

# Identity Functions

FASC-N Open Source Description

Russell Davis  
12/2/2011

## Contents

Preface .....	1
From FIPS 201-1 .....	1
From SP 800-73 .....	2
Program Descriptions.....	2
Agency Code .....	2
Convert Character .....	2
Credential Number .....	3
Credential Series .....	4
Expand FASC-N.....	5
Individual Credential.....	6
Organizational Category .....	6
Organizational Identifier .....	7
Personal Identifier.....	8
Character Return.....	8
Number Return .....	8
Person Organization Association .....	9
System Code.....	9
To Integer Conversion.....	10
Test Fixtures .....	11
fascn .....	11
fascnOut .....	12
Longitudinal Redundancy Character .....	12
fascnParse .....	13
Source Code Listing .....	13
Agency Check .....	15
Outputs .....	15
Source Code .....	16

## Preface

Key (privileged) personnel within the VA and DoD will have a valid identification cards that meets the requirements in Homeland Security Presidential Directive 12 (HSPD-12), "Policy for a Common Identification Standard for Federal Employees and Contractors." Direct results of HSPD-12 are Common Access Card (CAC)/Personal Identity Verification (PIV) cards.

Furthermore, the Department of Defense is using EDI-PI as a unique personal identifier. This value is located in multiple locations within the CAC/PIV card within the Federal Agency Smart Credential Number (FASC-N). In turn, the FASC-N is located in the Cardholder Unique Identifier (CHUID) and as part of the authentication digital certificate. What's more, the CHUID is available as a free read from the contactless ISO 14443 interfaces as well as the contact interface. Consequently, the CHUID/FASC-N could be extracted hands free or through the contact interface. Additionally, the digital certificate used for CAC/PIV login includes the FASC-N.

Thinking outside the box, a surgeon could authenticate inside an operating room using the contactless interface. One would need only to move the CAC/PIV to a configured ISO 14443 reader. Similarly, a user logging into a system using the contact interface could be identified from the digital certificate used.

The FASC-N contains information that identifies the Federal agency that issued the CAC/PIV card. This is not limited to the DoD and VA but includes all Government users such as HHS and FDA. As a final note, this guarantees unique and positive identification of the cardholder.

Note: The FASC-N was designed to work with Physical Access Control Systems (PACS). Many of the legacy systems include twisted pair cable and are challenged by serial communications. As a consequence, the FASC-N includes many parity checks to ensure the integrity of the transmission. Moreover, the data was specified for serial transmission. This necessitated a communications view when writing the functions. Also, note there are many values within the FASC-N that are currently unused. These could be used to specify employee attributes to deep levels of granularity and still be HSPD-12 compliant.

## From FIPS 201-1

To better understand what the CAC/PIV card includes, several requirement document descriptions are included.

*The PIV CHUID shall be accessible from both the contact and contactless interfaces of the PIV Card without card activation. The PIV FASC-N shall not be modified post-issuance.*

### 4.2.1 PIV CHUID Data Elements

*In addition to the mandatory FASC-N that identifies a PIV Card, the CHUID shall include an expiration date data element in machine readable format that specifies when the card expires. The expiration date format and encoding rules are as specified in [SP 800-73]. For PIV Cards, the format of the asymmetric signature field is specified in Section 4.2.2.*

*The PIV Card shall store a corresponding X.509 certificate to support validation of the public key. The X.509 certificate shall include the FASC-N in the subject alternative name extension using the pivFASC-N attribute to support physical access procedures.*

*Certificates that contain the FASC-N in the subject alternative name extension, such as PIV Authentication certificates and Card Authentication certificates, shall not be distributed publicly (e.g., via LDAP or HTTP accessible from the public Internet).*

*FASC-N Identifier: The FASC-N shall be in accordance with [SP 800-73]. A subset of FASC-N, a FASC-N Identifier, is a unique identifier as described in [SP 800-73].*

## From SP 800-73

*The Federal Agency Smart Credential Number (FASC-N) shall be SCEPACS [4]. A subset of the FASC-N, the FASC-N Identifier, shall be the unique identifier as described in [4, 6.6]: “The combination of an Agency Code, Credential Number is a fully qualified number that is uniquely assigned to individual”.*

## Program Descriptions

Functions are provided to convert each value within the FASC-N into a number. With the current generation 64-bit computers, even the longest number, the Personal Identifier, can be loaded into a single register or memory location as a 64-bit integer (or less). In turn, this allows efficient computation. Also note, the FASC-N was specified for a serial communications environment.

## Agency Code

```
/******
```

As noted in the Technical Implementation Guidance:  
Smart Card Enabled Physical Access Control Systems

AGENCY CODE

Identifies the government agency issuing the credential

Dependencies:

Requires the `toint()` function that converts a character in the range '0' to '9' to an integer. If the character converted is not '0' - '9' the function returns a value of 0.

```
*****/
```

```
int agency_code(fascn)
    char fascn[40];
{
    int temp;
    temp = toint(fascn[1]) * 1000;
    temp += toint(fascn[2]) * 100;
    temp += toint(fascn[3]) * 10;
    temp += toint(fascn[4]);
    return(temp);
}
```

## Convert Character

The FASC-N is using Binary Coded Decimals (BCDs) that include a parity bit. If you count the number of 1's it must be an odd number. Also note, the values 'A' through 'F' are normally valid BCD characters. However, the values for 'B', 'D', and 'F' are used for special separator values. For example, unlike opening a file where we know where the first character is, the FASC-N uses a start sentinel to identify the start of a new FASC-N. As the meaningful values are extracted from the FASC-N, the Start Sentinel, Stop Sentinel, Field Separator, and the calculated LRC values have no meaning outside of the FASC-N

```
/******
```

The convert number uses the PACS table as a conversion function. This is a diagnostic function to display the FASC-N values as characters.

Dependencies:

The external value for error must be defined

```
*****/
```

```
char convertCharacter(number)
char number;

{
char return_value;
switch (number) {
case 0x01: { return_value = '0'; /* 0000 0001 */
break; }
case 0x10: { return_value = '1'; /* 0001 0000 */
break; }
case 0x08: { return_value = '2'; /* 0000 1000 */
break; }
case 0x19: { return_value = '3'; /* 0001 1001 */
break; }
case 0x04: { return_value = '4'; /* 0000 0100 */
break; }
case 0x15: { return_value = '5'; /* 0001 0101 */
break; }
case 0x0d: { return_value = '6'; /* 0000 1101 */
break; }
case 0x1c: { return_value = '7'; /* 0001 1100 */
break; }
case 0x02: { return_value = '8'; /* 0000 0010 */
break; }
case 0x13: { return_value = '9'; /* 0001 0011 */
break; }
case 0x1a: { return_value = 'S'; /* Start Sentinel 0001 1010 */
break; }
case 0x16: { return_value = 'F'; /* Field Separator 0001 0110 */
break; }
case 0x1f: { return_value = 'E'; /* End Sentinel 0001 1111 */
break; }
default: { return_value = error; /* not a valid character */
break; }
}
return(return_value);
}
```

## Credential Number

```
*****
```

As noted in the Technical Implementation Guidance:  
Smart Card Enabled Physical Access Control Systems

CREDENTIAL NUMBER

Encoded by the issuing agency. For a given system no duplicate numbers are active

The value is returned as a long integer (32-bits in length)

As noted in the Technical Implementation Guidance:  
Smart Card Enabled Physical Access Control Systems

#### Credential Number

In order to insure uniqueness of the fully qualified number assignment the Credential Number assignment is the responsibility of the CIO for the organization referenced by the Agency Code. Under the assigned Agency Code the CIO may not delegate the responsibility for Agency policy ensuring unique fully qualified number assignment to individuals. The authority to assign Credential Numbers may be delegated by the CIO.

Agency CIOs are responsible for insuring non-overlapping Credential Numbers are issued for all interoperable systems issuing FASC-N codes within their Agency.

The combination of an Agency Code, System Code and Credential Number is a fully qualified number that is uniquely assigned to a single individual.

#### Dependencies:

Requires the toint() function that converts a character in the range '0' to '9' to an integer. If the character converted is not '0' - '9' the function returns a value of 0.

```
*****/
```

```
long int credential_number(fascn)
    char fascn[40];
{
    long int temp;
    temp = toint(fascn[11]) * 100000;
    temp += toint(fascn[12]) * 10000;
    temp += toint(fascn[13]) * 1000;
    temp += toint(fascn[14]) * 100;
    temp += toint(fascn[15]) * 10;
    temp += toint(fascn[16]);
    return(temp);
}
```

## Credential Series

```
*****/
```

As noted in the Technical Implementation Guidance:  
Smart Card Enabled Physical Access Control Systems

#### CREDENTIAL SERIES (SERIES CODE)

Field is available to reflect major system changes

#### Dependencies:

Requires the toint() function that converts a character in the range '0' to '9' to an integer. If the character converted is not '0' - '9' the function returns a value of 0.

```
*****/
```

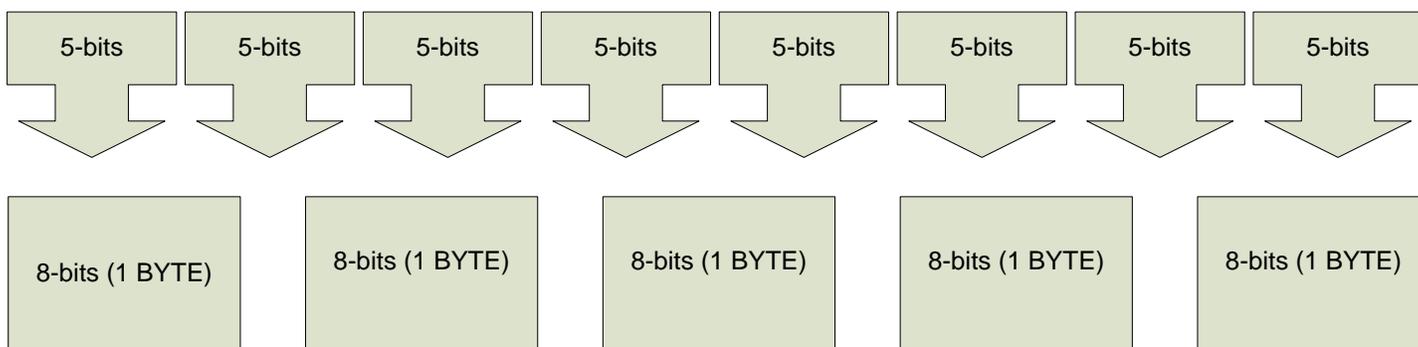
```

int credential_series(fascn)
    char fascn[40];
{
    int temp;
    temp = toint(fascn[18]);
    return(temp);
}

```

## Expand FASC-N

This is perhaps the most challenging part of the FASC-N conversion. That is, the FASC-N is a 25-Byte value that includes forty 5-bit characters. The characters are actually 4-bit Binary Coded Decimals (BCDs) that include a parity bit. In this parity approach, the number of 1's must be an odd number; thus making 5-bit characters. Although there are one 13 valid characters ('A', 'C', and 'E' are not used), they are converted back into the original 40-byte character array. In turn, the functions defined that extract specific values are converted to numbers. The next figure depicts how the 5-bit characters are combined into 8-bit bytes arrays. This necessitates breaking bytes into 5-bit chunks. For example, in the next figure, the second 5-bit character has 3-bits in the first byte and 2-bits in the second.



This is the only function that uses convertCharacter() and an error test was included. Also note, the last value, the LRC checksum, is not returned as it is not needed for the data extraction. For poor communications, the algorithm is included in the fascnOut program.

```

/*****
*
* This function was developed on an Intel architecture. The big endian once little
* endian issues must match the bit flow as if it were a communications stream. That is, the
* first bits to flow would be in character array location 0.
*
* The FASC-N encodes information as five-bit groups that are concatenated into as 25-Byte
* value. The purpose of the expand_fascn()function is to take the 25-byte FASC-N and expand it
* into a 40-Byte character array.

```

This function requires the value of error to be defined externally

```

*
*****/
#define msk 0x80 /* this is the most significant bit mask */
#define addBit 0x01 /* this is a character with the least significant bit set */

int expand_fascn(fascn25, fascn40)
    char fascn25[25], fascn40[40];
{
    int byte_count, bit_count, bit_index, expanded_index;
    char temp, temp2=0;

    for (bit_count = 0, byte_count = 0, expanded_index=0; byte_count < 25; byte_count++) {

```

```

temp = fascn25[byte_count];
for (bit_index=0; bit_index < 8; bit_index++) {
  if (msk & temp) /* is the most significant bit in temp a 1? */
    temp2 ^= addBit; /* Then change the least significant bit in temp2 to a 1 */

  temp <<= 1; /* shift left to prepare the next bit for testing */

  bit_count += 1;
  if ((bit_count % 5)== 0) { /* is this the 5th 1 or 0 printed? */
    temp2= convertCharacter(temp2);

    if (temp2==error)
      return(1);
    fascn40[expanded_index] = temp2; /* over write with value */
    expanded_index += 1; /* increment the expanded index */
    temp2 = 0;
  } /* end if */
  temp2 <<= 1;
} /* end inner for loop */
} /* end for */
return(0);
}

```

## Individual Credential

\*\*\*\*\*

As noted in the Technical Implementation Guidance:  
Smart Card Enabled Physical Access Control Systems

INDIVIDUAL CREDENTIAL ISSUE (CREDENTIAL CODE)  
Recommend coding as a "1" always

Dependencies:

Requires the toint() function that converts a character in the range '0' to '9' to an integer.  
If the character converted is not '0' - '9' the function returns a value of 0.

\*\*\*\*\*/

```

int individual_credential_issue(fascn)
  char fascn[40];
{
  int temp;
  temp = toint(fascn[20]);
  return(temp);
}

```

## Organizational Category

\*\*\*\*\*

The following is taken from the Technical Implementation Guidance:  
Smart Card Enabled Physical Access Control Systems Version 2.3

ORGANIZATIONAL IDENTIFIER  
OC=1 - NIST SP800-87 Agency Code  
OC=2 - State Code  
OC=3 - Company Code  
OC=4 - Numeric Country Code

For DoD, VA, and other federal users, a value returned of 1 indicated the  
4 character Organizational Identifier will contain the issuing Agency's code

as specified in the National Institute of Standards & Technology (NIST) Special Publication 800-87 (current revision is 1)

Dependencies:

Requires the toint() function that converts a character in the range '0' to '9' to an integer. If the character converted is not '0' - '9' the function returns a value of 0.

\*\*\*\*\*/

```
int organizational_category(fascn)
    char fascn[40];
{
    int temp;
    temp = toint(fascn[32]);
    return(temp);
}
```

## Organizational Identifier

/\*\*\*\*\*

When the Organizational Category is 1, then the Organizational Identifier identifies the Government entity that issued the FASC-N. The complete list is specified in the National Institute of Standards and Technology (NIST) Special Publication 800-87 (current revision is 1). Some values are listed below:

- 1200 AGRICULTURE, Department of
- 1300 COMMERCE, Department of
- 9700 DEFENSE, Department of (except military departments)
- 5700 AIR FORCE, Department of the (Headquarters, USAF)
- 2100 ARMY, Department of the (except Corps of Engineers Civil Program Financing)
- 1700 NAVY, Department of the
- 9100 EDUCATION, Department of
- 8900 ENERGY, Department of
- 7500 HEALTH AND HUMAN SERVICES, Department of
- 7000 HOMELAND SECURITY, Department of
- 8600 HOUSING AND URBAN DEVELOPMENT, Department of
- 1400 INTERIOR, Department of the
- 1500 JUSTICE, Department of
- 1600 LABOR, Department of
- 1900 STATE, Department of
- 6900 TRANSPORTATION, Department of
- 2000 TREASURY, Department of the
- 3600 VETERANS AFFAIRS, Department of
- 6800 ENVIRONMENTAL PROTECTION AGENCY
- 4700 GENERAL SERVICES ADMINISTRATION
- 8000 NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
- 8800 NATIONAL ARCHIVES AND RECORDS ADMINISTRATION
- 4900 NATIONAL SCIENCE FOUNDATION
- 2400 OFFICE OF PERSONNEL MANAGEMENT
- 2800 SOCIAL SECURITY ADMINISTRATION

Dependencies:

Requires the toint() function that converts a character in the range '0' to '9' to an integer. If the character converted is not '0' - '9' the function returns a value of 0.

\*\*\*\*\*/

```

int organizational_identifier(fascn)
    char fascn[40];
{
    int temp;
    temp = toint(fascn[33]) * 1000;
    temp += toint(fascn[34]) * 100;
    temp += toint(fascn[35]) * 10;
    temp += toint(fascn[36]);
    return(temp);
}

```

## Personal Identifier

Two versions of the 10-character PI are included. In the first, just the character array is extracted. If the device running the program cannot easily handle 64-bit integers, this may be a more efficient function to use. Note, when used character arrays, programmers need to be mindful of string terminations and bounds checking.

## Character Return

```

/*****

```

As noted in the Technical Implementation Guidance:  
Smart Card Enabled Physical Access Control Systems

PERSON IDENTIFIER:

Numeric Code used by the identity source to uniquely identify the token carrier. (e.g. DoD EDI PN ID, TWIC credential number, NASA UUPIC)

Note: the DoD and VA use the EDI-PI to uniquely identify people.

Users of string functions should ensure the proper termination of the string function.

```

*****/

```

```

int personalIdentifier(fascn, PI)
    char fascn[40], PI[10];
{
    int index;
    for (index=0; index<10; index++)
        PI[index] = fascn[22+index];
    return(0);
}

```

## Number Return

```

/*****

```

With the newer 64-bit computers and associated operating system, using 64-bit integers should be more efficient than comparing character strings. The PI\_number() takes the 10-digit Personal Identifier and converts it to a number. Note, the DoD and VA use an EDI-PI that is a 9-digit number with a concatenated checksum. Although the number only has 9-digits of meaningful information, applications might have been expanded to accommodate 10-digit numbers.

Dependencies:

Requires the toint() function that converts a character in the range '0' to '9' to an integer. If the character converted is not '0' - '9' the function returns a value of 0.

```
*****/
```

```
long long PI_number(fascn)
    char fascn[40];
{
    int index;
    long long PI;
    PI = ((long long)toint(fascn[22])*100000000);
    PI += ((long long)toint(fascn[23])*10000000);
    PI += ((long long)toint(fascn[24])*1000000);
    PI += ((long long)toint(fascn[25])*100000);
    PI += ((long long)toint(fascn[26])*10000);
    PI += ((long long)toint(fascn[27])*1000);
    PI += ((long long)toint(fascn[28])*100);
    PI += ((long long)toint(fascn[29])*10);
    PI += ((long long)toint(fascn[30])*1);
    PI += ((long long)toint(fascn[31]));
    return(PI);
}
```

## Person Organization Association

```
*****/
```

As noted in the Technical Implementation Guidance:  
Smart Card Enabled Physical Access Control Systems

### PERSON/ORGANIZATION ASSOCIATION CATEGORY

- 1 - Employee
- 2 - Civil
- 3 - Executive Staff
- 4 - Uniformed Service
- 5 - Contractor
- 6 - Organizational Affiliate
- 7 - Organizational Beneficiary

Dependencies:

Requires the toint() function that converts a character in the range '0' to '9' to an integer.  
If the character converted is not '0' - '9' the function returns a value of 0.

```
*****/
```

```
int person_organization_association(fascn)
    char fascn[40];
{
    int temp;
    temp = toint(fascn[37]);
    return(temp);
}
```

## System Code

```
*****/
```

As noted in the Technical Implementation Guidance:  
Smart Card Enabled Physical Access Control Systems

SYSTEM CODE

Identifies the system the card is enrolled in and is unique for each site

In order to ensure uniqueness of the fully qualified number assignment the System Code number assignment is the responsibility of the CIO for the organization referenced by the Agency Code. The authority to assign a single and blocks of System Codes may be delegated by the CIO.

Agency CIOs are responsible for ensuring non-overlapping System Codes are issued for all interoperable systems issuing SEIWG-012 credential number or FASC-N codes within their Agency.

The combination of each Agency Code and System Code permit one million unique fully qualified numbers. If a particular issuing system requires more than one million credentials issued then that system would require an additional system code assigned corresponding to each million credentials that will be issued by that system.

Dependencies:

Requires the toint() function that converts a character in the range '0' to '9' to an integer. If the character converted is not '0' - '9' the function returns a value of 0.

```
*****/
```

```
int system_code(fascn)
    char fascn[40];
{
    int temp;
    temp = toint(fascn[6]) * 1000;
    temp += toint(fascn[7]) * 100;
    temp += toint(fascn[8]) * 10;
    temp += toint(fascn[9]);
    return(temp);
}
```

## To Integer Conversion

```
*****/
```

This function converts a text character to an integer. Same function as found in ctype.h

Note:

In SP 800-87 rev 1, many of the agency codes include the hexadecimal values in the range 'A' to 'F'. However, the Technical Implementation Guidance: Smart Card Enabled Physical Access Control Systems [PACS] uses the table below. Thus, the FASC-N only recognizes 0 - 9. Moreover, the values for the Start Sentinel, Field Separator, and End Sentinel correspond to three of these BCD values so they cannot be used. Consequently, the default return is a "0" if the character is unrecognized.

```
*****/
```

```
int toint(ch)
    char ch;
{
    int return_value;
    switch (ch) {
    case '0': { return_value = 0;
               break; }
    }
```

```

case '1': { return_value = 1;
           break; }
case '2': { return_value = 2;
           break; }
case '3': { return_value = 3;
           break; }
case '4': { return_value = 4;
           break; }
case '5': { return_value = 5;
           break; }
case '6': { return_value = 6;
           break; }
case '7': { return_value = 7;
           break; }
case '8': { return_value = 8;
           break; }
case '9': { return_value = 9;
           break; }
default: return_value = 0;
}
return(return_value);
}

```

## Test Fixtures

Before the software could be tested, it was necessary to create a program to correct implementation of the FASC-N.

### fascn

This C language program

***fascn ac sc cn cs ici pi oc oi poa***

Where ***ac*** is the Agency Code

***sc*** is the System Code

***cn*** is the Credential Number

***cs*** is the Credential Series

***ici*** is the Individual Credential Issue

***pi*** is the Personal Identifier

***oc*** is the Organizational Category

***oi*** is the Organizational Identifier

***poa*** is the Person/Organization

There are 9 arguments passed in the order listed and all are characters. Using the reference values provided in [Technical Implementation Guidance: Smart Card Enabled Physical Access Control Systems \[PACS\]](#)

AGENCY CODE = 0032

SYSTEM CODE = 0001

CREDENTIAL# = 092446

CS = 0

ICI = 1

PI = 1112223333

OC = 1



```

/* add up everything using XOR (mode 2 addition) */
for (checksum=0, count=0; count < 39; count++)
    checksum ^= expanded_fascn[count];

/* shit the temporary working value one place to the right */
temp_char = (checksum >> 1); /* right shift to delete parity bit */

/* sum the number of bits */
for (count = 0; count < 4; count++) {
    if (temp_char & 0x01)
        ones += 1; /* increment the number of ones */
    temp_char >>= 1; /* right shift */
}

/* See if the number of 1's is an even or odd number where */
/* a 1 indicates odd and a 0 even */

if ((ones % 2) == 1)
    checksum &= 0x1e; /* Make the least significant bit 0 */
else
    checksum |= 0x01; /* Make the least significant bit 1 */

/* Expand the checksum (LRC) at offset 39 */

expanded_fascn[39] = checksum;
return(0);
}

```

## fascnParse

This program combines each of the source files. After compilation, the program produces the following output:

```

The Personal Identifier is: 1112223333
The PI as a number is: 1112223333
The Agency Code is: 32
The System Code is: 1
The credential number is: 92446
The credential series is: 0
The individual credential issue is: 1
The Organizational Category is: 1
The Organizational Identifier is: 1223
The Person/Organization Association Category is: 2

```

Again, this is consistent with the [PACS] reference.

## Source Code Listing

```

#include <stdio.h>

#define error 'Z' /* The error return value */

```

```

char padded_fascn[25]; /* This is the 25-byte FASC-N format on the CAC/PIV card */
char expanded_fascn[40]; /* The 40-byte expanded FASC-N format */
char PI[10]; /* A text return of the personal identifier. Note, programmers must check array
bounds */

```

```

/*****

```

```

read_fascn() will read in the binary FASC-N from a default file fascn.dat
While this is adequate for testing, a different delivery is expected in practice.

```

```

*****/

```

```

int read_fascn()
{
    int byteCount;
    FILE *fp;
    if ((fp = fopen("fascn.dat", "r")) == NULL)
        return(1);
    for(byteCount=0; byteCount<25; byteCount++)
        padded_fascn[byteCount] = fgetc(fp);
    fclose(fp);
    return(0);
}

```

```

#include "toint.c"
#include "convertCharacter"
#include "expandFascn.c"
#include "personalIdentifier.c" /* text return of PI */
#include "PINumber.c" /* Number return of PI */
#include "agencyCode.c"
#include "systemCode.c"
#include "credentialNumber.c"
#include "credentialSeries.c"
#include "individualCredential.c"
#include "organizationalCategory.c" /* Is NIST SP 800-87 used? */
#include "organizationalIdentifier.c" /* Agency issuing FASC-N */
#include "poa.c"

```

```

/*****

```

```

This program is a sample of pulling in each of the software programs. Note, the definitions
provided at the top along with the data structures. Note, a function to recalculate the
longitudinal redundancy character (LRC) was not performed. If needed, the fascn.c test fixture
includes a calculating function. Programmers are expected to define their own error value
with the proper programming scope. A default of 'Z' is used in this test.

```

```

*****/

```

```

main(argc, argv)
int argc;
char *argv[];
{

    read_fascn(); /* function to load in the FASC-N from a file */

    expand_fascn(padded_fascn, expanded_fascn);
    /* The personalIdentifier sets PI to the character array of the PI */
    personalIdentifier(expanded_fascn, PI);
    printf("The Personal Identifier is: %s\n",PI);
}

```

```

printf("The PI as a number is: %lld\n",PI_number(expanded_fascn));
printf("The Agency Code is: %d\n", agency_code(expanded_fascn));
printf("The System Code is: %d\n", system_code(expanded_fascn));
printf("The credential number is: %ld\n", credential_number(expanded_fascn));
printf("The credential series is: %d\n", credential_series(expanded_fascn));
printf("The individual credential issue is: %d\n", individual_credential_issue(expanded_fascn));
printf("The Organizational Category is: %d\n", organizational_category(expanded_fascn));
printf("The Organizational Identifier is: %d\n", organizational_identifier(expanded_fascn));
printf("The Person/Organization Association Category is: %d\n",
        person_organization_association(expanded_fascn));
}

```

## Agency Check

In this program, just a few functions are called to determine if the CAC/PIV card holder is VA or DoD. It also shows how the cardholder's ID number is extracted.

### Outputs

Several tests were performed to illustrate how the various departments could be determined.

#### *VA Test*

##### Results from the fascnParse program

The Personal Identifier is: 1234567890

The PI as a number is: 1234567890

The Agency Code is: 32

The System Code is: 1

The credential number is: 92446

The credential series is: 0

The individual credential issue is: 1

The Organizational Category is: 1

**The Organizational Identifier is: 3649**

The Person/Organization Association Category is: 2

##### Results from the agencyCheck program

**Agency = 3600**

**VA identified**

The ID Number is: 1234567890

Note: the Organizational Identifier (OI) was changed to 3600 and the VA identified. The original OI, 3649, corresponded to the VA's Immediate Office of Assistant Secretary for Human Resources and Administration.

#### *DoD Test*

##### Results from the fascnParse program

The Personal Identifier is: 3445566778

The PI as a number is: 3445566778

The Agency Code is: 12

The System Code is: 232

The credential number is: 133401

The credential series is: 0

The individual credential issue is: 1

The Organizational Category is: 1

## The Organizational Identifier is: 9765

The Person/Organization Association Category is: 2

Results from the agencyCheck program

**Agency = 9700**

**DoD identified**

The ID Number is: 3445566778

Note: the original OI, 9765, corresponded to the Pentagon Force Protection Agency.

## *Army Test*

Results from the fascnParse program

The Personal Identifier is: 1015566778

The PI as a number is: 1015566778

The Agency Code is: 312

The System Code is: 132

The credential number is: 133401

The credential series is: 0

The individual credential issue is: 1

The Organizational Category is: 1

**The Organizational Identifier is: 2130**

The Person/Organization Association Category is: 2

Results from the agencyCheck program

**Agency = 2100**

**Army identified**

The ID Number is: 1015566778

Note: The original OI, 2130, is for the Army National Guard Bureau

## Source Code

```
#include <stdio.h>
```

```
#define error 'Z' /* The error return value */
```

```
char padded_fascn[25]; /* This is the 25-byte FASC-N format on the CAC/PIV card */
```

```
char expanded_fascn[40]; /* The 40-byte expanded FASC-N format */
```

```
char PI[10]; /* A text return of the personal identifier. Note, programmers must check array bounds */
```

```
/******
```

```
read_fascn() will read in the binary FASC-N from a default file fascn.dat
```

```
While this is adequate for testing, a different delivery is expected in practice.
```

```
*****/
```

```
int read_fascn()
```

```
{  
    int byteCount;  
    FILE *fp;  
    if ((fp = fopen("fascn.dat", "r")) == NULL)  
        return(1);  
    for(byteCount=0; byteCount<25; byteCount++)  
        padded_fascn[byteCount] = fgetc(fp);  
    fclose(fp);  
    return(0);  
}
```

```

}

#include "toint.c"
#include "convertCharacter.c"
#include "expandFascn.c"
#include "PINumber.c" /* Number return of PI */
#include "organizationalCategory.c" /* Is NIST SP 800-87 used? */
#include "organizationalIdentifier.c" /* Agency issuing FASC-N */

/*****

This program is a sample of using six of the C functions to determine if a person
is part of the DoD or VA and then what their ID number

*****/

main(argc, argv)
int argc;
char *argv[];
{
    int agency;
    read_fascn(); /* function to load in the FASC-N from a file */
    expand_fascn(padded_fascn, expanded_fascn);
    if(organizational_category(expanded_fascn) == 1)
    {
        agency = organizational_identifier(expanded_fascn);
        agency -= agency % 100; /* truncate to department number */
        printf("Agency = %d\n", agency);
        switch(agency){
            case 1700: {
                printf("Navy identified\n");
                break; }
            case 2100: {
                printf("Army identified\n");
                break; }
            case 3600: {
                printf("VA identified\n");
                break; }
            case 5700: {
                printf("Air Force identified\n");
                break; }
            case 9700: {
                printf("DoD identified\n");
                break; }
        }
    }
    printf("The ID Number is: %lld\n",PI_number(expanded_fascn));
}

```