Send feedback for this document.

Both 2003 and 2004 versions of Digivote contain major errors that compromise the anonymity of the voting procedure.

Document History

May 26, 2003	Paul - first version
Jun 02, 2003	Paul - added randomize paragraph, routine table
Jun 06, 2003	Paul - added comment, added to randomize paragraph, added yet another bug
Jun 07, 2003	Paul - added conclusion
Jun 12, 2003	Paul - added TOC, Jites paragraph
Apr 10, 2004	Paul - downgraded non-ANSI, added sessionkey/timestamp problems
May 18, 2004	Paul - fixed typo
Jun 27, 2004	Paul - major rewrite for 2004 version
Jul 01, 2004	Paul - rebuttal
Jul 03, 2004	Paul - added software overview paragraph
Jul 05, 2004	Paul - added database schema and syntax for several B0XX files
Jul 06, 2004	Paul - updated for cleaning up voting-related globals
Jul 08, 2004	Paul - updated with B019 and B020 syntax

Conclusions:

Casual inspection of the Digivote sourcecode reveals obvious errors from which we deduce scant peer review of the code, if any, has taken place. Keeping the voting anonymous isn't high on the priorities list: stack variables are not zeroed after their useful lifetime has expired, the randomize function is misused thus that the data on the magnetic cards contain a timestamp, and the order of votes can in almost all cases be deduced from the contents of the B003 and B013 files.

0) Introduction.

If you vote electronically, you will get a magnetic card you can insert into a computer in a voting booth (running the MAV program). With an optical pen you select the party and or candidates you want to vote for and this data will be written to the card. The card is then ejected and you take it with you out of the voting booth and deposit it in another machine (running the URN program). Upon depositing, the vote is read and written to disk in encrypted form. At the end of the election, the votes are decrypted again and counted.

We will take a look at the Digivote applications. The sourcecode for both the 2003 and 2004 versions can, at the moment of writing, be downloaded from elections.fgov.be. If that fails, you can download our copy (2003) (2004). The 2003 archive is called digivote.exe, but on *NIX, just do an "unzip digivote.exe".

Design documents for Digivote are not available. We could only find the publicly available instruction manuals for election officials (available at elections.fgov.be), and two leaked evaluation documents: http://www.poureva.be/article.php3?id_article=32 and http://www.poureva.be/IMG/pdf/19990701-2.pdf.

1) Overview of the software.

The Digivote sourecode is divided in two parts: the PRG directory contains the sourcecode for the MAV and URN applications used in the voting stations, while the VOTE directory contains the sourecode for both the preparation (logistics) and totalization (counting).

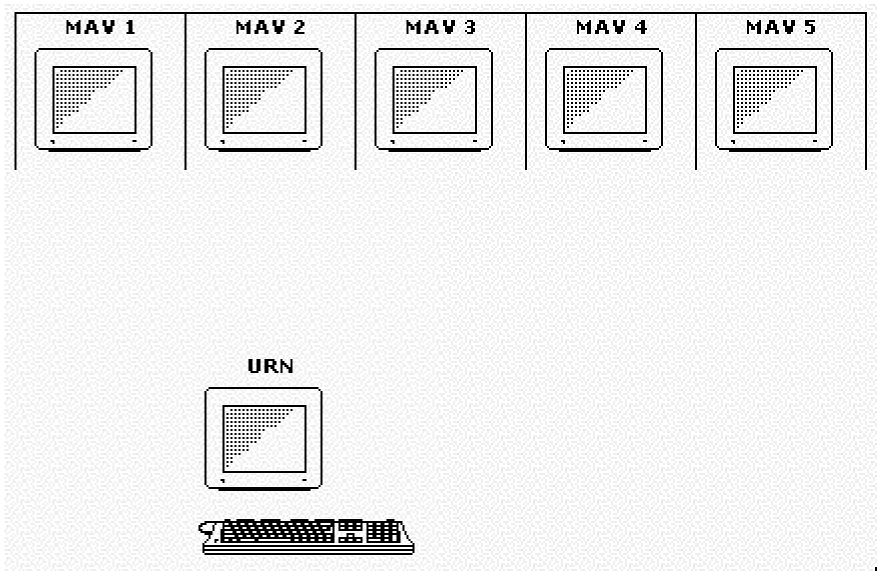

figure 1:
Layout of a typical voting station: each voting booth contains a machine running the MAV program. A single machine running the URN program is used to issue and collect magnetic cards.

All machines in the voting station are booted from the same floppy disk. Depending on the hardware attached, either the URN or MAV program will be started. Both programs require a 16-character password to run. In the case of the MAV machines, in the absence of a keyboard a magnetic card is used to transfer the password.

The sourecode for the MAV and URN applications is nominally in C++, though the bulk of the code is plain C. The code is of decent quality and fairly legible.

The software for the preparation and totalization is written in Progress. As far as we could determine, the sourecode does not include the backend database or the database schemes, so the types of field names have to be inferred from the context.

directory	contents	total lines
PRG	C sourcecode	
AES	crypto routines	2064
ARC	dos compression routines	1524
DIAGNOST	diagnostic tools	4336
DIVERS	other tools	5361
GEN	code common to MAV and URN programs	9077
MAV	MAV specific code	12163
TOOLS	debug routines	85

URN	URN specific code	3676
VOTE	Progress sourcecode	
GEN	subroutines	3950
HLC	C support code	5148
INCL	include files	700
PRP	preparation and logistics programs	11923
TOOLS	other tools	4167
TOT	totalization programs	3395

table 1:
Digivote sourcetree overview.

2) Anonymity compromised.

In both 2003 and 2004 versions, there are two major errors that compromise the anonymity of the voting procedure. Both are caused by the same mistake: an assumption that the "random" function returns a truly random number. In reality, the "random" function will return a value that is a deterministic function of the "seed" value used to seed the Pseudo Random Number Generator (PRNG). For the Borland library used in the Digivote applications, this function can be written as:

```
#define MULTIPLIER      0x015a4e35L   /* 22,695,477 */
#define INCREMENT       1
static  long     Seed = 1;

void srand(unsigned seed) {
   Seed = seed;
}

int rand(void) {
   Seed = MULTIPLIER * Seed + INCREMENT;
   return((int)(Seed >> 16) & 0x7fff);
}


#define RAND_MAX 0x7FFFU
#define randomize() srand((unsigned)time(NULL))
#define random(num)(int)(((long)rand()*(num))/(RAND_MAX+1))
```

If the PRNG is seeded with the time, a call to "random" will get you a number that is for all practical purposes a timestamp.

The MAV program uses the "random" function to create a sessionkey, used to add variance to the voting data written to the magnetic card, before a digital signature is added. The digital signature allows the URN machine to have a high degree of certainty that the vote on the magnetic card was written by one of its own MAV machines. It also prevents against subverted magnetic card readers modifying data, either on the MAV or URN machines.

The need for an extra sessionkey is not immediately clear. The only thing we could come up with was to detect "cloned" votes, i.e. cards that are bit for bit copies of a card with a valid signature, and hence have the same sessionkey. However, the URN machine does not actively scan for collision of sessionkeys.

Since the URN machine needs the sessionkey to verify the digital signature, the sessionkey is written on the magnetic card. And since the PRNG is seeded with the time for each vote, this sessionkey is for all practical purposes a timestamp.

This code will create a lookup table of sessionkeys (using random and srand as defined above):

```
int sessionkey(unsigned s)
{
	int i;
	int key = 0;
	srand(s);

	for(i=0; i < 8; i++)
		key = 10*key+random( 10);

	return key;
}



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

	long elections = 0x40CBDF50; /* june 13 2004 0600 UTC */
	int duration = 12*3600;

	for(i=0;i<duration;i++) {
		int key = sessionkey(elections+i);

		int hrs = i /3600;
		int min = (i -(hrs*3600))/60;
		int sec = (i -(hrs*3600)-(min*60));

		printf("%08i %02i:%02i:%02i\n",key,hrs+6,min,sec);


		}
	}
```

On the URN side, the votes are read, encrypted and written to disk. Again, the "random" function is used to write each vote to a supposedly arbitrary position in the B003 and B013 files. However, as here the PRNG is seeded only once, the order of the votes can easily be reconstructed in almost all cases.

Modifications made to the 2004 version include the zeroing of sensitive globals before ejecting the magnetic card from the MAV machine, but those modifications do not include routines copying this information to local variables cleaning up after themselves. As it would take some effort to find out if the stackspace used by, e.g. the routine Build_Card_Buffer for a buffer will be overwritten by stackspace used by other routines by the time the voter leaves the vooting booth, we are for the moment not certain whether all sensitive data is zeroed by the time the magnetic card is ejected and the voter leaves the voting booth.

3) Expert denial.

In a draft report, the committee of experts denies the possibility that the order of votes can be deduced from the contents of the B003 or B013 file:

- Les remarques concernant l'ordre aléatoire d'encodage des votes dans l'urne sont non pertinentes dans la mesure où la séquence aléatoire n'est reproductible que si on connaît avec certitude la seconde à laquelle l'urne-PC a démarré le programme principal, ce qui est impossible.

- De opmerkingen betreffende de willekeurige volgorde van de opslag van de stemmen in de urne zijn niet terecht in de zin dat de willekeurige volgorde slechts herhaald kan worden indien men met zekerheid de seconde kent waarop de urne-PC het hoofdprogramma heeft opgestart, hetgeen onmogelijk is.

This is incorrect, because in most cases (i.e when less than 1995 of the maximum 2000 possible votes are cast), the actual seed of the PRNG and the second of initialisation can be deduced from the pattern of "holes" left in the list of votes.

Because it is impossible to know how many voters will turn up, all votes are pseudorandomly distributed over 2000 positions (an upper bound for the number of voters in a single voting station). When all votes are cast, the positions that do not contain a vote allow us to determine the seed for the PRNG.

What we do is we assume a range of possible values for the moment the machine was booted, and hence the number the PRNG was seeded with. Assuming the URN machine must have been booted up to a few hours prior to the opening of the voting station, we only have to try on the order of 10^4 (2^13) possibilities. For a reasonable number of votes (i.e. not to close to 0 or 2000) each of these possibilities will have a unique set of "holes", and only one of these will correspond to actual distribution of votes in the B003 and B013 files.

Once the correct seed is known, the order of the votes can be computed.

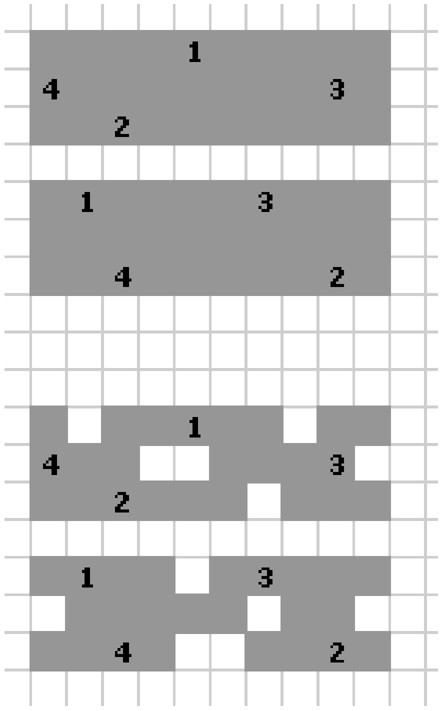

figure 2:
top half: different pseudo-random distributions of n elements over n positions offer no information to the distribution used.
bottom half: different pseudo-random distributions of less than n elements over n positions create different patterns, that can be used to determine which distribution was used.

references:
McGraw, Viega: Make your software behave: Playing the numbers
www-106.ibm.com/developerworks/library/s-playing

Appendix 1: Changes to the 2004 version.

Comparison of the 2003 with the 2004 versions has been complicated by the removal of comments from most of the sourcefiles. Non-trivial changes include:

- Major changes to the lightpen driver.

- Support for higher resolution graphics driver.

- Support for compression and authentication of data files.

- A more complex counting procedure that would allow the detection of some RAM soft-errors.

- Interface changes including screen layout and voting procedure.

- Better error messages in some programs.

- MAV Globals containing sensitive data are now zeroed before ejecting the magnetic card.

- A bug in the 2003 version first identfied by us was fixed in the 2004 without attribution:

In the Verify_Password routine in gencryp.cpp :

```
    memcpy( sKey.data, &record[MAX_IV_SIZE], MAX_IV_SIZE);
```

was changed to

```
    memcpy( sKey.data, &record[MAX_IV_SIZE], AESKEYLEN);
```

Appendix 2: Overview of selected MAV routines.

routine name	functionality	remarks
`for(;;)`	main loop	
` Format_Blanco_Card`		
` Election_Loop`		
`  Init_Card_Buffer`		
`  Process_Election*`		
`   Process_College*`		
`    Process_Party*`		
`     Init_Candidates`		
`     Display_Candidates`		
`      Input_Candidates`		
`       Is_Head_Vote_Selected*`		
`       Are_Candidates_Selected*`		
`  Buffer_Vote`	convert to proto card format	
`   Init_Vote_`		
`   Format_Vote_`		

Append_To_Card		
initCard		
Generate_Mav_Session	create session key "mavSessionKey"	keyspace=10^8
randomize		
Build_Card_Buffer	serialize for magnetic card format	
Convert_Vote_Buffer_To_Card_Buffer		
MAC	calculate hash	
Calculate_Internal_MAVVOT		
computeMac		
Write_Votes		
Reader_Write_Card		
printTicket		
Reader_Eject_Card		
Reader_Wait_Card_Removed		

table 2:
Call tree of selected MAV routines.

Appendix 3: Layout of the data on the magnetic card.

Layout of the magnetic card data used for the Digivote automatic voting system used for the 2003 elections.

The magnetic card used for the Digivote automatic voting system is listed as ISO 7810/7811-2 compliant. [1]
From the definition of the constant MAX_CARD_BUF as 104 [2], we can reasonable infer track 3 (ISO 4909) is used, allowing up to 104 5-bit characters.

The actual layout is as follows [3],[4]:

- 8 chars session key.

This sessionkey seriously compromises the anonymity of the voting process, as the PRNG is seeded with the time for each sessionkey generated.

- 1 char vote flag.

0 = initial, 1 = voted, 2-9 = canceled.

- A series of voting data for each election. These are either 8 or 34 characters each, depending on the type of elections.
The total size can be up to 80 characters.

The first four characters are the same for each type of elections:

  - 1 char election id
  - 2 chars party id (0 for blank)
  - 1 char vote type (0 party vote, 1 effective only, 2 replacement only, 3 effective+replacement)

the remainder, for type 0 elections:

  - 2 char effective
  - 2 char replacement

for type 1 elections:

  - 30 chars for up to 90 bits, one for each candidate.

- 1 char voter type

0 = Belgian, 1 = EU.

- 0 to 7 bytes padding with zeroes to the next multiple of 8

- 8 chars signature / MAC

the remainder of the data is padded with zeroes.

[1] College van deskundigen belast met de controle van de geautomatiseerde stemmingen en stemopneming - verslag betreffende de verkiezingen van 13 juni 1999.
[2] Digivote sourcecode gen/gencard.h, line 25.
[3] Digivote sourcecode rev 9.12 gen/gentype.h, lines 207-267.
[4] Digivote sourcecode rev 9.13 gen/gendata.cpp, lines 907-948.

Appendix 4: Partial reconstruction of the backend database scheme.

election

```
 s-id           Date
 coll-id
 coll-name
 colls
 e-type         Int     0: 1 vote 1: Multiple votes
 el-mode        Int     Eligible voters as a flags 1:Bel. resident 2:Bel.
non-resident 4:Non-Bel. EU nat.
 et-id
 e-id
 e-pr           Bool
 head0[]
 head1[]
 head2[]
 head3[]
 short-name[]   Char[]
 long-name[]    Char[]
 mandatory      Bool
 org-type
 separate_TOT
 supps          Bool            True if the election has both effective
and replacement candidates
 maxcan[]       Int[3]          constraints on number of candidates by
language group
 maxsup[]       Int[3]
 minsup[]       Int[3]

party
 p-id
 e-id
 s-id           Date
 coll-id
 party-name
 taalgroep[]
 num-can        Int             # of candidate detail records with c-
type==0
 num-sup        Int             # of candidate detail records with c-
type==1
 logo[]
 logo-height    Int
 logo-width     Int
 logo-bytes     Int
 vote_top       Char[32]        Encrypted
 vote_can       Char[32]        Encrypted
 vote_sup       Char[32]        Encrypted
 vote_cs        Char[32]        Encrypted

urnedest
 urne-id
 urne-area
 org-type
 sys-type
 total-cards    Int
 annul-card     Int
 unused-card    Int
 count-card[]   Int[16]
 elects[]
 out-status     Int
 master-key     Char[16]
 session-key
 adres          Char[30]
 name           Char[30]
 postcode       Char[4]
 tel-nr         Char[20]
 lokal          Char[25]
 data-read      Int             0:Not read 1: Read from Master 2: Read
from Backup
 time-rcv       Time
 disk-type

lstdest
 lst-id
 lst-area
 org-type
 sys-type
 master-key
 session-key
 out-status
 adres
 name
 postcode
 tel-nr
 lokal

setup                           defines capabilities of station running
the software
 setup-id       Int
 org-type       Char[]
 sys-type       Char[]          "M", "I", "G" or "T"
 area
 areaname       Char[]
 orginator      Int
 startup[]      Char[3][]       Session, Election and Organisation info
 lang-pc        Int             range 0..4 : D, F, G, D/F, F/G
 lang-bur[]     Bool[3]         languages used D/F/G
 paswdprp       Char[32]        password for logistics activities
 paswdtot       Char[32]        password for totalization activities
 master-key     Char[]
 session-key    Char[]

session
 s-id           Date
 upmas
 upses
 print-done     Bool
 prpmade        Int
 prpread        Int
 urnmade        Int
 urnread        Int
 lang           Int             range 0..4 : D, F, G, D/F, F/G
 lang2          Int             as lang
 prt-mav        Int
 tel-nr         Char[]
 hands-off      Bool
 nbr_flop       Int

paswdest
 pasw-id
 pasw-area
 org-type
 master-key
 session-key
 sys-type
 adres
 name
 postcode
 tel-nr
 lokal
 data-read
 out-status

candidate
 p-id
 e-id
 c-id
 s-id           Date
 coll-id
 c-type         Int             0: effective, 1: replacement
 c-name1        Char[]
 c-name2        Char[]
 vote

language        localization strings
 var_lang
 var_name       Char[]          variable name
 var_string[]   Char[3][]       localized string
 fra_name       Char[]          frame name
 len_string

organisation
 setup-id
 Ingave[]
 heading1[]
 heading2[]
 heading3[]
 org-type
 area[]

layout
```

```
 doc-id         Int
 page-nbr       Int
 sect-nbr       Int
 line-nbr       Int
 contents       Char[]  Substrings of the form @Fnn indicate fields to be
inserted

selection
 c-id           Int
 c-type         Int
 e-id           Int
 lang-pc        Int
 p-id
 party-name
 s-id           Date
 verk_name      Char[]

types           template records used to create election records
 coll-id
 coll-name      Char[3][]       College name by language
 colls
 e-type         Int             0: 1 vote 1: multiple votes
 el-mode        Int             Eligible voters as flags 1:Bel. resident
2:Bel. non-resident 4:Non-Bel. EU nat.
 et-id
 head0[]
 head1[]
 head2[]
 head3[]
 short-name[]   Char[3][]       Short name by language
 long-name[]    Char[3][]       Full name by language
 mandatory
 org-type
 separate_TOT
 supps
 maxcan[]       Int             Constraint on number of candidates by
language group
 maxsup[]       Int
 minsup[]       Int
 used

names
 area[]


session-exceptions
 setup-id       date
 lang           Int
 heading3[]
 prt-mav        Int

usage
 doc-id
 doc-lang
 doc-type       Int     1: PV 2:  3: receipt PRP 4: receipt TOT 5: 6: 7:
9: 81: 82 83:
 et-combi
 et-ids
 setup-id       Int

urnechku                List of files on the U-disk requiring a checksum
file
 boot-file      Bool    If true: File in root directory
 copy-file      Bool    If true: file in \VOTE\FILES\BAT\ if false: file
in \VOTE\FILES\INCL\
 src-name       Char[]  filename
 dest-name      Char[]  name of checksum file to be placed in
\VOTE\FILES\CHECK\

sysinfo
 userName
 licenseNo
 serialNo
 offSet

panache         workfile
 id             Int
 type           Int
 maxpar         Int

verkiezing      workfile
 e-id
 e-pr           Bool

wf              workfile
 w-prg
 w-db
 w-ldb

wfrow           workfile
 nr
 txt[]

all_election    buffer for election

setup2          buffer for setup

f2              generic buffer


_file           database metadata tables
_field
_index
_index-field
```

Appendix 5: Syntax of the B019 file.

The B019 file is encrypted, the unencrypted form of the B019 file is called INFLIS.

# indicate our meta-comments, original comments start with //
\ are used to split long lines

all P,L and C records are space delimited.

related files:
PRP/CREATLIS.P
PRP/CRINFLIS.P
PRP/IMPB019K.P
PRP/IMPORTPR.P

```
Paswoord_check
<lstdest.sys-type><lstdest.org-type><lstdest.lst-area><lstdest.lst-id>
<types.et-id>
<election.e-id>
P <party.s-id> <party.e-id> <party.coll-id> <party.p-id>
<party.party_name> \
 <party.taalgroep[1..3]>
L <party.s-id> <party.e-id> <party.coll-id> <party.p-id> 0\
 <party.logo_width> <party.logo_height> <party.logo_bytes>
#for each byte in the logo:
L <party.s-id> <party.e-id> <party.coll-id> <party.p-id>
<1..party.logo_bytes> \
 <logobyte>
C <candidate.s-id> <candidate.e-id> <candidate.coll-id> <candidate.p-id>
\
 <candidate.c-type> <candidate.c-id> <candidate.c_name1>
<candidate.c_name2>
```

Appendix 6: Syntax of the B020 file.

The B020 file is encrypted, the unencrypted form of the B020 file is called INFPAS.

# indicate our meta-comments, original comments start with //
\ are used to split long lines

all D records are space delimited.

related files:
PRP/CREATPAS.P
PRP/CRINFPAS.P
PRP/IMPB020D.P
PRP/IMPORTPR.P

```
Paswoord_check
<paswdest.sys-type><paswdest.org-type><paswdest.pasw-
```

```
area><paswdest.pasw-id>
D <urnedest.sys-type> <urnedest.org-type> <urnedest.urne-area>
<urnedest.urne-id> \
 <urnedest.name> <urnedest.adres> <urnedest.postcode> <urnedest.lokal> \
 <urnedest.tel-nr> <urnedest.master-key> <urnedest.session-key>
```

Appendix 7: Syntax of the B021 file.

The B021 file contains the election-related data for the MAV application.
The B021 file is decompressed and read from the same floppy used to boot the MAV machine.

The A record has one line for each field, identified by a tag.
B,C,D and E records have one line for each record.

# indicate our meta-comments, original comments start with //
\ are used to split long lines

All records are tab-separated.

related files:
PRP/CPURN.P
PRP/CREATURN.P
PRP/CRINFURN.P
TOOLS/BHURNCHK.P

```
// A Verkiezings gegevens : Gebruikt voor configuratie.
A       1       "<urnename>" #BureauName
A       2       "<orgname>"
A       7       <session.tel-nr>
A       8       <wz-nbr_flop>
A       9       <session.s-id> #Electiondate
A       10      <wz-prt-mav> #ticketing
//__________________________________________________________
//ELECTION DATA
//__________________________________________________________
B       <election.et-id>        <election.e-type>       \
      <election.long-name>    <election.short-name>   \
#       <election.long-name>    <election.short-name> (if bilingual D/F
or F/G)
//__________________________________________________________
//COLLEGE DATA
//__________________________________________________________
C       <election.et-id>        <election.coll-id>
<election.coll_name>
//__________________________________________________________
//PARTY DATA
//__________________________________________________________
D       <election.et-id>        <election.coll-id>      \
      <party.p-id>            <party.party_name>      \
      <party.logo_width>      <party.logo_height>     \
      <languagegroup>
```

```
//____________________________________________________________
//CANDIDATE DATA
//____________________________________________________________
E       <election.et-id>        <election.coll-id>      \
       <party.p-id>            <candidate.c-type>      <candidate.c-id> \
       <candidate.c_name1>     <candidate.c_name2>
```

Appendix 8: Syntax of the B001 file.

The B001 contain the election results for each URN machine, or are the result of merging several individual B001 files. Only the B001 files produced by the URN application contain B records.

Each line of the B001 file is individually encrypted, the unencrypted form of the B001 file is called ELECT.LST.

The A record has one line for each field, identified by a tag.
B,D and E records have one line for each record.

# indicate our meta-comments, original comments start with //
\ are used to split long lines

related files:
URN/URNRECO.CPP
GEN/GENDATA.CPP
TOT/CREATTOT.P
TOT/TOTURNE.P
TOT/CRTOTINF.P

```
A       0       <cards> <count[1]> ... <count[16]>
A       1       <urnename>
# optional: B records as in B021
D       <election.et-id>        <election.coll-id>      <party.p-
id>    \
       <party.vote_top>        <party.vote_can>        \
       <party.vote_sup>        <party.vote_cs>
E       <election.et-id>        <election.coll-id>      <party.p-
id>    \
       <candidate.c-type>      <candidate.c-id>        <vote_pers>
```

Appendix 9: Syntax of the B011 file.

The B011 file contains the election results for a single kanton.
It is computed from either individual B001 files produced by the URN application, or intermediate aggregate B001 files.

The A record has one line for each field, identified by a tag.
B,C,D,E,F and G records have one line for each record.

# indicate our meta-comments, original comments start with //

\ are used to split long lines

related files:
TOT/EXPZET.P
TOT/RD_RESUL.P

```
//-----------------------
// A : CONFIGURATION DATA
//-----------------------
A       1       <setup.orginator>
A       2
A       3
A       4
A       5
A       6
A       7
A       8
A       10
A       11      <setup.areaname>
A       12      <session.lang2>
A       13      <session.s-id>
//-----------------------
// B : ELECTION DATA
//-----------------------
B       <election.et-id>        <e-type>        \
       <election.long-name>    <election.short-name> \
#       <election.long-name>    <election.short-name> (if bilingual D/F
or F/G)
//-----------------------
// C : COLLEGE DATA
//-----------------------
C       <election.et-id>        <election.coll-id>
<election.coll_name>
//-----------------------
// D : PARTY DATA
//-----------------------
D       <election.et-id>        <election.coll-id>      \
       <party.p-id>            <party.party_name>
//-----------------------
// E : CANDIDATE DATA
//-----------------------
E       <election.et-id>        <election.coll-id>      \
       <party.p-id>            <candidate.c-type>      <candidate.c-id>
\
       <candidate.c_name1>     <candidate.c_name2>
//-----------------------
// F : PARTY RESULTS
//-----------------------
F       <election.et-id>        <election.coll-id>      <party.p-
id>    \
       <vote_top>              <vote_can>      \
       <vote_sup>              <vote_cs>
```

```
//------------------------
// G : CANDIDATE RESULTS
//------------------------
G        <election.et-id>        <election.coll-id>       \
       <party.p-id>            <candidate.c-type>       \
       <candidate.c-id>        <vote_pers>
```