType` - provide valid example numbers for all countries/regions, with the option of specifying which type of example phone number is needed.
- `isPossibleNumber` - quickly guesses whether a number is a possible phone number by using only the length information, much faster than a full validation.
- `isValidNumber` - full validation of a phone number for a region using length and prefix information.Tests whether a phone number matches a valid pattern. Note this doesn't verify the number is actually in use, which is impossible to tell by just looking at a number itself. It only verifies whether the parsed, canonicalized number is valid: not

whether a particular series of digits entered by the user is diallable from the region provided when parsing.

- `isValidNumberForRegion(number, regionCode)` - tests whether a phone number is valid for a certain region. Note this doesn't verify the number is actually in use, which is impossible to tell by just looking at a number itself. If the country calling code is not the same as the country calling code for the region, this immediately exits with false. After this, the specific number pattern rules for the region are examined. This is useful for determining for example whether a particular number is valid for Canada, rather than just a valid NANPA number.
- `findNumbers` - finds numbers in the text.
- `parseAndKeepRawInput(numberToParse, defaultRegion)` - parses a string and returns it in proto buffer format.
- `parse(numberToParse, defaultRegion)` - the most important method of the library. Parses a string and returns it as a phone number in proto buffer format. The method is quite lenient and looks for a number in the input text (raw input) and does not check whether the string is definitely only a phone number. To do this, it ignores punctuation and white-space, as well as any text before the number (e.g. a leading "Tel: ") and trims the non-number bits. It will accept a number in any format (E164, national, international etc), assuming it can be interpreted with the defaultRegion supplied. It also attempts to convert any alpha characters into digits if it thinks this is a vanity number of the type "1800 MICROSOFT"

PhoneNumberToCarrierMapper

Namespace: `noltic_libphone`

Class: `PhoneNumberToCarrierMapper`

Methods (provide timezone information related to a phone number)**:**

- `getNameForValidNumber` - Returns a carrier name for the given phone number, in the language provided. If no mapping is found an empty string is returned.
- `getNameForNumber` - Gets the name of the carrier for the given phone number, in the language provided, but explicitly checks the validity of the number passed in.
- `getSafeDisplayName` - Gets the name of the carrier for the given phone number only when it is 'safe' to display to users. A carrier name is considered safe if the number is valid and for a region that doesn't support mobile number portability.

PhoneNumberToTimeZonesMapper

Namespace: `noltic_libphone`

Class: `PhoneNumberToTimeZonesMapper`

Methods (provide timezone information related to a phone number)**:**

- `getTimeZonesForGeographicalNumber` - returns a list of time zones to which a phone number belongs.
- `getUnknownTimeZone` - returns a String with the ICU unknown time zone.

Quick Examples

1. Parse Number

Let's say you have a string representing a phone number from Switzerland.

This is how you parse/normalize it into a PhoneNumber object:

```
/**
 * swissNumberProto = PhoneNumber:[countryCode=41,
nationalNumber=446681800]
 */
String swissNumberStr = '044 668 18 00';
noltic_libphone.PhoneNumberUtil phoneUtil =
noltic_libphone.PhoneNumberUtil.getInstance();
try {
    noltic_libphone.PhoneNumber swissNumberProto =
phoneUtil.parse(swissNumberStr, 'CH');
} catch (noltic_libphone.NumberParseException
 e) {
    System.debug('NumberParseException was thrown: ' + e);
}
```

It’s possible to parse the phone number without specifying the region, as long as it’s provided in the international format starting with a ‘+’:

```
String nzNumberStr = '+41 044 668 18 00';
noltic_libphone.PhoneNumberUtil phoneUtil =
noltic_libphone.PhoneNumberUtil.getInstance();

try {
    noltic_libphone.PhoneNumber swissNumberProto =
phoneUtil.parse(nzNumberStr, RegionCode.ZZ);
} catch (noltic_libphone.NumberParseException
 e) {
    System.debug('NumberParseException was thrown: ' + e);
}
```

2. Validating Number

```
Boolean isValid = phoneUtil.isValidNumber(swissNumberProto); // returns
true
```

```
Boolean isValid = phoneUtil.isValidNumberForRegion(swissNumberStr,
'CH'); // returns true since this number is valid for CH region
```

3. Formatting Number

There are a few formats supported by the formatting method, as illustrated below:

```
// Produces "+41 44 668 18 00"
            System.debug(phoneUtil.format(swissNumberProto,
noltic_libphone.PhoneNumberUtil.PhoneNumberFormat.INTERNATIONAL));
            // Produces "044 668 18 00"
            System.debug(phoneUtil.format(swissNumberProto,
noltic_libphone.PhoneNumberUtil.PhoneNumberFormat.NATIONAL));
            // Produces "+41446681800"
            System.debug(phoneUtil.format(swissNumberProto,
noltic_libphone.PhoneNumberUtil.PhoneNumberFormat.E164));
```

You could also choose to format the number in the way it is dialed from another country:

```
// Produces "011 41 44 668 1800", the number when it is dialed in the
United States.

System.debug(phoneUtil.formatOutOfCountryCallingNumber(swissNumberProto,
'US'));
```

4. Mapping Phone Numbers to original carriers

Caveat: We do not provide data about the current carrier of a phone number, only the original carrier who is assigned the corresponding range.

```
noltic_libphone.PhoneNumber swissMobileNumber =
           new
noltic_libphone.PhoneNumber().setCountryCode(41).setNationalNumber(79876
5432L);
       noltic_libphone.PhoneNumberToCarrierMapper carrierMapper =
noltic_libphone.PhoneNumberToCarrierMapper.getInstance();
       // Outputs "Swisscom"
       System.debug(carrierMapper.getNameForNumber(swissMobileNumber,
noltic_libphone.Locale.ENGLISH));
```

5. Mapping Phone Numbers to time zones

```
noltic_libphone.PhoneNumber auNumber = new noltic_libphone
       .PhoneNumber().setCountryCode(61).setNationalNumber(236618300L);

noltic_libphone.PhoneNumberToTimeZonesMapper timezoneMapper =
       noltic_libphone.PhoneNumberToTimeZonesMapper.getInstance();

// Outputs "Australia/Sydney"
System.debug(
        timezoneMapper.getTimeZonesForGeographicalNumber(auNumber)
);
```

6. Get country code for parsed number

```
String nzNumberStr = '+41 044 668 18 00';
noltic_libphone.PhoneNumberUtil phoneUtil =
noltic_libphone.PhoneNumberUtil.getInstance();

try {
    noltic_libphone.PhoneNumber swissNumberProto =
phoneUtil.parse(nzNumberStr, RegionCode.ZZ);

    // Outputs "41"
    System.debug(
        swissNumberProto.getCountryCode()
    );

    // Outputs "CH"
    System.debug(
        phoneUtil.getRegionCodeForCountryCode(
                swissNumberProto.getCountryCode()
        )
    );
} catch (noltic_libphone.NumberParseException
 e) {
    System.debug('NumberParseException was thrown: ' + e);
}
```

7. Validate number for Contact object from trigger

```
trigger ContactTrigger on Contact (before insert, before update) {


    noltic_libphone.PhoneNumberUtil phoneUtil =
noltic_libphone.PhoneNumberUtil.getInstance();

    for (Contact contact : Trigger.new) {
        try {
            noltic_libphone.PhoneNumber numberProto =
phoneUtil.parse(contact.Phone, contact.MailingCountryCode);

            if (!phoneUtil.isValid(numberProto)) {
                contact.Phone.addError('The number you provided is
invalid');
            } else {
                contact.Phone = phoneUtil.format(numberProto,
noltic_libphone.PhoneNumberUtil.PhoneNumberFormat.E164);
            }
        } catch (noltic_libphone.NumberParseException e) {
            contact.Phone.addError(e);
        }
    }

}
```