```
(************************************************************************
  *
  *      Group Communication Service, VERSION 1.13
  *
  *      Daniel Amyot and Jacques Sincennes
  *      University of Ottawa, August 15, 1996
  *
  *      Purpose: Multicast messages to the members of a group.
  *      =======
  *
  *      Channels:
  *      ========
  *      mgcs_ch:  Master Group Communication Server Channel
  *      gcs_ch:   Group Communication Server Channels
  *      out_ch:   mean of distributing messages to group members
  *
  *      Groups:
  *      ======
  *      - administered: admin alone creates and destroys group
  *
  *      - moderated:    moderator is the only one allowed to multicast.
  *                      All other messages are sent to him for approval.
  *
  *      - public    anyone can register
  *                  e.g. mailing lists
  *      - private   admin must register another one. Must be member to see list
  *                  of group members.
  *                  e.g. telephone conference
  *
  *      - opened    anyone can send
  *                  e.g. mailing lists
  *      - closed    must be member to send
  *                  e.g. IRC
  *
  ************************************************************************)

specification Group_Communication_Service[mgcs_ch, gcs_ch, out_ch]:noexit

library
    Boolean, NaturalNumber, HexDigit
endlib

(*============================================*)
(*            IS8807 ADT definitions           *)
(*============================================*)

(* Types FBoolean, Element, and Set contain corrections *)
(* to the library from the International Standard 8870  *)

type        FBoolean is
formalsorts FBool
formalopns  true        : -> FBool
            not         : FBool -> FBool
formaleqns
    forall  x : FBool
    ofsort  FBool
    not(not(x)) = x;
endtype     (* FBoolean *)
```

```
(*******************************************)

type        Element is FBoolean
formalsorts Element
formalopns  _eq_, _ne_          : Element, Element -> FBool
formaleqns
        forall  x, y, z : Element
        ofsort  Element
            x eq y = true =>
        x       = y             ;

        ofsort  FBool
            x = y =>
        x eq y  = true          ;
            x eq y =true , y eq z = true =>
        x eq z  = true          ;

        x ne y  = not(x eq y)   ;
endtype     (* Element *)

(*******************************************)

type    Set is Element, Boolean, NaturalNumber
sorts   Set
opns    {}                                  :               -> Set
        Insert, Remove                      : Element, Set  -> Set
        _IsIn_, _NotIn_                     : Element, Set  -> Bool
        _Union_, _Ints_, _Minus_            : Set, Set      -> Set
        _eq_, _ne_, _Includes_, _IsSubsetOf_    : Set, Set      -> Bool
        Card                                : Set           -> Nat

eqns    forall  x, y    : Element,
                s, t    : Set
        ofsort  Set

            x IsIn Insert(y,s) =>
        Insert(x, Insert(y,s))  = Insert(y,s)           ;
        Remove(x, {})           = {}                    ;
        Remove(x, Insert(x,s))  = s                     ;
            x ne y = true of FBool =>
        Remove(x, Insert(y,s))  = Insert(y, Remove(x,s));

        {} Union s              = s                     ;
        Insert(x,s) Union t     = Insert(x,s Union t)   ;

        {} Ints s               = {}                    ;
            x IsIn t =>
        Insert(x,s) Ints t      = Insert(x,s Ints t)    ;
            x NotIn t =>
        Insert(x,s) Ints t      = s Ints t              ;

        s Minus {}              = s                     ;
        s Minus Insert(x, t)    = Remove(x,s) Minus t   ;

        ofsort  Bool

        x IsIn {}               = false                 ;
            x eq y = true of FBool =>
        x IsIn Insert(y,s)      = true                  ;
```

```
            x ne y = true of FBool =>
        x IsIn Insert(y,s)       = x IsIn s              ;
        x NotIn s                = not(x IsIn s)         ;

        s Includes {}            = true                  ;
        s Includes Insert(x,t)   = (x IsIn s) and (s Includes t)    ;

        s IsSubsetOf t           = t Includes s          ;

        s eq t                   = (s Includes t) and (t Includes s);

        s ne t                   = not(s eq t)           ;

        ofsort  Nat

        Card({})                 = 0                     ;
            x NotIn s =>
        Card(Insert(x,s))        = Succ(Card(s))         ;
endtype (* Set *)


(*=============================================*)
(*             GCS ADT definitions             *)
(*=============================================*)

(* A group member is identified by a hexadecimal digit. *)

(* Use hexadecimal digits as Member Identifiers *)
type        MIDType0 is HexDigit renamedby
sortnames
            MID for HexDigit
endtype     (* MIDType0 *)

type    MIDType is MIDType0
opns
        (* Special MID reserved for admin/moder modification *)
        Nobody  : -> MID
eqns
        ofsort Nat
        (* For comparisons with other MIDs *)
        NatNum(NoBody) = Succ(NatNum(F of MID));
endtype (* MIDType *)

(*********************************************)

(* List of MIDs.  Used to answer "Members" requests *)
(* Implemented as a set. *)
(* We avoid the problem with ISLA's renaming in actualization *)
type            MIDListType0 is Set
actualizedby    MIDType using
sortnames
            MID for Element
            Bool  for FBool
endtype     (* MIDListType0 *)


type        MIDListType is MIDListType0 renamedby
sortnames
            MIDList for Set
```

```
opnames
            Empty   for {}      (* Empty list of members *)
endtype     (* MIDListType *)

(*********************************************)

(* We use hexadecimal digits as Channel Identifiers (within a channel type) *)
(* Used to describe the specifics of the multicasting type (requestor's *)
(* host, IP, socket) according to group requirements *)
type        CIDType is HexDigit renamedby
sortnames
            CID for HexDigit
endtype     (* CIDType *)

(*********************************************)

(* Set of pairs (member, channelID), to be registered in a group *)
type    MemberType is MIDType, CIDType
sorts   MBR
opns
        _._         : MID, CID      -> MBR
        MID         : MBR           -> MID
        CID         : MBR           -> CID
        _eq_, _ne_  : MBR, MBR      -> Bool
eqns
        forall  mid1, mid2: MID,
                cid1, cid2: CID,
                mbr1, mbr2          : MBR

        ofsort MID
        MID(mid1.cid1)              = mid1;

        ofsort CID
        CID(mid1.cid1)              = cid1;

        ofsort Bool
        (mid1.cid1) eq (mid2.cid2)  = (mid1 eq mid2) and (cid1 eq cid2);

        mbr1 ne mbr2                = not (mbr1 eq mbr2);
endtype (* MemberType *)

(*********************************************)

(* Type for a list of member-channel pairs (implemented as a set) *)
(* We avoid the problem with ISLA's renaming in actualization *)
type            MemberListType0 is Set
actualizedby    MemberType using
sortnames
            MBR for Element
            Bool  for FBool
endtype     (* MemberType0 *)


type        MemberListType1 is MemberListType0 renamedby
sortnames
            MemberList for Set
opnames
            NoMBR   for {}      (* Empty list of members *)
endtype     (* MemberListType1 *)
```

```
(* Additional operations needed to act like a list *)
type    MemberListType is MemberListType1, MIDListType
opns
        ErrorNoTop      :                   -> MBR
        ErrorNoTail     :                   -> MemberList
        Top             : MemberList        -> MBR
        Tail            : MemberList        -> MemberList

        (* The following functions act on the MID only. *)
        _IsIn_          : MID, MemberList   -> Bool
        _NotIn_         : MID, MemberList   -> Bool
        RemoveMBR       : MID, MemberList   -> MemberList
        MembersOnly     : MemberList        -> MIDList
eqns
        forall  member  : MBR,
                m       : MemberList,
                id1, id2: Mid,
                chnl    : CID

        ofsort MBR
        Top (NoMBR)                 = ErrorNoTop; (* Should not happen *)
        Top (Insert (member, m))    = member;

        ofsort MemberList
        Tail (NoMBR)                = ErrorNoTail; (* Should not happen *)
        Tail (Insert (member, m))   = m;

        RemoveMBR (id1, NoMBR)                  = NoMBR;
        RemoveMBR (id1, Insert(id1.chnl, m))    = m;
            id1 ne id2 = (true of Bool) =>
        RemoveMBR (id1, Insert(id2.chnl, m))    = Insert(id2.chnl,
                                                         RemoveMBR(id1, m));

        ofsort Bool
        id1 IsIn m                          = id1 IsIn MembersOnly(m);
        id1 NotIn m                         = Not(id1 IsIn m);

        ofsort MIDList
        MembersOnly(NoMBR)                  = Empty;
        MembersOnly(Insert(id1.chnl, m))    = Insert(id1, MembersOnly(m));

endtype (* MemberListType *)

(*********************************************)

(* Use hexadecimal digits as GCS IDentifiers *)
type        GIDType is HexDigit renamedby
sortnames
            GID for HexDigit
endtype     (* GIDType *)

(*********************************************)

(* List of channel multicasting types available *)
type    ChanType is
sorts   Chan
opns
```

```
        Mail,
        Socket,
        Text,
        Audio,
        Video       :   -> Chan
endtype (* ChanType *)

(*****************************************************
  * A group contains characteristics and a list of members.
  * Characteristics are tuples (records):
  *                 (groupID,           (of GID)
  *                  rationale,         (not shown here)
  *                  channelType,       (of Chan)
  *                  administeredBool,  (of Bool)
  *                  adminID,           (of MID)
  *                  openedBool,        (of Bool)
  *                  privateBool,       (of Bool)
  *                  moderatedBool,     (of Bool)
  *                  moderID            (of MID) )
  *
  * Used in messages and in GCS databases.
  *****************************************************)

type    GCSinfoRecordType is GIDType, MemberListType, ChanType
sorts   GSCinfo
opns
        Info        : GID, Chan, Bool, MID,
                      Bool, Bool, Bool, MID             -> GSCinfo
        _eq_, _ne_  : GSCinfo, GSCinfo                  -> Bool
eqns
        forall  gcs1, gcs2  : GSCinfo

        ofsort Bool
        gcs1 eq gcs2        = true;   (* Needed by ISLA? *)
        gcs1 ne gcs2        = not(gcs1 eq gcs2);
endtype (* GCSinfoRecordType *)

(*******************************************)

(* List of Groups for Master GCS, implemented as a set. *)
(* We avoid the problem with ISLA's renaming in actualization *)
type            GroupList0 is Set
actualizedby    GIDType using
sortnames
            GID for Element
            Bool  for FBool
endtype     (* GroupList0 *)


type        GroupList is GroupList0 renamedby
sortnames
            GroupList for Set
opnnames
            NoGCS     for {}    (* Empty list of GCS *)
endtype     (* GroupList *)

(*******************************************)

(* Request messages to be sent to the server *)
```

```
type        RequestType is HexDigit renamedby
sortnames   Request for HexDigit
opnames
            CREATEGROUP     for 1
            DELETEGROUP     for 2
            REGISTER        for 3
            DEREGISTER      for 4
            MEMBERS         for 5
            GROUPS          for 6
            MULTICAST       for 7
            CHANGEADMIN     for 8
            CHANGEMODER     for 9
endtype     (* RequestType *)


(* Resulting acknowledgement messages (OK and Error) from the server *)
type        AckType0 is HexDigit renamedby
sortnames   Ack for HexDigit
opnames
            (* OK *)
            GROUPCREATED        for 1
            GROUPDELETED        for 2
            REGISTERED          for 3
            DEREGISTERED        for 4
            MESSAGESENT         for 5
            ADMINMODERCHANGED   for 6  (* Admin or Moderator was changed successfully *
)
            SENTTOMODERATOR     for 7
            GROUPWASDELETED     for 8  (* Multicast when a group is deleted *)

            (* Errors *)
            GROUPEXISTS         for 9
            GROUPDOESNOTEXIST   for A
            MEMBERNOTINGROUP    for B
            NOTADMINMODER       for C
            NOADMINGROUP        for D
            NOMODERGROUP        for E
            UNKNOWNREQUEST      for F
endtype     (* AckType0 *)

(* Additional operation to encode lists of members. *)
type    AckType is AckType0, MIDListType
opns
        MembersAre  : MIDList   -> Ack
eqns
        forall  m: MIDList
        ofsort Nat
        (* for comparison with other Acks *)
        NatNum(MembersAre(m))  = succ(NatNum(UNKNOWNREQUEST));
endtype (* AckType *)


(* Instances of messages that can be multicast (for readability). *)
(* A message should be, however, a bit string. *)
type    InfoMsgType is Boolean
sorts   InfoMsg
opns
        Hi,
        Hello,
```

```
        Salut,
        GoodBye,
        Packet,
        GroupIsDeleted      :   -> InfoMsg
endtype (* InfoMsgType *)

(*******************************************)

(* Encoding/Decoding of messages *)
(* Several types of packets are needed. *)
(* Supplemental packet types are provided for simpler execution *)
type    MsgType is GCSinfoRecordType, RequestType, AckType, InfoMsgType
sorts   Msg
opns
        (************************************************************
          * No Message Packet.  Used for:
          *      Group deletion
          *      List members
          *      List groups
          *      Deregister
          ************************************************************)
        NoMsg   : -> Msg

        (************************************************************
          * General Packet for Group Creation:
          *       Channel Type    (of Chan)
          *       ChanID          (od CID)
          *       Administered?   (of Bool)
          *       Administrator   (of MID)
          *       Opened?         (of Bool)
          *       Private?        (of Bool)
          *       Moderated?      (of Bool)
          *       Moderator       (of MID)
          *************************************************************)
        Encode      : Chan, CID, Bool, MID, Bool, Bool, Bool, MID   -> Msg
        (* Extraction Operations *)
        ChanType    : Msg                                           -> Chan
        ChanID      : Msg                                           -> CID
        IsAdmin     : Msg                                           -> Bool
        Admin       : Msg                                           -> MID
        IsOpened    : Msg                                           -> Bool
        IsPrivate   : Msg                                           -> Bool
        IsModerated : Msg                                           -> Bool
        Moderator   : Msg                                           -> MID

       (************************************************************
          * Packet for Group Administrator modification:
          *      Administrator   (of MID)
          *      Private?        (of Bool)
          *
          * Packet for Group Moderator modification:
          *      Moderator   (of MID)
          *      Moderated?  (of Bool)

         ************************************************************)
        Encode      : MID, Bool     -> Msg
        (* Modification Operations *)
        SetAdmin    : MID, Bool, Msg-> Msg
        SetModer    : MID, Bool, Msg-> Msg
```

```
(**************************************************************
  * Packet for moderator approval
  **************************************************************)
ToApprove   : Msg   -> Ack

(**************************************************************
  * Packets for multicasting
  **************************************************************)
Encode  : InfoMsg   -> Msg
Encode  : Ack       -> Msg
(* Modification Operations *)
GetAck  : Msg       -> Ack
GetInfo : Msg       -> InfoMsg

(**************************************************************
  * Packet for subscription
  *      ChanID  (of CID)
  **************************************************************)
Encode  : CID       -> Msg

(**************************************************************
  * Packet for subscription by Administrator (for private groups)
  *      NewMember   (of MID)
  *      ChanID      (of CID)
  **************************************************************)
Encode      : MID, CID  -> Msg
NewMember   : Msg       -> MID
```

eqns

```
forall  msg                 : Msg,
        info                : InfoMsg,
        CT                  : Chan,
        ChID                : CID,
        Adm, NewAdm, Mem,
        Mod, NewMod         : MID,
        AB, OB, PB, NewPB,
        MB, NewMB           : Bool,
        ack                 : Ack

ofsort Chan
ChanType(Encode(CT, ChID, AB, Adm, OB, PB, MB, Mod))    = CT;

ofsort CID
ChanID(Encode(CT, ChID, AB, Adm, OB, PB, MB, Mod))      = ChID;
ChanID(Encode(ChID))                                    = ChID;
ChanID(Encode(Mem, ChID))                               = ChID;

ofsort Bool
IsAdmin(Encode(CT, ChID, AB, Adm, OB, PB, MB, Mod))     = AB;
IsOpened(Encode(CT, ChID, AB, Adm, OB, PB, MB, Mod))    = OB;
IsPrivate(Encode(CT, ChID, AB, Adm, OB, PB, MB, Mod))   = PB;
IsPrivate(Encode(Adm, PB))                              = PB;
IsModerated(Encode(CT, ChID, AB, Adm, OB, PB, MB, Mod)) = MB;
IsModerated(Encode(Mod, MB))                            = MB;

ofsort MID
Admin(Encode(CT, ChID, AB, Adm, OB, PB, MB, Mod))       = Adm;
Admin(Encode(Adm, PB))                                  = Adm;
```

```
        Moderator(Encode(CT, ChID, AB, Adm, OB, PB, MB, Mod))    = Mod;
        Moderator(Encode(Mod, MB))                               = Mod;
        NewMember(Encode(Mem, ChID))                             = Mem;

        ofsort InfoMsg
        GetInfo(Encode(info))                                    = info;

        ofsort Ack
        GetAck(Encode(ack))                                      = ack;

        ofsort Msg
        (* If the group becomes unadministred (the new admin is Nobody), *)
        (* then it must be public. *)
            NewAdm eq Nobody =>
        SetAdmin(NewAdm, NewPB, Encode(CT, ChID, AB, Adm, OB, PB, MB, Mod)) =
                        Encode(CT, ChID, false, NewAdm, OB, false, MB, Mod);

            NewAdm ne Nobody =>
        SetAdmin(NewAdm, NewPB, Encode(CT, ChID, AB, Adm, OB, PB, MB, Mod)) =
                        Encode(CT, ChID, AB, NewAdm, OB, NewPB, MB, Mod);

        (* If the new moderator is Nobody, the group becomes unmoderated *)
            NewMod eq NoBody =>
        SetModer(NewMod, NewMB, Encode(CT, ChID, AB, Adm, OB, PB, MB, Mod)) =
                        Encode(CT, ChID, AB, Adm, OB, PB, false, NewMod);
            NewMod ne NoBody =>
        SetModer(NewMod, NewMB, Encode(CT, ChID, AB, Adm, OB, PB, MB, Mod)) =
                        Encode(CT, ChID, AB, Adm, OB, PB, NewMB, NewMod);

        ofsort Nat
        (* for comparison with other Acks *)
        NatNum(ToApprove(msg))  = succ(succ(NatNum(UNKNOWNREQUEST)));

endtype (* MsgType *)

(*********************************************)

(*********************************************
  *
  * Buffering of requests and acknowledgements
  *
  * A generic type is created and actualized as
  * FIFO buffers for reqs and acks.
  *
  *********************************************)

(* Generic FIFO type definition *)
type    FIFOType is Boolean
formalsorts
        Element

sorts   FIFO
opns
        Nothing :                -> FIFO
        Put     : Element, FIFO -> FIFO
        Get     : FIFO           -> Element
        Consume : FIFO           -> FIFO
        ErrorFIFO:               -> Element
        IsEmpty : FIFO           -> Bool
```

```
eqns
        forall f:FIFO, e, e2 :Element

        ofsort FIFO
        Consume(Nothing)             = Nothing;
        Consume(Put(e, Nothing))     = Nothing;
        Consume(Put(e2,Put(e,f)))    = Put(e2, Consume(Put(e, f)));
        ofsort Element
        Get(Nothing)                 = ErrorFIFO;
        Get(Put(e, Nothing))         = e;
        Get(Put(e2, Put(e, f)))      = Get(Put(e, f));

        ofsort Bool
        IsEmpty(Nothing)             = true;
        IsEmpty(Put(e,f))            = false;
endtype (* FIFO *)


(* Encoding of requests and acknowledgements as Records *)
type    BufferEncodingType is MIDType, AckType, RequestType, MsgType
sorts   ReqRecord, AckRecord
opns
        AckElem     : MID, Ack           -> AckRecord
        ReqElem     : MID, Request, Msg -> ReqRecord

        S           : AckRecord          -> MID      (* Extract Sender *)
        A           : AckRecord          -> Ack      (* Extract Ack *)
        S           : ReqRecord          -> MID      (* Extract Sender *)
        R           : ReqRecord          -> Request  (* Extract Request *)
        M           : ReqRecord          -> Msg      (* Extract Message *)
eqns
        forall S1:MID, A1:Ack, R1:Request, M1:Msg

        ofsort MID
        S(AckElem(S1, A1))      = S1;
        S(ReqElem(S1, R1, M1))  = S1;

        ofsort Ack
        A(AckElem(S1, A1))      = A1;

        ofsort Request
        R(ReqElem(S1, R1, M1))  = R1;

        ofsort Msg
        M(ReqElem(S1, R1, M1))  = M1;

endtype (* BufferEncodingType *)


(* Actualization (and renaming) of a Buffer type for records of requests *)
type            FIFOreqsType0 is FIFOType
actualizedby    BufferEncodingType using
sortnames
                ReqRecord for Element
endtype          (* FIFOreqsType *)

type        FIFOreqsType is FIFOreqsType0 renamedby
sortnames
            FIFOreqs for FIFO
```

```
opnames
            NoReq    for Nothing
endtype         (* FIFOreqsType0 *)


(* Actualization (and renaming) of a Buffer type for records of acks *)
type            FIFOacksType0 is FIFOType
actualizedby    BufferEncodingType using
sortnames
                AckRecord for Element
endtype         (* FIFOacksType0 *)

type        FIFOacksType is FIFOacksType0 renamedby
sortnames
            FIFOacks for FIFO
opnames
            NoAck    for Nothing
endtype         (* FIFOacksType *)

(*============================================*)
(*              Main behaviour                *)
(*============================================*)

behaviour

    Control_Team [mgcs_ch, gcs_ch, out_ch]

where

(********************************************
  *
  *        Structure of Control Team
  *
  ********************************************)

process Control_Team [mgcs_ch, gcs_ch, out_ch] : noexit :=

    hide agcs_ch in (* Administrative channel *)
    (
        MGCS[agcs_ch, mgcs_ch](NoGCS)           (* Management *)

        |[agcs_ch]|

        Spawn_GCS[agcs_ch, gcs_ch, out_ch]      (* GCS spawning *)
    )

endproc (* Control_Team *)


(***************************************************************************
  *
  *      Master Group Communication Server (MGCS)
  *
  *      Listens on the MGCS channel for requests for:
  *          - the creation of a new Group Communication Server (GCS)
  *          - the list of existing GCS
  *
  ***************************************************************************)
```

```
process MGCS[agcs_ch, mgcs_ch](GCSlist:GroupList) :noexit:=

    (* Request for the creation of a new GCS *)
    (* the ID of the group must not be used by an existing group *)

    mgcs_ch ?caller:MID !CREATEGROUP ?newgroupid:GID ?infos:msg;
    (
        [newgroupid IsIn GCSlist] ->
            mgcs_ch !caller! GROUPEXISTS !newgroupid;
                MGCS[agcs_ch, mgcs_ch](GCSlist)
        []
        [newgroupid NotIn GCSlist] ->
            agcs_ch !CREATEGROUP !newgroupid
                    !Insert(caller.ChanID(infos), NoMBR) ! infos;
                mgcs_ch !caller !GROUPCREATED !newgroupid;
                    MGCS[agcs_ch, mgcs_ch](Insert(newgroupid, GCSlist))
    )

    []

    (* Request for the list of existing groups *)
    mgcs_ch ?caller:MID !GROUPS;
        mgcs_ch !caller !GCSlist;
            MGCS[agcs_ch, mgcs_ch](GCSlist)

    []

    (* Process destruction of a GCS. All verifications were done by the GCS. *)
    (* Used to update the goup list database *)
    [GCSlist ne NoGCS]->
        (* allowed only if there exists at least one group *)
        agcs_ch !GROUPDELETED ?id:GID;
            MGCS [agcs_ch, mgcs_ch](Remove(id, GCSlist))

endproc (* MGCS *)

(***************************************************************************
  *
  *      Spawn Group Communication Server
  *
  *      Creates a new GCS and forwards messages to it.
  *
  ***************************************************************************)

process Spawn_GCS[agcs_ch, gcs_ch, out_ch] :noexit:=

    agcs_ch !CREATEGROUP ?id:GID ?mbrL:MemberList ?infos: msg;
        (
            GCS_Team[agcs_ch, gcs_ch, out_ch] (id, mbrL, infos)
            |||
            Spawn_GCS[agcs_ch, gcs_ch, out_ch]
        )

endproc (* Spawn_GCS *)


(****************************************
  *
  *        Structure of GCS Team
```

```
 *
 ****************************************)

process GCS_Team[agcs_ch, gcs_ch, out_ch]
                (id:GID, mbrL:MemberList, infos:msg) : exit :=

    hide inter_ch in
        DoubleBuffer[agcs_ch, gcs_ch, inter_ch](id, false, NoReq, NoAck)
        |[inter_ch]|
        GCS[inter_ch, out_ch] (id, mbrL, infos)

endproc (* GCS_Team *)

(***************************************************************************
 *
 *      DoubleBuffer
 *
 *      Routes messages for group ID from gcs_ch on inter_ch; ignores others.
 *      Routes back acknowledgements to sender or to master.
 *      Used for decoupling. Avoids waiting for all GCS to be ready for a
 *      request to be processed.
 *
 *      Requests and acknowledgements are buffered.
 *      We use an infinite, 2-way buffer that holds two ordered lists
 *      (for acks and requests).
 *
 ***************************************************************************)

process DoubleBuffer[agcs_ch, gcs_ch, inter_ch](id:GID,
                                                terminated: Bool,
                                                bufreqs:FIFOreqs,
                                                bufacks:FIFOacks ) :exit:=
    (* Buffer requests from senders. *)
    (* Refuse them if group is deleted (terminated) *)
    [Not(terminated)] ->
       (
           gcs_ch ?sender:MID !id ?req:Request ?msg:Msg;
               DoubleBuffer[agcs_ch, gcs_ch, inter_ch](id, terminated,
                            Put(ReqElem(sender, req, msg), bufreqs), bufacks)
        )

    []

    (* Forwards buffered requests to GCS *)
    [Not(IsEmpty(bufreqs))] ->
        (
           inter_ch !S(Get(bufreqs)) !R(Get(bufreqs)) !M(Get(bufreqs));
               DoubleBuffer[agcs_ch, gcs_ch, inter_ch](id, terminated,
                            Consume(bufreqs), bufacks)
        )

    []

    (* Buffer acks from GCS. *)
    inter_ch ?sender:MID ?ack:Ack;
        DoubleBuffer[agcs_ch, gcs_ch, inter_ch](id, terminated,
                bufreqs, Put(AckElem(sender, ack), bufacks))

    []
```

```
    (* Forwards buffered acks to senders *)
    [Not(IsEmpty(bufacks))] ->
        (
            [A(Get(bufacks)) eq GROUPDELETED] ->
                (* Intercept this special case of ack. *)
                (* Informs MGCS of deletion, for dBase update. *)
                agcs_ch ! GROUPDELETED ! id;
                    (* Forward ack to sender, empty the request buffer, *)
                    (* and terminate listening *)
                    gcs_ch !S(Get(bufacks)) !A(Get(bufacks)) !id;
                        DoubleBuffer[agcs_ch, gcs_ch, inter_ch](id,
                                     true, NoReq, Consume(bufacks))
            []
            [A(Get(bufacks)) ne GROUPDELETED] ->
                (* Forward ack to sender *)
                gcs_ch !S(Get(bufacks)) !A(Get(bufacks)) !id;
                    DoubleBuffer[agcs_ch, gcs_ch, inter_ch](id, terminated,
                                 bufreqs, Consume(bufacks))
        )

    []

    (* After everyone has been informed of the group deletion, terminate. *)
    (* Waits for Req and Ack buffers to be empty *)
    [IsEmpty(bufreqs) and IsEmpty(bufacks) and terminated] ->
        inter_ch ! GROUPDELETED;
            exit

endproc (* DoubleBuffer *)

(***************************************************************************
  *
  *      Group Communication Server (GCS)
  *
  *      inter_ch: to communicate with sender and master (for administration)
  *      out_ch:   to multicast messages to members
  *
  *      Receives requests on inter_ch with event structure:
  *                 !MID !Request !Msg
  *      Gives acknowledgements with event structure:
  *                 !MID !Ack
  *
  ***************************************************************************)


process GCS[inter_ch, out_ch](id:GID, mbrL:MemberList, infos:msg) :exit:=

    (* gets the request from the sender *)
    inter_ch ?sender:MID ?req:Request ?msg:Msg;
    (
        [req eq DELETEGROUP] ->
            Req_DeleteGroup[inter_ch, out_ch](sender, msg, id, mbrL, infos)
        []
        (* provides the list of group members *)
        [req eq MEMBERS] ->
            Req_Members[inter_ch, out_ch](sender, msg, id, mbrL, infos)
        []
        (* subscription *)
```

```
    [req eq REGISTER] ->
        Req_Register[inter_ch, out_ch](sender, msg, id, mbrL, infos)
    []
    (* unsubscription *)
    [req eq DEREGISTER] ->
        Req_DeRegister[inter_ch, out_ch](sender, msg, id, mbrL, infos)
    []
    (* destruction of the group by the administrator *)
    (* change the moderator *)
    [req eq CHANGEMODER] ->
        Req_ChangeModer[inter_ch, out_ch](sender, msg, id, mbrL, infos)
    []
    (* change the administrator *)
    [req eq CHANGEADMIN] ->
        Req_ChangeAdmin[inter_ch, out_ch](sender, msg, id, mbrL, infos)
    []

    (* multicasting messages; will block until successful *)
    [req eq MULTICAST] ->
        Req_Multicast[inter_ch, out_ch](sender, msg, id, mbrL, infos)
)

where

process Req_DeleteGroup[inter_ch, out_ch](sender: MID, msg: Msg, id:GID,
                                       mbrL:MemberList, infos:msg) :exit:=

    [not(IsAdmin(infos)) or (Admin(infos) eq sender)] ->
        (
            (* Inform members other than sender *)
            Multicast[inter_ch](sender,Encode(GROUPWASDELETED),
                                    RemoveMBR(sender, mbrL), false)>>
                inter_ch !sender !GROUPDELETED;
                     inter_ch !GROUPDELETED;
                         exit
        )
    []
    [IsAdmin(infos) and (Admin(infos) ne sender)] ->
                inter_ch !sender !NOTADMINMODER;
                    GCS[inter_ch, out_ch](id, mbrL, infos)

endproc (* Req_DeleteGroup *)


process Req_Members[inter_ch, out_ch](sender: MID, msg: Msg, id:GID,
                                       mbrL:MemberList, infos:msg) :exit:=

    (* Check if group is private *)
    [not(IsPrivate(infos)) or
        (IsPrivate(infos) and (sender IsIn MembersOnly(mbrL)))] ->
        inter_ch !sender !MembersAre(MembersOnly(mbrL));
            GCS[inter_ch, out_ch](id, mbrL, infos)
    []
    [IsPrivate(infos) and (sender NotIn MembersOnly(mbrL))] ->
        inter_ch !sender !MEMBERNOTINGROUP;
            GCS[inter_ch, out_ch](id, mbrL, infos)

endproc (* Req_Members *)
```

```
process Req_Register[inter_ch, out_ch](sender: MID, msg: Msg, id:GID,
                                       mbrL:MemberList, infos:msg) :exit:=

    [not(IsPrivate(infos))] ->
    (* Register only if group is public *)
        inter_ch !sender !REGISTERED;
        (
            [sender NotIn MembersOnly(mbrl)] ->
               (* Insert pair MID.CID *)
                GCS[inter_ch, out_ch]
                   (id, Insert(sender.ChanId(msg), mbrL), infos)
            []

            [sender IsIn MembersOnly(mbrl)] ->
                (* Modify CID only *)
                GCS[inter_ch, out_ch]
                    (id, Insert(sender.ChanId(msg),
                                RemoveMBR(sender,mbrL)), infos)
        )
    []
    [IsPrivate(infos)] ->
    (
        [sender ne Admin(infos)] ->
        (* Cannot register; group is private *)
            inter_ch !sender !NOTADMINMODER;
                GCS[inter_ch, out_ch](id, mbrL, infos)
        []
        [sender eq Admin(infos)] ->
        (* Admin can register another member in private group *)
            inter_ch !sender !REGISTERED;
            (
                [NewMember(msg) NotIn MembersOnly(mbrl)] ->
                   (* Insert pair MID.CID *)
                    GCS[inter_ch, out_ch]
                        (id, Insert(NewMember(msg).ChanId(msg),
                                    mbrL), infos)
                []
                [NewMember(msg) IsIn MembersOnly(mbrl)] ->
                    (* Modify CID only *)
                    GCS[inter_ch, out_ch]
                        (id, Insert(NewMember(msg).ChanId(msg),
                         RemoveMBR(NewMember(msg),mbrL)), infos)
            )
    )

endproc (* Req_Register *)


process Req_DeRegister[inter_ch, out_ch](sender: MID, msg: Msg, id:GID,
                                         mbrL:MemberList, infos:msg) :exit:=

    (* Is sender a member of the group? *)
    [sender IsIn MembersOnly(mbrL)] ->
        inter_ch !sender !DEREGISTERED;
        (
            [Card(mbrL) eq succ(0)] ->
                (*  the GCS dies if no member left *)
                (* We do not need to inform the members... *)
```

```
                    inter_ch !sender !GROUPDELETED;
                        inter_ch !GROUPDELETED;
                            exit
            []
            [Card(mbrL) ne succ(0)] ->
                GCS[inter_ch, out_ch](id, RemoveMBR(sender, mbrL), infos)
        )
    []
    [sender NotIn MembersOnly(mbrL)] ->
        inter_ch !sender !MEMBERNOTINGROUP;
            GCS[inter_ch, out_ch](id, mbrL, infos)

endproc (* Req_DeRegister *)


process Req_ChangeModer[inter_ch, out_ch](sender: MID, msg: Msg, id:GID,
                                        mbrL:MemberList, infos:msg) :exit:=

    (* The group does not have a moderator *)
    (* (and the sender is not admin) *)
    [not(IsModerated(infos)) and
     not(IsAdmin(infos) and (Admin(infos) eq sender))] ->
        inter_ch !sender !NOMODERGROUP;
            GCS[inter_ch, out_ch](id, mbrL, infos)
    []
    (* The sender does not have the privilege to change moder *)
    (* (neither admin, nor moderator *)
    [(IsModerated(infos) and (Moderator(infos) ne sender)) and
     ((IsAdmin(infos) and (Admin(infos) ne sender) or
       not(IsAdmin(infos)))) ]->
            inter_ch !sender !NOTADMINMODER;
                GCS[inter_ch, out_ch](id, mbrL, infos)
    []
    (* The sender has the privilege. *)
    [(IsModerated(infos) and (Moderator(infos) eq sender)) or
     (IsAdmin(infos) and (Admin(infos) eq sender))] ->
        (
            (* If the group is closed, *)
            (* then moderator must be a group member *)
            [not(IsOpened(infos)) and (Moderator(msg) NotIn mbrL)] ->
                inter_ch !sender !MEMBERNOTINGROUP;
                    GCS[inter_ch, out_ch](id, mbrL, infos)
            []
            [IsOpened(infos) or (Moderator(msg) IsIn mbrL)] ->
                inter_ch !sender !ADMINMODERCHANGED;
                    GCS[inter_ch, out_ch]
                        (id, mbrL, SetModer(Moderator(msg), IsPrivate(msg), infos))

        )

endproc (* Req_ChangeModer *)


process Req_ChangeAdmin[inter_ch, out_ch](sender: MID, msg: Msg, id:GID,
                                        mbrL:MemberList, infos:msg) :exit:=

    (* The group does not have an administrator *)
    [not(IsAdmin(infos))] ->
        inter_ch !sender !NOADMINGROUP;
```

```
                GCS[inter_ch, out_ch](id, mbrL, infos)
        []
        (* The sender does not have the privilege to change admin *)
        [IsAdmin(infos) and (Admin(infos) ne sender)] ->
            inter_ch !sender !NOTADMINMODER;
                GCS[inter_ch, out_ch](id, mbrL, infos)
        []
        (* The sender has the privilege. *)
        [IsAdmin(infos) and (Admin(infos) eq sender)] ->
            (
                [Admin(msg) NotIn mbrL] ->
                    inter_ch !sender !MEMBERNOTINGROUP;
                        GCS[inter_ch, out_ch](id, mbrL, infos)
                []
                [Admin(msg) IsIn mbrL] ->
                    inter_ch !sender !ADMINMODERCHANGED;
                        GCS[inter_ch, out_ch]
                           (id, mbrL, SetAdmin(Admin(msg), IsPrivate(msg), infos))
            )

    endproc (* Req_ChangeAdmin *)


    process Req_Multicast[inter_ch, out_ch](sender: MID, msg: Msg, id:GID,
                                         mbrL:MemberList, infos:msg) :exit:=

    (* Is the group Opened, or is the sender a member of the group? *)
    [IsOpened(infos) or
     (not(IsOpened(infos)) and (sender IsIn MembersOnly(mbrl)))] ->
        (
            (* Not moderated, or sender is moderator *)
            [not(IsModerated(infos)) or
             (IsModerated(infos) and (sender eq Moderator(infos)))] ->
                (
                    Multicast[out_ch](sender, msg, mbrL, true) >>
                        inter_ch !sender !MESSAGESENT;
                            GCS[inter_ch, out_ch](id, mbrL, infos)
                )
            []
            (* Moderated, and sender is not moderator. *)
            (* Forward to group moderator only, for approval *)
            [IsModerated(infos) and (sender ne Moderator(infos))] ->
                (
                    inter_ch !Moderator(infos) !ToApprove(msg);
                        inter_ch !sender !SENTTOMODERATOR;
                            GCS[inter_ch, out_ch](id, mbrL, infos)
                )
        )
    []
    [not(IsOpened(infos)) and (sender NotIn MembersOnly(mbrl))]->
        (
            inter_ch !sender !MEMBERNOTINGROUP;
                GCS[inter_ch, out_ch](id, mbrL, infos)
        )

    endproc (* Req_Multicast *)

endproc (* GCS *)
```

```
(************************************************************************
  *
  *      Multicast
  *
  *      Send the message to all subscriber of the group (concurrently).
  *      No other message will be multicast until all members of the group
  *      received it.
  *
  ************************************************************************)

process Multicast[out](sender:MID, msg:Msg, mbrL:MemberList, UseChannel: Bool)
                 : exit :=

    [mbrL eq NoMBR] ->
         exit

    []

    [mbrL ne NoMBR] ->
        (
            (
                [UseChannel] ->
                    (* Multicasts message to members on their appropriate
                        data channel *)
                    out !Top(mbrL) !sender !msg;
                        exit
                []
                [Not(UseChannel)] ->
                    (* Use request/ack channel *)
                    out !MID(Top(mbrl)) !GetAck(msg);
                        exit
            )
            |||
            Multicast[out](sender, msg, Tail(mbrL), UseChannel) (* loop... *)
        )

endproc (* Multicast *)


(*============================================*)
(*                 Test Cases                 *)
(*============================================*)

(* Check group creation and deletion. *)
process Test1 [mgcs_ch, gcs_ch, out_ch, success]:noexit :=

    mgcs_ch !0 of MID !CREATEGROUP !a of GID
            !Encode(Mail, 4, false, Nobody, true, false, false, Nobody);
    mgcs_ch !0 of MID !GROUPCREATED !a of GID;

    mgcs_ch !9 of MID !CREATEGROUP !b of GID
            !Encode(Video, 4, false, Nobody, true, false, false, Nobody);
    mgcs_ch !9 of MID !GROUPCREATED !b of GID;

    mgcs_ch !0 of MID !GROUPS;
    mgcs_ch !0 of MID !Insert(b, Insert(a, NoGCS));

    gcs_ch !1 of MID !a of GID !REGISTER !Encode(0 of CID);
```

```
    gcs_ch !1 of MID !REGISTERED !a of GID;

    mgcs_ch !1 of MID !GROUPS;
    mgcs_ch !1 of MID !Insert(b, Insert(a, NoGCS));

    gcs_ch !1 of MID !a of GID !MEMBERS !NoMsg;
    gcs_ch !1 of MID !MembersAre(Insert(1, Insert(0, Empty)))  !a of GID;

    gcs_ch !9 of MID !a of GID !DELETEGROUP !NoMsg;
    (
        gcs_ch !0 of MID !GROUPWASDELETED !a of GID; exit
        |||
        gcs_ch !1 of MID !GROUPWASDELETED !a of GID; exit
    )
    >>
    gcs_ch !9 of MID !GROUPDELETED !a of GID;

    mgcs_ch !1 of MID !GROUPS;
    mgcs_ch !1 of MID !Insert(b, NoGCS);
    success;
    stop

endproc (* Test1 *)

(* Check group administration. *)
process Test2 [mgcs_ch, gcs_ch, out_ch, success]:noexit :=

    mgcs_ch !0 of MID !CREATEGROUP !a of GID
            !Encode(Mail, 4, true, 0, true, false, false, Nobody);
            (* NOTE: We should not have to indicate the admin. *)
    mgcs_ch !0 of MID !GROUPCREATED !a of GID;

    mgcs_ch !1 of MID !CREATEGROUP !b of GID
            !Encode(Text, 4, true, 1, true, false, false, Nobody);
    mgcs_ch !1 of MID !GROUPCREATED !b of GID;

    mgcs_ch !9 of MID !CREATEGROUP !c of GID
            !Encode(Video, 4, false, Nobody, true, false, false, Nobody);
    mgcs_ch !9 of MID !GROUPCREATED !c of GID;

    mgcs_ch !0 of MID !GROUPS;
    mgcs_ch !0 of MID !Insert(c, Insert(b, Insert(a, NoGCS)));

    gcs_ch !2 of MID !a of GID !REGISTER !Encode(0 of CID);
    gcs_ch !2 of MID !REGISTERED !a of GID;

    gcs_ch !9 of MID !a of GID !DELETEGROUP !NoMsg;
    gcs_ch !9 of MID !NOTADMINMODER !a of GID;

    gcs_ch !1 of MID !b of GID !DELETEGROUP !NoMsg;
    gcs_ch !1 of MID !GROUPDELETED !b of GID;

    gcs_ch !1 of MID !a of GID !CHANGEADMIN !Encode(2 of MID, false);
    gcs_ch !1 of MID !NOTADMINMODER !a of GID;

    gcs_ch !2 of MID !a of GID !CHANGEADMIN !Encode(2 of MID, false);
    gcs_ch !2 of MID !NOTADMINMODER !a of GID;

    gcs_ch !0 of MID !a of GID !CHANGEADMIN !Encode(1 of MID, false);
```

```
    gcs_ch !0 of MID !MEMBERNOTINGROUP !a of GID;

    gcs_ch !0 of MID !a of GID !CHANGEADMIN !Encode(2 of MID, false);
    gcs_ch !0 of MID !ADMINMODERCHANGED !a of GID;

    mgcs_ch !0 of MID !GROUPS;
    mgcs_ch !0 of MID !Insert(c, Insert(a, NoGCS));

    gcs_ch !2 of MID !a of GID !CHANGEADMIN !Encode(2 of MID, false);
    gcs_ch !2 of MID !ADMINMODERCHANGED !a of GID;

    gcs_ch !0 of MID !a of GID !DELETEGROUP !NoMsg;
    gcs_ch !0 of MID !NOTADMINMODER !a of GID;

    gcs_ch !2 of MID !a of GID !DELETEGROUP !NoMsg;
    gcs_ch !0 of MID !GROUPWASDELETED !a of GID;
    gcs_ch !2 of MID !GROUPDELETED !a of GID;

    mgcs_ch !0 of MID !GROUPS;
    mgcs_ch !0 of MID !Insert(c, NoGCS);

    gcs_ch !9 of MID !c of GID !CHANGEADMIN !Encode(2 of MID, false);
    gcs_ch !9 of MID !NOADMINGROUP !c of GID;

    (* Make this group administered and private *)
    mgcs_ch !5 of MID !CREATEGROUP !d of GID
            !Encode(Audio, 4, true, 5, true, true, false, Nobody);
    mgcs_ch !5 of MID !GROUPCREATED !d of GID;

    gcs_ch !2 of MID !d of GID !REGISTER !Encode(0 of CID);
    gcs_ch !2 of MID !NOTADMINMODER !d of GID;

    gcs_ch !5 of MID !d of GID !REGISTER !Encode(6 of MID, 7 of CID);
    gcs_ch !5 of MID !REGISTERED !d of GID;

    gcs_ch !9 of MID !d of GID !MEMBERS !NoMsg;
    gcs_ch !9 of MID !MEMBERNOTINGROUP !d of GID ;

    gcs_ch !5 of MID !d of GID !MEMBERS !NoMsg;
    gcs_ch !5 of MID !MembersAre(Insert(6, Insert(5, Empty))) !d of GID;

    success;
    stop

endproc (* Test2 *)

(* Check multicasting *)
process Test3 [mgcs_ch, gcs_ch, out_ch, success]:noexit :=

    mgcs_ch !0 of MID !CREATEGROUP !a of GID
            !Encode(Mail, 0, false, Nobody, true, false, false, Nobody);
    mgcs_ch !0 of MID !GROUPCREATED !a of GID;
    gcs_ch !1 of MID !a of GID !REGISTER !Encode(0 of CID);
    gcs_ch !1 of MID !REGISTERED !a of GID;
    gcs_ch !2 of MID !a of GID !REGISTER !Encode(0 of CID);
    gcs_ch !2 of MID !REGISTERED !a of GID;
    gcs_ch !9 of MID !a of GID !MEMBERS !NoMsg;
    gcs_ch !9 of MID !MembersAre(Insert(2, Insert(1, Insert(0, Empty))))
                     !a of GID;
```

```
    gcs_ch !7 of MID !a of GID !MULTICAST !Encode(Hello);
    (* NOTE: All CIDs are 0 by default for the moment *)
    out_ch !0.0 !7 of MID !Encode(Hello);      (* any order! *)
    out_ch !2.0 !7 of MID !Encode(Hello);
    out_ch !1.0 !7 of MID !Encode(Hello);
    gcs_ch !7 of MID !MESSAGESENT !a of GID;

    gcs_ch !9 of MID !a of GID !DELETEGROUP !NoMsg;
    (* Each member is informed *)
    (
        gcs_ch !1 of MID !GROUPWASDELETED !a of GID; exit  (* any order! *)
        |||
        gcs_ch !2 of MID !GROUPWASDELETED !a of GID; exit
        |||
        gcs_ch !0 of MID !GROUPWASDELETED !a of GID; exit
    )
    >>
    gcs_ch !9 of MID !GROUPDELETED !a of GID;

    mgcs_ch !1 of MID !GROUPS;
    mgcs_ch !1 of MID !NoGCS;
    success;
    stop

endproc (* Test3 *)

(* More realistic scenario, with all data (even if not used) *)
process Test4 [mgcs_ch, gcs_ch, out_ch, success]:noexit :=

    (* Create group A, administered by 0. CID of member 0 is 4. *)
    mgcs_ch !0 of MID !CREATEGROUP !A of GID
              !Encode(Mail, 4, true, 0, true, false, false, 0);
    mgcs_ch !0 of MID !GROUPCREATED !A of GID;

    (* 1 joins A on its CID 8 *)
    gcs_ch !1 of MID !a of GID !REGISTER !Encode(8 of CID);
    gcs_ch !1 of MID !REGISTERED !a of GID;

    (* 2 joins A on its CID 2 *)
    gcs_ch !2 of MID !a of GID !REGISTER !Encode(2 of CID);
    gcs_ch !2 of MID !REGISTERED !a of GID;

    (* List members of group A *)
    gcs_ch !9 of MID !a of GID !MEMBERS !NoMsg;
    gcs_ch !9 of MID !MembersAre(Insert(2, Insert(1, Insert(0, Empty))))
                     !a of GID;

    (* 7 multicasts a Hello message on group A *)
    gcs_ch !7 of MID !a of GID !MULTICAST !Encode(Hello);
    out_ch !0.4 !7 of MID !Encode(Hello);      (* any order! *)
    out_ch !2.2 !7 of MID !Encode(Hello);
    out_ch !1.8 !7 of MID !Encode(Hello);
    gcs_ch !7 of MID !MESSAGESENT !a of GID;

    (* 2 tries to destroy the administred group A *)
    gcs_ch !2 of MID !a of GID !DELETEGROUP !NoMsg;
    gcs_ch !2 of MID !NOTADMINMODER !a of GID;
```

```
    (* 0 decides to destroy the administred group A *)
    gcs_ch !0 of MID !a of GID !DELETEGROUP !NoMsg;
    (* Each other member is informed *)
    (
        gcs_ch !2 of MID !GROUPWASDELETED !a of GID; exit  (* any order! *)
        |||
        gcs_ch !1 of MID !GROUPWASDELETED !a of GID; exit
    )
    >>
    gcs_ch !0 of MID !GROUPDELETED !a of GID;

    (* List the remaining groups *)
    mgcs_ch !1 of MID !GROUPS;
    mgcs_ch !1 of MID !NoGCS;
    success;
    stop

endproc (* Test4 *)

(* Check possible deadlocks with DoubleBuffer *)
process Test5 [mgcs_ch, gcs_ch, out_ch, success]:noexit :=

    (* Create group A, administered by 0. CID of member 0 is 4. *)
    mgcs_ch !0 of MID !CREATEGROUP !a of GID
              !Encode(Mail, 4, true, 0, false, false, false, 0);
    mgcs_ch !0 of MID !GROUPCREATED !a of GID;

    (* 9 lists members of group A *)
    gcs_ch !9 of MID !a of GID !MEMBERS !NoMsg;
    (* so does 8 right after... *)
    gcs_ch !8 of MID !a of GID !MEMBERS !NoMsg;

    (* In theory, we should have this, but Listen will deadlock... *)
    (
        gcs_ch !9 of MID !MembersAre(Insert(0, Empty)) !a of GID; exit
        |||
        gcs_ch !8 of MID !MembersAre(Insert(0, Empty)) !a of GID; exit
    )
    >>
    success;
    stop

endproc (* Test5 *)

(* Test Opened/closed, with all data *)
process Test6 [mgcs_ch, gcs_ch, out_ch, success]:noexit :=

    (* Create Closed group A, administered by 0. CID of member 0 is 4.  *)
    mgcs_ch !0 of MID !CREATEGROUP !A of GID
              !Encode(Mail, 4, true, 0, false, false, false, 0);
    mgcs_ch !0 of MID !GROUPCREATED !A of GID;

    (* 1 joins A on its CID 8 *)
    gcs_ch !1 of MID !a of GID !REGISTER !Encode(8 of CID);
    gcs_ch !1 of MID !REGISTERED !a of GID;

    (* Create Opened group B, administered by 2. CID of member 2 is 4. *)
    mgcs_ch !2 of MID !CREATEGROUP !B of GID
              !Encode(Mail, 4, true, 0, true, false, false, 0);
```

```
    mgcs_ch !2 of MID !GROUPCREATED !B of GID;

    (* 3 joins B on its CID 2 *)
    gcs_ch !3 of MID !b of GID !REGISTER !Encode(2 of CID);
    gcs_ch !3 of MID !REGISTERED !b of GID;

    (* 7 multicasts a Hello message on group A (Closed). Should not work. *)
    gcs_ch !7 of MID !a of GID !MULTICAST !Encode(Hello);
    gcs_ch !7 of MID !MEMBERNOTINGROUP !a of GID;

    (* 7 multicasts a Hello message on group B (Opened). Should work. *)
    gcs_ch !7 of MID !b of GID !MULTICAST !Encode(Hello);
    out_ch !2.4 !7 of MID !Encode(Hello);                          (* any order! *)
    out_ch !3.2 !7 of MID !Encode(Hello);
    gcs_ch !7 of MID !MESSAGESENT !b of GID;

    (* 1 multicasts a Hello message on group A (Closed). Should work. *)
    gcs_ch !1 of MID !a of GID !MULTICAST !Encode(Hello);
    out_ch !0.4 !1 of MID !Encode(Hello);                          (* any order! *)
    out_ch !1.8 !1 of MID !Encode(Hello);
    gcs_ch !1 of MID !MESSAGESENT !a of GID;

    (* 2 multicasts a Hello message on group B (Opened). Should work. *)
    gcs_ch !2 of MID !b of GID !MULTICAST !Encode(Hello);
    out_ch !3.2 !2 of MID !Encode(Hello);                          (* any order! *)
    out_ch !2.4 !2 of MID !Encode(Hello);
    gcs_ch !2 of MID !MESSAGESENT !b of GID;

    success;
    stop

endproc (* Test6 *)


(* Test Moderator functionalities, with all data *)
process Test7 [mgcs_ch, gcs_ch, out_ch, success]:noexit :=

    (* Create Opened/Moderated group A, administered by 0, *)
    (* moderated by 1. CID of member 0 is 4.  *)
    mgcs_ch !0 of MID !CREATEGROUP !A of GID
              !Encode(Mail, 4, true, 0, true, false, true, 1);
    mgcs_ch !0 of MID !GROUPCREATED !A of GID;

    (* 1 joins A on its CID 8 *)
    gcs_ch !1 of MID !a of GID !REGISTER !Encode(8 of CID);
    gcs_ch !1 of MID !REGISTERED !a of GID;

    (* Create Closed/Unmoderated group B, administered by 2, *)
    (* CID of member 2 is 4. *)
    mgcs_ch !2 of MID !CREATEGROUP !B of GID
              !Encode(Mail, 4, true, 0, false, false, false, Nobody);
    mgcs_ch !2 of MID !GROUPCREATED !B of GID;

    (* 3 joins B on its CID 2 *)
    gcs_ch !3 of MID !b of GID !REGISTER !Encode(2 of CID);
    gcs_ch !3 of MID !REGISTERED !b of GID;

    (* 7 multicasts a Hello message on group A (Moderated). *)
    (* Should be forwarded to Moderator *)
```

```
gcs_ch !7 of MID !a of GID !MULTICAST !Encode(Hello);
gcs_ch !1 of MID !ToApprove(Encode(Hello)) !a of GID;
gcs_ch !7 of MID !SENTTOMODERATOR !a of GID;

(* 2 multicasts a Hello message on group B (Closed). Should work. *)
gcs_ch !2 of MID !b of GID !MULTICAST !Encode(Hello);
out_ch !2.4 !2 of MID !Encode(Hello);                          (* any order! *)
out_ch !3.2 !2 of MID !Encode(Hello);
gcs_ch !2 of MID !MESSAGESENT !b of GID;

(* Moderator (1) multicasts a Hello message on group A (Moderated). *)
(* Should work as normal. *)
gcs_ch !1 of MID !a of GID !MULTICAST !Encode(Hello);
out_ch !0.4 !1 of MID !Encode(Hello);                          (* any order! *)
out_ch !1.8 !1 of MID !Encode(Hello);
gcs_ch !1 of MID !MESSAGESENT !a of GID;

(* 1 changes moderator to 2 *)
gcs_ch !1 of MID !a of GID !CHANGEMODER !Encode(2 of MID, true);
gcs_ch !1 of MID !ADMINMODERCHANGED !a of GID;
gcs_ch !7 of MID !a of GID !MULTICAST !Encode(Hello);
gcs_ch !2 of MID !ToApprove(Encode(Hello)) !a of GID;
gcs_ch !7 of MID !SENTTOMODERATOR !a of GID;

(* 0 changes moderator to 7 *)
gcs_ch !0 of MID !a of GID !CHANGEMODER !Encode(7 of MID, true);
gcs_ch !0 of MID !ADMINMODERCHANGED !a of GID;

(* 1 tries to change moderator of A to itself, but fails *)
gcs_ch !1 of MID !a of GID !CHANGEMODER !Encode(1 of MID, true);
gcs_ch !1 of MID !NOTADMINMODER !a of GID;

(* 1 tries to change moderator of B to itself, but fails *)
gcs_ch !1 of MID !b of GID !CHANGEMODER !Encode(1 of MID, true);
gcs_ch !1 of MID !NOMODERGROUP !b of GID;

(* Create Closed/Moderated group C, administered by 0, *)
(* moderated by 0. CID of member 0 is 4.  *)
mgcs_ch !0 of MID !CREATEGROUP !C of GID
          !Encode(Mail, 4, true, 0, false, false, true, 0);
mgcs_ch !0 of MID !GROUPCREATED !C of GID;

(* Moderator tries to change moderator of C to 1, but fails *)
gcs_ch !0 of MID !c of GID !CHANGEMODER !Encode(1 of MID, true);
gcs_ch !0 of MID !MEMBERNOTINGROUP !c of GID;

(* Admin makes group A moderated *)
gcs_ch !0 of MID !a of GID !CHANGEMODER !Encode(1 of MID, true);
gcs_ch !0 of MID !ADMINMODERCHANGED !a of GID;
gcs_ch !7 of MID !a of GID !MULTICAST !Encode(Hello);
gcs_ch !1 of MID !ToApprove(Encode(Hello)) !a of GID;
gcs_ch !7 of MID !SENTTOMODERATOR !a of GID;

(* Admin makes group A unmoderated again *)
gcs_ch !0 of MID !a of GID !CHANGEMODER !Encode(Nobody, false);
gcs_ch !0 of MID !ADMINMODERCHANGED !a of GID;
gcs_ch !1 of MID !a of GID !CHANGEMODER !Encode(1 of MID, true);
gcs_ch !1 of MID !NOMODERGROUP !a of GID;
```

```
    success;
    stop

endproc (* Test7 *)

endspec (* GCS, Group Communication Server *)
```