

# Programme Identifiant National de Santé

## Dossier de conception INS-C – Exemples de code

### « ASIP Santé / PRAS »

Identification du document	
Référence	<b>ASIP - Programme INS - Dossier De Conception INS-C - Algorithme De Calcul</b>
Date de création	<b>26/06/09</b>
Date de dernière mise à jour	<b>30/10/09</b>
Rédaction (R)	ASIP Santé / PRAS
Version	<b>V 1.0.0</b>
Nombre de pages	<b>27</b>

Référence	Document	Version	Date
1. [SPEC INSC CONCEPT]	« Dossier de conception INSC »	1.0.0	26/11/09
2. [SPEC INS-C]	« Dossier de conception INSC – Algorithme de calcul »	1.0.0	26/10/09
3. [INSC-DCSSI]	« Demande GIP-DMP 005 – Solution proposée pour la création de l'Identifiant National de Santé calculé localement dit « INS-C » »	-	11/05/09

Historique du document			
Version	Date	Auteur	Commentaires
1.0.0	26/09/09	GIPDMP/DRAS	Initialisation du document

# Liminaire

Ce document est l'annexe technique du « dossier de conception INS-C – algorithme de calcul ».

Il a pour objet de démontrer la faisabilité de l'implémentation de l'algorithme de calcul de l'INS-C dans les langages de développement Java, C (C++) et C#.

Les étapes clés spécifiques à l'algorithme de calcul des INS-C sont illustrées par des exemples de code libres de droit. Un éditeur peut s'en inspirer librement pour sa propre implémentation de l'algorithme.

Ces exemples de codes sont publiés en « l'état », sans aucune garantie de quelque nature que ce soit. L'ASIP Santé n'assurera pas de maintenance de ces exemples ; en particulier les évolutions des langages de programmation utilisés devront être prises en compte par les éditeurs.

En aucun cas l'ASIP ne saurait être tenue responsable de dommages résultant de l'utilisation directe de ces exemples de code.

# Sommaire

1	Introduction .....	4
1.1	Contexte.....	4
1.2	Objet .....	4
2	Rappel de l'algorithme.....	5
3	Exemples de code.....	6
3.1	Etape 3 : Constituer la graine .....	6
3.1.1	Normaliser le prénom .....	6
3.1.2	Former la graine.....	9
3.2	Etape 4 : Calculer l'empreinte .....	12
3.3	Etape 5 : Convertir l'empreinte en numérique.....	15
3.4	Etape 6 : Calcul de l'INS-C.....	19
3.5	Algorithme de calcul de l'INS-C.....	21
4	Annexe : références externes.....	27

# 1 Introduction

## 1.1 Contexte

Pour répondre à leurs besoins de conservation, voire d'échange et de partage de données de santé, plusieurs régions ont développé des solutions d'attribution d'identifiants de patients. Les principes retenus permettent une génération distribuée des identifiants de patient, au niveau des LPS.

Ces principes étant différents selon les projets, l'échange et le partage de données de santé entre régions nécessitent la mise en place d'accords bilatéraux et de mécanismes de rapprochement d'identité des patients souvent complexes et débordant du périmètre de responsabilité des acteurs impliqués.

Le déploiement de l'identifiant national de santé généré aléatoirement et attribué par un système central permettra de sécuriser le contenu des échanges (la bonne donnée pour le bon patient), en garantissant les qualités suivantes à cet identifiant : identifiant sans collision, sans doublon, non signifiant et non prévisible, Cet INS géré par un système central est l'INS-A.

Dans l'attente de l'INS-A, un algorithme partagé de génération d'identifiant permet d'apporter une amélioration immédiate à la situation actuelle. Les identifiants ainsi calculés sont appelés INS-C (INS Calculé).

L'INS-C constitue le premier palier de mise en œuvre de l'identifiant national de santé.

## 1.2 Objet

Le choix d'une génération répartie des INS-C impose un processus basé sur un algorithme unique partagé et connu de tous les acteurs, permettant une intégration facile dans les outils des professionnels de santé.

Par définition, ce processus ne peut être un processus aléatoire car il doit être reproductible et doit se fonder sur des critères discriminants disponibles pour les systèmes devant générer l'INS-C.

Ce document illustre l'algorithme de génération partagé avec des exemples de codes réalisés dans trois langages de programmation Java, C et C#.

Les exemples de code proposés dans ce document ont été testés sur des postes équipés des trois systèmes d'exploitation suivant : Windows XP Pro, Linux Fedora 10 et Mac OS X.

## 2 Rappel de l'algorithme

### Algorithme de calcul de l'INS-C

1. Récupérer les données nécessaires au calcul (lecture de la carte Vitale) : NIR, prénoms et date de naissance de la personne concernée.
2. Normaliser les données selon les règles énoncées dans la suite du document dans l'ordre suivant :
  - a. normalisation des caractères
  - b. puis normalisation des champs.
3. Concaténer sans séparateur les trois champs dans l'ordre suivant : Prénom, date de naissance puis NIR.  
L'ensemble concaténé constitue la « graine ».  
Graine = Prénom (10 caractères) + Date de naissance (6 caractères) + NIR (13 caractères).
4. Calculer le haché de la chaîne de caractère 'Graine' au moyen de la fonction SHA256 (cf. le référentiel cryptographique de l'ANSSI)  
Le résultat obtenu est une empreinte de 256 bits.
5. Conversion de l'empreinte en format numérique (chiffres de 0 à 9)  
Les 64 bits de poids fort de l'empreinte sont convertis en base 10 afin d'obtenir une chaîne composée de 20 chiffres. Si le nombre obtenu s'exprime sur moins de 20 chiffres, il est complété à gauche par des zéros.
6. Calcul de l'INS-C  
L'INS-C est constitué du nombre de 20 chiffres obtenu à l'étape précédente complété par une clé de contrôle sur 2 chiffres.  
La clé de contrôle est calculée selon la même règle que la clé du NIR : la clé est égale au complément à 97 du reste de la division euclidienne du nombre de 20 chiffres par 97.




Les exemples de code publiés dans ce document concernent les étapes 3 à 5.

Pour la récupération des données nécessaires au calcul de l'INS-C, nous vous invitons à visiter le site Internet du GIE SESAM-VITALE pour obtenir des informations sur les différentes possibilités de lecture des données stockées dans la carte Vitale.

## 3 Exemples de code

Ce chapitre contient autant de sous chapitre que d'étapes illustrés par des exemples de code.

Le code couleur suivant est proposé pour faciliter l'identification du langage de programmation utilisé pour l'exemple de code :

	Code en langage Java
	Code en langage C
	Code en langage C#

### 3.1 Etape 3 : Constituer la graine

#### 3.1.1 Normaliser le prénom

Si le champ prénom de la carte Vitale du patient contient plusieurs prénoms (un prénom composé par exemple), il s'agit dans une première étape de supprimer tous les espaces, puis de retenir les 10 premiers caractères de la chaîne ainsi obtenue. Si la chaîne résultante est constituée de moins de 10 caractères, elle est complétée à droite par des espaces (0x20).

## Exemple de code en JAVA

```
//  
// Fonction qui permet de normaliser le prenom pour le calcul du INS-  
//  
public static String normaliserPrenoms(String prenomEntree) {  
    String prenomSortie = "";  
    String[] result = prenomEntree.split(" ");  
  
    for (int i = 0; i < result.length; i++) {  
        DmpInscLog.debug(result[i]);  
        prenomSortie = prenomSortie.concat(result[i]);  
  
    }  
  
    //  
    // Cas ou le prenom contient moins de 10 caracteres  
    //  
    if (prenomSortie.length() < PRENOM_SIZE) {  
  
        StringBuffer buffer = new StringBuffer(" ");  
        DmpInscLog.debug(buffer.toString());  
        //  
        // On prend les premiers caracteres du prenom  
        //  
        buffer.replace(0, prenomSortie.length(), prenomSortie);  
  
        DmpInscLog.debug(buffer.toString());  
  
        prenomSortie = buffer.toString();  
  
    }  
  
    if (prenomSortie.length() >= PRENOM_SIZE) {  
  
        prenomSortie = prenomSortie.substring(0, PRENOM_SIZE);  
  
    }  
  
    DmpInscLog.debug("Le prenom normalise est :"+ prenomSortie);  
  
    return prenomSortie;  
}
```

## Exemple de code en Langage C

```
#ifdef __STDC__
static int _insc_normalise_prenom(const char *prenom, char
*prenom_normalise)
#else
static int _insc_normalise_prenom(prenom, prenom_normalise)
const char *prenom;
char *prenom_normalise;
#endif
{
    int _cpt=-1, cpt=-1;
    size_t lg_prenom;
    char *_prenom;
    if (prenom_normalise==NULL)
    {
        _insc_erreur("_insc_normalise_prenom","prenom_normalise est NULL" ,
1, e_insc_ko);
        return(e_insc_ko);
    }
    /*
     * Suppression des espaces
     */
    lg_prenom=strlen(prenom);
    _prenom=malloc(lg_prenom+1);
    strcpy(_prenom,prenom);
    _cpt=0;
    for (cpt=0; cpt < lg_prenom; cpt++)
    {
        if (!isspace(prenom[cpt]))
        {
            _prenom[_cpt]=prenom[cpt];
            _cpt++;
        }
    }
    for (cpt=0; cpt < _cpt && cpt<c_insc_min_lg_prenom; cpt++)
        prenom_normalise[cpt]=_prenom[cpt];
    free(_prenom);
    /*
     * Ajout d'espaces a droite si la longueur du prenom normalise est < a 10
     */
    while (_cpt < c_insc_min_lg_prenom)
    {
        prenom_normalise[_cpt]=' ';
        _cpt++;
    }
    /*
     * Troncature a 10 caracteres
     */
    prenom_normalise[c_insc_min_lg_prenom]='\0';
    if (v_insc_debug>=c_insc_debug_2) {
        fprintf(stderr, "_insc_normalise_prenom, prenom normalise : <%s>\n",
prenom_normalise);
        fflush(stderr);
    }
    /*
     * Journalisation
     */
    strcpy(v_insc_calc_insc_trace.prenom_normalise,prenom_normalise);
    return(e_insc_ok);
}
```



### 3.1.2 Former la graine

**La graine est constituée par la concaténation dans l'ordre du prénom « normalisé » avec la date de naissance et le NIR.**

```
// Fonction pour former la graine
//
// Concatenation dans l'ordre du prénom normalise, de la date de naissance
// et du NIR normalise
//
private String formerGraine(String prenom, String dateNaissance, String NIR) {

    String graine = "";

    if (prenom == null)
        return null;

    if (dateNaissance == null)
        return null;

    if (NIR == null)
        return null;

    graine = graine.concat(NormaliserDonnees.normaliserPrenoms(prenom))
                .concat(dateNaissance).concat(NIR);

    DmpInscLog.debug("La chaine qui constitue la graine est <" + graine + ">");

    return graine;

}
```

## Exemple de code en Langage C

```
#ifndef __STDC__
extern int _insc_normalise_graine(const char * prenom, const char * date_naissance,
const char * nir, char * graine)
#else
extern int _insc_normalise_graine(prenom, date_naissance, nir, graine)
const char * prenom;
const char * date_naissance;
const char * nir;
char * graine;
#endif
{
char prenom_normalise[c_insc_min_lg_prenom+1];
int status=-1;

if (graine==NULL)
{
_insc_erreur("_insc_normalise_graine","graine est NULL" , 1, e_insc_fatal);
return(e_insc_fatal);
}

/*
 * Normalisation prealable du prenom
 */
memset(prenom_normalise, '\0', sizeof(prenom_normalise));
status=_insc_normalise_prenom(prenom, prenom_normalise);
if (status!=e_insc_ok)
{
_insc_erreur("_insc_normalise_graine","_insc_normalise_prenom" , 1, status);
return(e_insc_ko);
}

/*
 * Concatenation dans l'ordre du prenom normalise avec la date de naissance et le
nir
 */
graine[0]='\0';
strcpy(graine,prenom_normalise);
strcat(graine,date_naissance);
strcat(graine,nir);

if (v_insc_debug>=c_insc_debug_2) {
fprintf(stderr, "_insc_normalise_graine, graine : <%s>\n", graine);
fflush(stderr);
}

/*
 * Journalisation
 */
strcpy(v_insc_calc_insc_trace.graine,graine);

return(e_insc_ok);
}
```

## Exemple de code en C #

```
//  
// Fonction pour former la graine  
// Concatenation dans l'ordre du prenom normalise, de la date de naissance et du  
// NIR normalise  
//  
//  
private string FormerGraine(string prenom, string dateNaissance, string NIR)  
{  
    string graine = "";  
  
    if (prenom == null)  
        return null;  
  
    if (dateNaissance == null)  
        return null;  
  
    if (NIR == null)  
        return null;  
  
    graine = graine + NormaliserDonnees.NormaliserPrenoms(prenom) + (dateNaissance) +  
    NIR;  
  
    Logger.Info("La chaine qui constitue la graine est <" + graine + ">");  
  
    return graine;  
}
```

## 3.2 Etape 4 : Calculer l’empreinte

Le haché est une empreinte SHA256 de la chaîne de caractère ‘Graine’

### Exemple de code en JAVA

```
//  
// Fonction qui retourne le SHA 256 de la chaîne passée en entrée,  
// sous la forme d'une chaîne de 256 caractères Hexa  
//  
private String ShaINSC(String graine) {  
  
    byte[] hash;  
    if (graine == null) {  
  
        return null;  
  
    }  
  
    //  
    // Calcul de l'empreinte de la graine  
    //  
    MessageDigest md = null;  
  
    try {  
        md = MessageDigest.getInstance("SHA-256");  
  
    } catch (NoSuchAlgorithmException e) {  
  
        DmpInscLog.error("erreur dans l'algorithme " + e.getMessage());  
  
    }  
    hash = md.digest(graine.getBytes());  
  
    //  
    // Conversion de l'empreinte  
    //  
    return byteToHex(hash);  
}
```

## Exemple de code en Langage C

```
#ifdef __STDC__
static int _insc_sha256(const char * donnees, char *sha2)
#else
static int _insc_sha256(donnees, sha2)
const char * donnees;
char * sha2;
#endif
{
    int lg;
    SHA256_CTX ctx256;
    if (donnees==NULL)
    {
        _insc_erreur("_insc_sha256","donnees est NULL" , 1,
e_insc_fatal);
        return(e_insc_fatal);
    }
    if (sha2==NULL)
    {
        _insc_erreur("_insc_sha256","sha2 est NULL" , 1,
e_insc_fatal);
        return(e_insc_fatal);
    }
    /*
     * Appel à une fonction de hachage SHA256
     */
    SHA256_Init(&ctx256);
    SHA256_Update(&ctx256, (unsigned char*)donnees, strlen(donnees));
    SHA256_End(&ctx256, sha2);
    lg=strlen(sha2);
    if (lg!=c_insc_max_lg_sha256)
    {
        _insc_erreur("_insc_sha256","empreinte de longueur incorrecte"
, lg, e_insc_fatal);
        _insc_erreur("_insc_sha256","empreinte de longueur incorrecte"
, 1, e_insc_fatal);
        return(e_insc_fatal);
    }
    if (v_insc_debug>=c_insc_debug_2) {
        fprintf(stderr, "_insc_sha256, empreinte sha2 : <%s>\n",sha2);
        fflush(stderr);
    }
    /*
     * Journalisation
     */
    strcpy(v_insc_calc_insc_trace.sha2,sha2);
    return(e_insc_ok);
}
```

## Exemple de code en C #

```
//  
// Fonction qui retourne le SHA 256 de la chaine passee en entree,  
// sous la forme d'une chaine de 256 caracteres Hexa  
//  
public string ShaINSC(string graine)  
{  
    if (graine == null)  
    {  
        return null;  
    }  
    //  
    // Calcul de l'empreinte de la graine  
    //  
    SHA256 md = SHA256.Create();  
    byte[] gr = Encoding.UTF8.GetBytes(graine);  
    byte[] hash = md.ComputeHash(gr);  
    StringBuilder result = new StringBuilder();  
    for (int i = 0; i < hash.Length; i++)  
    {  
        // Affiche le Hash en hexadecimal  
        result.Append(hash[i].ToString("x2"));  
    }  
    return result.ToString();  
}
```

### 3.3 Etape 5 : Convertir l’empreinte en numérique

**Conversion de l’empreinte SHA256 en format numérique (chiffres de 0 à 9). Les 64 bits de poids fort de l’empreinte sont convertis en base 10 afin d’obtenir une chaîne composée de 20 chiffres. Si le nombre obtenu s’exprime sur moins de 20 chiffres, il est complété à gauche par des zéros.**

#### Exemple de code en JAVA

```
/
// Fonction qui permet :
//
// - d'extraire les 64 bits (16 caracteres Hexa) les plus forts (a gauche) du
// sha256 (256 caracteres Hexa)
// - de les convertir ensuite en chaîne décimale
// - puis de calculer la cle de l'INS-C
//
private String[] formerINSC(String shaHex) {

    //
    // Extraction des 16 premiers caracteres a gauche
    //
    shaHex = shaHex.substring(0, 16);

    DmpInscLog.debug("Les 16 premieres chiffres sont : " + shaHex);

    //
    // Conversion en decimale
    //
    BigInteger dix = new BigInteger(shaHex, 16);

    DmpInscLog.debug("Decimal des 16 premiers chiffres : " + dix.toString());

    //
    // Calcul de la cle
    //
    BigInteger temp = dix.mod(new BigInteger("97"));
    int maCle = 97 - temp.intValue();

    DmpInscLog.debug("Cle : " + maCle);

    return new String[] { dix.toString(), "" + maCle };

}
```

## Exemple de code en Langage C

```
#ifdef __STDC__
static int _insc_sha256_chaine_numerique(const char *sha2, char
*sha2_chaine_numerique)
#else
static int _insc_sha256_chaine_numerique(sha2, sha2_chaine_numerique)
const char *sha2;
char *sha2_chaine_numerique;
#endif
{
    int status=-1, cpt, lg;
    unsigned long long base, puissance, chiffre, sha2_numerique, increment;

    if (sha2==NULL)
    {
        _insc_erreur("_insc_sha256_chaine_numerique","sha2 est NULL" , 1,
e_insc_fatal);
        return(e_insc_fatal);
    }

    if (sha2_chaine_numerique==NULL)
    {
        _insc_erreur("_insc_sha256_chaine_numerique","sha2_chaine_numerique
est NULL" , 1, e_insc_fatal);
        return(e_insc_fatal);
    }

    sha2_numerique=0;
    puissance=16;
    base=16;
    for(cpt=0; cpt<strlen(sha2); cpt++)
    {
        puissance--;
        chiffre=0;
        status= _insc_hexa_numerique(sha2[cpt], &chiffre);
        if (status != e_insc_ok)
        {
            _insc_erreur("_insc_sha256_chaine_numerique","_insc_hexa_numerique" , 1, status);
            return(e_insc_fatal);
        }

        increment=chiffre*powl(base,puissance);
        sha2_numerique=sha2_numerique+increment;

        if (v_insc_debug>=c_insc_debug_3) {
            fprintf(stderr, "sha2_chaine_numerique, caractere lu : <%c>
valeur numerique du caractere <%llu>\n",sha2[cpt], chiffre);
            fprintf(stderr, "sha2_chaine_numerique, puissance courante :
<%llu>\n",puissance);
            fprintf(stderr, "sha2_chaine_numerique, increment :
<%llu>\n",increment);
            fprintf(stderr, "sha2_chaine_numerique, sha2_numerique :
<%llu>\n", sha2_numerique);
            fflush(stderr);
        }

        if (puissance == 0)
            break;
    }

    if (v_insc_debug>=c_insc_debug_2) {
        fprintf(stderr, "sha2_chaine_numerique, sha2_numerique : <%llu>\n",
sha2_numerique);
    }
}
```



```
        fflush(stderr);
    }

    sprintf(sha2_chaine_numerique,"%llu",sha2_numerique);
    lg=strlen(sha2_chaine_numerique);

    if (lg>c_insc_max_lg_insc)
    {
        _insc_erreur("_insc_sha256_chaine_numerique","sha2_numerique trop
long" , lg, e_insc_fatal);
        _insc_erreur("_insc_sha256_chaine_numerique","sha2_numerique trop
long" , 1, e_insc_fatal);
        return(e_insc_fatal);
    }

    if (v_insc_debug>=c_insc_debug_2) {
        fprintf(stderr, "sha2_chaine_numerique, empreinte sha2 convertie en
chaine numerique : <%s>\n",sha2_chaine_numerique);
        fflush(stderr);
    }

    /*
     * Journalisation
     */
    strcpy(v_insc_calc_insc_trace.sha2_chaine_numerique,sha2_chaine_numerique);

    return(e_insc_ok);
}
```

## Exemple de code en C #

```
//  
// Fonction qui permet :  
//  
// - d'extraire les 64 bits(16 caracteres Hexa) les plus forts (a gauche) du  
sha256(256 caracteres Hexa)  
// - de les convertir ensuite en chaine decimale  
// - puis de calculer la cle de l'INS-C  
//  
public string[] FormerINSC(string shaHex)  
{  
    //  
    // Extraction des 16 premiers caracteres a gauche  
    //  
    shaHex = shaHex.Substring(0, 16);  
  
    Logger.Info("Les 16 premieres chiffres sont : " + shaHex);  
    //  
    // Conversion en decimale  
    //  
    ulong dix = ulong.Parse(shaHex, System.Globalization.NumberStyles.HexNumber);  
  
    Logger.Info("Decimal des 16 premiers chiffres : " + dix);  
  
    //  
    // Calcul de la cle  
    //  
    ulong temp = dix % 97;  
  
    int maCle = System.Convert.ToInt16(97 - temp);  
  
    Logger.Info("La cle de l'INS-C est : " + maCle);  
  
    return new string[] { "" + dix, "" + maCle };  
}
```

## 3.4 Etape 6 : Calcul de l'INS-C

L'INS-C est constitué par le nombre de 20 chiffres obtenu lors de l'étape précédente complété par une clé de contrôle sur 2 chiffres.  
La clé de contrôle est calculée selon la même règle que la clé du NIR : elle est égale au complément à 97 du reste de la division euclidienne du nombre de 20 chiffres par 97.

### Exemple de code en JAVA

Code de calcul du complément à 97 du modulo 97 : voir le code exemple fournie au niveau de l'étape 6.

### Exemple de code en Langage C

```
#ifdef __STDC__
static int _insc_calc_cle(const char * donnees, char *cle_insc)
#else
static int _insc_calc_cle(donnees, cle_insc)
const char * donnees;
char * cle;
#endif
{
    unsigned long long cle_insc_numerique;

    if (donnees==NULL)
    {
        _insc_erreur("_insc_calc_cle","donnees est NULL" , 1, e_insc_fatal);
        return(e_insc_fatal);
    }

    if (cle_insc==NULL)
    {
        _insc_erreur("_insc_calc_cle","cle_insc est NULL" , 1, e_insc_fatal);
        return(e_insc_fatal);
    }

    cle_insc_numerique=strtoull(donnees,NULL,10);

    if (v_insc_debug>=c_insc_debug_3) {
        fprintf(stderr, "_insc_calc_cle, cle sous forme numerique :
<%llu>\n",cle_insc_numerique);
        fflush(stderr);
    }

    cle_insc_numerique=cle_insc_numerique%97;

    if (v_insc_debug>=c_insc_debug_3) {
        fprintf(stderr, "_insc_calc_cle, reste la division par 97 :
<%llu>\n",cle_insc_numerique);
        fflush(stderr);
    }

    cle_insc_numerique=97-cle_insc_numerique;
    sprintf(cle_insc,"%llu",cle_insc_numerique);

    if (v_insc_debug>=c_insc_debug_2) {
        fprintf(stderr, "_insc_calc_cle, cle INSC calcule : <%s>\n",cle_insc);
        fflush(stderr);
    }
}
```

```
    return(e_insc_ok);  
}
```

### **Exemple de code en C #**

Code de calcul du complément à 97 du modulo 97 : voir le code exemple fournie au niveau de l'étape 6.

## 3.5 Algorithme de calcul de l'INS-C

Ce paragraphe présente l'enchaînement des 5 étapes précédentes.

### Exemple de code en JAVA

```
public INSCResultats calculerINSC(String prenom, String dateNaissance,
                                   String nir, String cleNIR) {

    int ret = 0;

    resultats.init();

    //
    // Etape 1 : Verification des donnees
    //
    if ((ret = ControlerDonnees.verifPrenomsOK(prenoms)) == 0) {

        if ((ret = ControlerDonnees.verifDateNaissanceOk(dateNaissance)) == 0) {

            if ((ret = ControlerDonnees.verifNIROK(nir, cleNIR)) == 0) {

                resultats.setCode_retour(ret);
                DmpInscLog.debug("Les données en entrée sont au bon format");

            } else {

                DmpInscLog.debug("Le NIR n'est pas au bon format");
                resultats.setCode_retour(ret);
                return resultats;
            }

        } else {

            DmpInscLog.debug("La date de naissance n'est pas au bon format");

            resultats.setCode_retour(ret);

            return resultats;
        }
    } else {

        DmpInscLog.debug("Les Prenoms ne sont pas dans le bon format");

        resultats.setCode_retour(ret);

        return resultats;
    }

    //
    // Etape 2 : Former la graine
    //
    String graine = formerGraine(prenoms, dateNaissance, nir);
    resultats.setGraine(graine);

    //
    // Etape 3 : Calculer le Sha256 sous forme Hexadecimale sur 64
    // caracteres
    //
```

```
String sha2 = ShaINSC(graine);
resultats.setSha2(sha2);

//
// Etape 4 : Calculer le INSC
//
String[] inscTab = formerINSC(sha2);
resultats.setSha2_chaine_numerique(inscTab[0]);

//
// Etape 5 : Normaliser le INSC et sa cle
//
String[] insc = NormaliserDonnees.normaliserINSC(inscTab);
resultats.setSha2_chaine_numerique_normalise(insc[0]);
resultats.setCle_insc(insc[1]);

//
// Resultat final
//
resultats.setInsc(insc[0] + insc[1]);

DmpInscLog.info("L'INS-C de <" + prenom + "> est <"
               + resultats.getCle_insc() + ">");

return resultats;
}
```

## Exemple de code en Langage C

```
#ifndef __STDC__
_INSC_INT_EXPORT _insc_calc(t_insc_calc_insc_trace * insc_trace, const char *
prenom, const char * date_naissance, const char * nir, const char * cle_nir, char *
insc)
#else
_INSC_INT_EXPORT _insc_calc(insc_trace, prenom, date_naissance, nir, insc)
t_insc_calc_insc_trace * trace;
const char * prenom;
const char * date_naissance;
const char * nir;
const char * cle_nir;
char * insc;
#endif
{
int status=-1;
char graine[c_insc_max_lg_graine+1];

memset(&v_insc_calc_insc_trace, '\0', sizeof(t_insc_calc_insc_trace));
v_insc_calc_insc_trace.debug=c_insc_debug_0;
if (insc_trace!=NULL)
{
v_insc_calc_insc_trace.debug=insc_trace->debug;
if (insc_trace->debug < c_insc_debug_min)
v_insc_calc_insc_trace.debug=c_insc_debug_0;
if (insc_trace->debug > c_insc_debug_max)
v_insc_calc_insc_trace.debug=c_insc_debug_0;
}
v_insc_debug=v_insc_calc_insc_trace.debug;

/*
* Controle des parametres en entree
*/
status=_insc_controle_prenom(prenom);
if (status!=e_insc_ok)
{
_insc_erreur("_insc_calc", "_insc_controle_prenom" , 1, status);
if (insc_trace!=NULL)

memcpy(insc_trace, &v_insc_calc_insc_trace, sizeof(t_insc_calc_insc_trace));
return(status);
}

status=_insc_controle_date_naissance(date_naissance);
if (status!=e_insc_ok)
{
_insc_erreur("_insc_calc", "_insc_controle_date_naissance" , 1, status);
if (insc_trace!=NULL)

memcpy(insc_trace, &v_insc_calc_insc_trace, sizeof(t_insc_calc_insc_trace));
return(status);
}

status=_insc_controle_nir_avec_cle(nir, cle_nir);
if (status!=e_insc_ok)
{
_insc_erreur("_insc_calc", "_insc_controle_nir_avec_cle" , 1, status);
if (insc_trace!=NULL)

memcpy(insc_trace, &v_insc_calc_insc_trace, sizeof(t_insc_calc_insc_trace));
return(status);
}

/*
* Normalisation puis calcul de la graine
```

```
*/
memset(graine, '\\0', sizeof(graine));
status=__insc_normalise_graine(prenom, date_naissance, nir, graine);
if (status!=e_insc_ok)
{
    _insc_erreur("__insc_calc", "__insc_normalise_graine" , 1, status);
    if (insc_trace!=NULL)

        memcpy(insc_trace, &v_insc_calc_insc_trace, sizeof(t_insc_calc_insc_trace));
    return(status);
}

/*
 * Calcul de l'identifiant
 */
status=__insc_calc(graine, insc);
if (status!=e_insc_ok)
{
    _insc_erreur("__insc_calc", "__insc_calc" , 1, status);
    if (insc_trace!=NULL)

        memcpy(insc_trace, &v_insc_calc_insc_trace, sizeof(t_insc_calc_insc_trace));
    return(status);
}

if (insc_trace!=NULL)
    memcpy(insc_trace, &v_insc_calc_insc_trace, sizeof(t_insc_calc_insc_trace));
return(e_insc_ok);
}
```



## Exemple de code en C #

```
//  
// Methode principale  
//  
public INSCResultats CalculerINSC(string prenoms, string dateNaissance,  
                                string nir, string cleNIR)  
{  
  
    int ret = 0;  
  
    resultats.init();  
  
    //  
    // Etape 1 : Verifications des donnees  
    //  
    if ((ret = ControlerDonnees.VerifPrenomsOK(prenoms)) == 0)  
    {  
        if ((ret = ControlerDonnees.VerifDateNaissanceOk(dateNaissance)) == 0)  
        {  
            if ((ret = ControlerDonnees.VerifNIROK(nir, cleNIR)) == 0)  
            {  
                Logger.Info("Les donn  es en entr  e sont au bon format");  
                resultats.Code_Retour = ret;  
  
            }  
            else  
            {  
                Logger.Info("Le NIR n'est pas au bon format");  
                resultats.Code_Retour = ret;  
                return resultats;  
            }  
        }  
        else  
        {  
            Logger.Info("La date de naissance n'est pas au bon format");  
            resultats.Code_Retour = ret;  
            return resultats;  
        }  
    }  
    else  
    {  
        Logger.Info("Les prenoms ne sont pas dans le bon format");  
        resultats.Code_Retour = ret;  
        return resultats;  
    }  
    //  
    // Etape 2 : Former la graine  
    //  
    string graine = FormerGraine(prenoms, dateNaissance, nir);  
    resultats.Graine = graine;  
  
    //  
    // Etape 3 : calculer le Sha256 sous forme Hexadecimale sur 64  
    // caracteres  
    //  
    string sha2 = ShaINSC(graine);  
    resultats.Sha2 = sha2;  
    //
```

```
// Etape 4 : Calculer le INSC
//
string[] inscTab = FormerINSC(sha2);
resultats.Sha2_chaine_numerique = inscTab[0];

//
// Etape 5 : Normaliser le INSC et sa cle
//
string[] insc = NormaliserDonnees.NormaliserINSC(inscTab);
resultats.Sha2_chaine_numerique_normalise = insc[0];
resultats.Cle_insc = insc[1];
Logger.Info("Le INS-C de <" + prenoms + "> est <"
            + resultats.Sha2_chaine_numerique_normalise + ">");
return resultats;
}
```

## 4 Annexe : références externes

Lien vers le site Internet de l'ASIP Santé : <http://www.asipsante.fr>

Lien vers le site Internet du GIE-Vitale : <http://www.sesam-vitale.fr>