

Os algoritmos utilizados pelo aplicativo de carga do sistema SIB.XML são descritos a seguir com a finalidade de informar e permitir compatibilizar os procedimentos de atualização cadastral de beneficiários entre as operadoras e a ANS.

Os algoritmos relacionados neste documento são os seguintes:

1. CPF
2. CEI
3. CNPJ
4. CNS
5. DN
6. NOME
7. PIS/PASEP
8. CAEPF

1 CPF:

```
public boolean isCpfValido(String cpf) {

    String strCpf = cpf.replaceAll("[^0-9]*", "");

    if (strCpf.length() != 11 || strCpf.equals("00000000000")
        || strCpf.equals("11111111111")
        || strCpf.equals("22222222222")
        || strCpf.equals("33333333333")
        || strCpf.equals("44444444444")
        || strCpf.equals("55555555555")
        || strCpf.equals("66666666666")
        || strCpf.equals("77777777777")
        || strCpf.equals("88888888888")
        || strCpf.equals("99999999999"))

    { return false;

    } else {
        int d1, d2;

        int digito1, digito2, resto;

        int digitoCPF;

        String nDigResult;

        d1 = d2 = 0;

        digito1 = digito2 = resto = 0;

        for (int nCount = 1; nCount < strCpf.length() - 1; nCount++) {
```

```
        digitoCPF = Integer.valueOf(strCpf.substring(

nCount - 1, nCount)).intValue();

        d1 = d1 + (11 - nCount) * digitoCPF;

        d2 = d2 + (12 - nCount) * digitoCPF;

    }
    resto = (d1 % 11);

    if (resto < 2) {

        digito1 = 0;

    } else {

        digito1 = 11 - resto;

    }

    d2 += 2 * digito1;

    resto = (d2 % 11);

    if (resto < 2) {

        digito2 = 0;

    } else {

        digito2 = 11 - resto;

    }

    String nDigVerific =
        strCpf.substring( strCpf.length() - 2,
            strCpf.length());

    nDigResult = String.valueOf(digito1) + String.valueOf(digito2);

    return nDigVerific.equals(nDigResult);

}

}
```

2 CEI:

```
public boolean novaValidaCEI(Object value) {
    final int[] peso = {7, 4, 1, 8, 5, 2, 1, 6, 3, 7, 4};
```

```
List<String> invalidos = Arrays.asList("000000000000", "111111111111",
    "222222222222", "333333333333", "444444444444", "555555555555",
    "666666666666", "777777777777", "888888888888", "999999999999");

String cei = (String) value;

if (cei.length() != 12 || !NumberUtils.isNumber(cei) ||
    invalidos.contains(cei) ||
    cei.substring(0, 10).equals("0000000000"))
    { return false;
}

char[] algarismos = cei.toCharArray();

Integer soma = 0;

for (int i = 0; i < 11; i++) {
    soma += peso[i] * toInteger(algarismos[i]);
}

int unidadeSoma = getUnidade(soma);
int dezenaSoma = getDezena(soma);

Integer soma2 = unidadeSoma + dezenaSoma;

int unidadeSoma2 = getUnidade(soma2);

Integer subtracao = 10 - unidadeSoma2;

int unidadeSubtracao = getUnidade(subtracao);

if (unidadeSubtracao == toInteger(algarismos[11]))
    { return true;
}

return false;
}
```

3 CNPJ:

```
protected boolean
isValidCnpj(String cnpj) {

    if (cnpj != null) {

        cnpj =

        removeCaracteresFormatacao(

        cnpj);

        if
```

```
(isCnpjFormacaoValidaComDV(  
cnpj)) {  
    String dvInformado =  
cnpj.substring(TAMANHO_CNPJ  
_SEM_DV);  
    String dvCalculado =  
calculaDV(cnpj.substring(0,  
TAMANHO_CNPJ_SEM_DV));  
    return  
dvCalculado.equals(dvInform  
ado);  
}  
}  
return false;  
}
```

```
protected String  
calculaDV(String baseCnpj)  
{  
    baseCnpj =  
removeCaracteresFormatacao(  
baseCnpj);  
  
    if  
(isCnpjFormacaoValidaSemDV(  
baseCnpj)) {  
        String dv1 =  
String.format("%d",  
calculaDigito(baseCnpj));  
        String dv2 =
```

```
String.format("%d",  
calculaDigito(baseCnpj.conc  
at(dv1)));  
    return dv1.concat(dv2);  
}
```

```
    throw new  
IllegalArgumentException(St  
ring.format("Cnpj %s não é  
válido para o cálculo do  
DV", baseCnpj));  
}
```

```
protected int  
calculaDigito(String cnpj)  
{  
    int soma = 0;  
    for (int indice =  
cnpj.length() - 1;  
indice >= 0; indice--) {  
        int valorCaracter =  
(int) cnpj.charAt(indice) -  
VALOR_BASE;  
        soma += valorCaracter *  
PESOS_DV[PESOS_DV.length -  
cnpj.length() + indice];  
    }  
    return soma % 11 < 2 ?  
0 : 11 - (soma % 11);  
}
```

```
        protected String  
        removeCaracteresFormatacao(  
        String cnpj) {  
            return  
        cnpj.trim().replaceAll(Regex  
        X_CARACTERES_FORMATACAO,  
        "");  
    }  
  
    protected boolean  
    isCnpjFormacaoValidaSemDV(  
        String cnpj) {  
        return  
        cnpj.matches(Regex_FORMACAO  
        _BASE_CNPJ)  
        && !cnpj.matches(Regex_V  
        ALOR_ZERADO);  
    }  
  
    protected boolean  
    isCnpjFormacaoValidaComDV(  
        String cnpj) {  
        return  
        cnpj.matches(Regex_FORMACAO  
        _BASE_CNPJ.concat(Regex_FOR  
        MACAO_DV))  
        && !cnpj.matches(Regex_V  
        ALOR_ZERADO);  
    }  
}
```

4 CNS:

```
public boolean isDvCnsValido(String numCns)
{ String cns = numCns.replaceAll("[^0-9]*",
  "");

  if (cns.equals("0000000000000000"))
    { return false;
    }

  return validaCns(cns) || validaCnsProv(cns);
}

public boolean validaCnsProv(String cns)
{ if (cns.trim().length() != 15) {
    return (false);
  }

  float resto, soma;

  soma = ((Integer.valueOf(cns.substring(0, 1)).intValue()) * 15)
    + ((Integer.valueOf(cns.substring(1, 2)).intValue()) * 14)
    + ((Integer.valueOf(cns.substring(2, 3)).intValue()) * 13)
    + ((Integer.valueOf(cns.substring(3, 4)).intValue()) * 12)
    + ((Integer.valueOf(cns.substring(4, 5)).intValue()) * 11)
    + ((Integer.valueOf(cns.substring(5, 6)).intValue()) * 10)
    + ((Integer.valueOf(cns.substring(6, 7)).intValue()) * 9)
```

```
+ ((Integer.valueOf(cns.substring(7, 8)).intValue()) * 8)
+ ((Integer.valueOf(cns.substring(8, 9)).intValue()) * 7)
+ ((Integer.valueOf(cns.substring(9, 10)).intValue()) * 6)
+ ((Integer.valueOf(cns.substring(10, 11)).intValue()) * 5)
+ ((Integer.valueOf(cns.substring(11, 12)).intValue()) * 4)
+ ((Integer.valueOf(cns.substring(12, 13)).intValue()) * 3)
+ ((Integer.valueOf(cns.substring(13, 14)).intValue()) * 2)
+ ((Integer.valueOf(cns.substring(14, 15)).intValue()) * 1);

resto = soma % 11;

if (resto != 0)
    { return (false);
} else {
    return (true);
}
}

public boolean validaCns(String cns)
{ if (cns.trim().length() != 15) {
    return (false);
}

float soma;

float resto, dv;

String pis = "";

String resultado = "";

pis = cns.substring(0, 11);

soma = ((Integer.valueOf(pis.substring(0, 1)).intValue()) * 15)
+ ((Integer.valueOf(pis.substring(1, 2)).intValue()) * 14)
+ ((Integer.valueOf(pis.substring(2, 3)).intValue()) * 13)
+ ((Integer.valueOf(pis.substring(3, 4)).intValue()) * 12)
+ ((Integer.valueOf(pis.substring(4, 5)).intValue()) * 11)
+ ((Integer.valueOf(pis.substring(5, 6)).intValue()) * 10)
+ ((Integer.valueOf(pis.substring(6, 7)).intValue()) * 9)
+ ((Integer.valueOf(pis.substring(7, 8)).intValue()) * 8)
+ ((Integer.valueOf(pis.substring(8, 9)).intValue()) * 7)
+ ((Integer.valueOf(pis.substring(9, 10)).intValue()) * 6)
+ ((Integer.valueOf(pis.substring(10, 11)).intValue()) * 5);

resto = soma % 11;

dv = 11 - resto;

if (dv == 11)
    { dv = 0;
}

if (dv == 10) {
    soma = ((Integer.valueOf(pis.substring(0, 1)).intValue()) * 15) +
+ ((Integer.valueOf(pis.substring(1, 2)).intValue()) * 14) +
+ ((Integer.valueOf(pis.substring(2, 3)).intValue()) * 13) +
```

```
((Integer.valueOf(pis.substring(3, 4)).intValue()) * 12) +
((Integer.valueOf(pis.substring(4, 5)).intValue()) * 11) +
((Integer.valueOf(pis.substring(5, 6)).intValue()) * 10) +
((Integer.valueOf(pis.substring(6, 7)).intValue()) * 9) +
((Integer.valueOf(pis.substring(7, 8)).intValue()) * 8) +
((Integer.valueOf(pis.substring(8, 9)).intValue()) * 7) +
((Integer.valueOf(pis.substring(9, 10)).intValue()) * 6) +
((Integer.valueOf(pis.substring(10, 11)).intValue()) * 5)
+ 2;

resto = soma % 11;

dv = 11 - resto;

resultado = pis + "001" + String.valueOf((int) dv);

} else {
    resultado = pis + "000" + String.valueOf((int) dv);
}

if (!cns.equals(resultado))
    { return (false);
} else {
    return (true);
}

}
```

5 DN:

```
public boolean isDNValida(String
dn){ if(dn.trim().length() != 11
    || dn.trim().equals("000000000000")
    || dn.trim().equals("111111111111")
    || dn.trim().equals("222222222222")
    || dn.trim().equals("333333333333")
    || dn.trim().equals("444444444444")
    || dn.trim().equals("555555555555")
    || dn.trim().equals("666666666666")
    || dn.trim().equals("777777777777")
    || dn.trim().equals("888888888888")
    ||
    dn.trim().equals("999999999999")){ return
false;
}

return true;
}
```

6 NOME:

```
private final String[] TOKENS_INICIAIS_VALIDOS = {"D", "I", "O", "U", "Y"};
private final String[] TOKENS_FINAIS_VALIDOS = {"I", "O", "U", "Y"};
private final String[] TOKENS_INTERMEDIARIOS_VALIDOS = {"E", "Y"};

private final String[] PATTERNS_CARACTERES_VALIDOS =
    {"[a-zA-ZãÄåÀàÃäÄéÉèÈēÊêËëÍíÎîÓóÔôÕõÖüÛüÑñÇç]*",
     "[a-zA-ZãÄåÀàÃäÄéÉèÈēÊêËëÍíÎîÓóÔôÕõÖüÛüÑñÇç]+'{1}"
     + "[a-zA-ZãÄåÀàÃäÄéÉèÈēÊêËëÍíÎîÓóÔôÕõÖüÛüÑñÇç]+'"};

private final String PATTERN_REPETICAO_3_PRIMEIROS_CARACTERES =
    "(A{3}|B{3}|C{3}|D{3}|E{3}|F{3}|G{3}|H{3}|I{3}|J{3}|K{3}|L{3}|M{3}"
    + "|N{3}|O{3}|P{3}|Q{3}|R{3}|S{3}|T{3}|U{3}|V{3}|X{3}|W{3}|Y{3}|"
    + "Z{3})([\\x00-\\x7F]*)";

public boolean isNomeValido(String nome)

    { nome = nome.replaceAll("[\\s]{2,}", "

    "); String[] tokens = nome.split(" ");

    if (tokens.length <= 1)
        { return false;
        }

    for (int i = 0; i < tokens.length; i++)
        { tokens[i] = tokens[i].toUpperCase().trim();
        }

    for (int i = 0; i < tokens.length; i++)

        { if (tokens[i].length() == 1) {

            if (((i == 0) &&
                Arrays.binarySearch( TOKENS_INICIAIS_VALIDOS,
                tokens[i]) < 0)) {
                return false;

            } else if (((i == tokens.length - 1) &&
                Arrays.binarySearch( TOKENS_FINAIS_VALIDOS, tokens[i]) <
                0)) {
                return false;

            } else if ((i == 1) || ((i > 1 && i < tokens.length - 1) &&
                !((Arrays.binarySearch( TOKENS_INTERMEDIARIOS_VALIDO
                S, tokens[i]) >= 0)))) {
                return false;

            }

        } else if (!isTokenValido(tokens[i]))
            { return false;
            }

        }

    }
```

```
        return true;
    }

    public boolean isTokenValido(String token) {
        if (Pattern.compile(PATTERNS_CARACTERES_VALIDOS[0])
            .matcher(token).matches() ||
            Pattern.compile(PATTERNS_CARACTERES_VALIDOS[1]).matcher(token)
            .matches()) {

            if (Pattern.compile(PATTERN_REPETICAO_3_PRIMEIROS_CARACTERES)
                .matcher(token).matches()) {

                return false;
            }

        } else {
            return false;
        }

        return true;
    }
}
```

7 PISPASEP:

```
public boolean isDvPisPasepValido(String pisOrPasep)

{
    final int DIGIT_COUNT = 11;

    if (pisOrPasep == null)
        { return false;
        }

    String n = pisOrPasep.replaceAll("[^0-9]*", "");

    if (n.length() != DIGIT_COUNT) {
        return false;
    }

    int i;
    int digit;
    int coeficient =2;
    int sum = 0;

    int foundDv;

    int dv = Integer.parseInt(String.valueOf(n.charAt(n.length() - 1)));

    for (i = n.length() - 2; i >= 0; i--) {

        digit = Integer.parseInt(String.valueOf(n.charAt(i)));
```

```
        sum += digit * coeficiente;

        coeficiente++;

        if (coeficiente > 9)
            { coeficiente = 2;
            }

    }

    foundDv = 11 - sum % 11;

    if (foundDv >= 10) {
        foundDv = 0;
    }

    return dv == foundDv;

}
```

8 CAEPF:

```
protected boolean validaCAEPF(String strcaepf)
{

    String caepf = strcaepf.replaceAll("[^0-9]*", "");

    if ((caepf == null) || (caepf.length() != 14))
    {

        return false;

    }

    Integer digito1 = this.calcularDigito(caepf.substring(0, 12));

    Integer digito2 = this.calcularDigito(caepf.substring(0, 12) + digito1);

    Integer digitocaepf = digito1 * 10 + digito2 + 12;

    if (digitocaepf > 99)
        digitocaepf = digitocaepf - 100;

    String strdigito = digitocaepf.toString();

    if(strdigito.length() == 1)
        strdigito = "0" + strdigito;

    return caepf.equals(caepf.substring(0, 12) + strdigito);

}
```

```
protected int calcularDigito(String str)
```

```
{  
  
    final int[] peso = { 6, 5, 4, 3, 2, 9, 8, 7, 6, 5, 4, 3, 2 };  
  
    int soma = 0;  
  
    for (int indice = str.length() - 1, digito; indice >= 0; indice--)  
    {  
  
        digito = Integer.parseInt(str.substring(indice, indice + 1));  
  
        soma += digito * peso[peso.length - str.length() + indice];  
  
    }  
  
    soma = 11 - soma % 11;  
  
    return soma > 9 ? 0 : soma;  
}
```