...
...

Process Model Attributes

attribute	*value*	*type*	*default value*
Sender	promoted	integer	0
Receiver	promoted	integer	0
NGRPS	promoted	integer	0
SESSMIN	promoted	integer	0
SESSMAX	promoted	integer	0
DATA_IAT	promoted	double	0.0
SESS_IAT	promoted	integer	0

Header Block

```
#include<string.h>

#define PK_ARRIVAL op_intrpt_type() == OPC_INTRPT_STRM
#define JOIN_TER  (op_intrpt_type() == OPC_INTRPT_SELF && op_intrpt_code() == 0 )
#define SEND_DATA  (op_intrpt_type() == OPC_INTRPT_SELF && op_intrpt_code() == 1 )
/*  #define SEND_RESERVE  (op_intrpt_type() == OPC_INTRPT_SELF && op_intrpt_code() == 2 )  */
#define END_SIM     op_intrpt_type() == OPC_INTRPT_ENDSIM

#define IGMP_STREAM      1
#define RSVP_STREAM      0
#define IP_STREAM        2

#define InLIST_SIZE      100

#define ON               1
#define OFF              0

#define SESSION_MSG      100
#define SENDER_MSG       101
#define RESERVE_MSG      102
#define RELEASE_MSG      103
#define DATA_MSG         104

#define MULTICAST              500
#define DEFAULT_DEST_PORT80
#define DEFAULT_SRC_PORT   80
#define DEFAULT_SEND_TTL   10
#define DEFAULT_SEND_BW    60000
#define DEFAULT_SEND_DELAY    1
#define DEFAULT_RECV_BW    50000
#define DEFAULT_RECV_DELAY    1

#define UDP_PROTOCOL_ID 20
#define MAX_INIT_TIME   20


#define VL if(OFF)
#define SVL if(OFF)
#define RADL if(OFF)

#define MAX_NO_SESSIONS     100
/*
**  IP Address
*/
typedef struct {
```

```
   int NetNum;
   int NodeNum;
}IPAddress;

typedef struct{
       IPAddress Address;
       int    Port;
}SenderInfo;

SenderInfo AllSendersList[InLIST_SIZE];

/*
** Data Packet Structure
*/
typedef struct{
   IPAddress SrcAddress;
   char    Data[500];
}SendDataPk;

typedef struct {
       IPAddress SenderAddr;
       IPAddress DestAddr;
       int SenderPort;
       double Bandwidth;
       double Delay;
}sd;

sd SessDirectory[MAX_NO_SESSIONS];

typedef struct {
       char Message[500];
}App_Data;
```

State Variable Block

```

Objid \MyObjId;
Objid \RsvpObjId;
Objid \IpObjId;
int \InStrm, \GrpId, \Sess, \NoOfGrps;
int \CurrSessionId, \SessionId, \RegSes[InLIST_SIZE];
double \Time, \DataRecv, \SessMin, \SessMax, \DataIat, \SessIat;
int \IsSender, \IsReceiver;
Distribution* \GrpDist, \TimeDist, \TermDist;

IPAddress \LocalAddress[InLIST_SIZE];
```

Temporary Variable Block

```
Packet* IgmpApplPkptr;
Packet* RsvpApplPkptr;
Packet* DataApplPkptr;

int RecvMsgType;
int RecvGrpId;
int RecvSessionId;
```

```
int DataAddr1, DataAddr2, DataAddr3;
FILE* fileptr;
```

forced state **Init**			
attribute	*value*	*type*	*default value*
name	Init	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **Init**

```
MyObjId = op_id_self();
RsvpObjId = op_topo_assoc( MyObjId, OPC_TOPO_ASSOC_OUT, OPC_OBJMTYPE_MODULE, 1 );
IpObjId = op_topo_assoc( RsvpObjId, OPC_TOPO_ASSOC_OUT, OPC_OBJMTYPE_MODULE, 0 );

DataRecv = 0;

op_ima_obj_attr_get( MyObjId, "NGRPS", &NoOfGrps );
op_ima_obj_attr_get( MyObjId, "SESSMIN", &SessMin );
op_ima_obj_attr_get( MyObjId, "SESSMAX", &SessMax );
op_ima_obj_attr_get( MyObjId, "DATA_IAT", &DataIat );
op_ima_obj_attr_get( MyObjId, "SESS_IAT", &SessIat );
op_ima_obj_attr_get( MyObjId, "Sender", &IsSender );
op_ima_obj_attr_get( MyObjId, "Receiver", &IsReceiver );

/*  LocalAddress[0] = ( IPAddress *)op_ima_obj_svar_get( RsvpObjId, "LocalAddress[0]" );  */

op_ima_obj_attr_get( IpObjId, "net_num0", &LocalAddress[0].NetNum );
op_ima_obj_attr_get( IpObjId, "node_num0", &LocalAddress[0].NodeNum );
if(OFF)   VL printf("LOCAL ADDRESS : %d.%d. SENDER = %d, RECEIVER = %d\n",
              LocalAddress[0].NetNum, LocalAddress[0].NodeNum, IsSender, IsReceiver );
/**
*** Generate a Random Variable between 0 and  Max. No of Groups
*** to select which Grp id to join
**/

GrpDist = op_dist_load( "uniform_int", 0, NoOfGrps );

/**
*** Time for the Application to join intially
**/
/*  TimeDist = op_dist_load( "exponential", SessIat, 0 );  */
TimeDist = op_dist_load( "uniform_int", 10, SessIat );
Time = op_dist_outcome( TimeDist );
op_intrpt_schedule_self( (op_sim_time() + Time), 0 );

Sess = 0;

/**
*** Session termiantion distrubution
**/
/*  TimeDist = op_dist_load( "exponential", SessIat, 0 );  */
```

...
...

transition **Init -> Idle**			
attribute	*value*	*type*	*default value*
name	tr_0	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **Idle**			
attribute	*value*	*type*	*default value*
name	Idle	string	st
enter execs	(empty)	textlist	(empty)
exit execs	(empty)	textlist	(empty)
status	unforced	toggle	unforced

transition **Idle -> Arr**			
attribute	*value*	*type*	*default value*
name	tr_1	string	tr
condition	PK_ARRIVAL	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **Idle -> EndSim**			
attribute	*value*	*type*	*default value*
name	tr_6	string	tr
condition	END_SIM	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **Idle -> Join**			
attribute	*value*	*type*	*default value*
name	tr_10	string	tr
condition	JOIN_TER	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **Idle -> Data_Send**			
attribute	*value*	*type*	*default value*
name	tr_11	string	tr
condition	SEND_DATA	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **Arr**			
attribute	*value*	*type*	*default value*
name	Arr	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **Arr**

```
InStrm = op_intrpt_strm();
RsvpApplPkptr = op_pk_get( op_intrpt_strm() );
switch(InStrm)
{
  case 0 :
          op_pk_nfd_get( RsvpApplPkptr, "UpcallType", &RecvMsgType );
    if( RecvMsgType == SESSION_MSG )
    {
      op_pk_nfd_get( RsvpApplPkptr, "SessionId", &RecvSessionId );
            VL  SVL printf("Received Session Id is %d\n", RecvSessionId );
      RegSes[RecvSessionId] = 1;
            CurrSessionId = RecvSessionId;
    }
  break;
  case 1 :
        {
            SendDataPk* RecvDataPk;
            IPAddress DataSndrAdd;
            RecvDataPk = ( SendDataPk* )op_prg_mem_alloc( sizeof(SendDataPk) );
            memset( RecvDataPk, 0, sizeof(SendDataPk) );
            op_pk_nfd_get( RsvpApplPkptr, "Data", &RecvDataPk );
            DataSndrAdd.NetNum = RecvDataPk->SrcAddress.NetNum;
            DataSndrAdd.NodeNum = RecvDataPk->SrcAddress.NodeNum;
            DataRecv++;
            VL  printf("Data Received from %d.%d\n", DataSndrAdd.NetNum, DataSndrAdd.NodeNum );
        }
  break;
  default :
    VL  printf("Packet has arrived from unrecognized stream\n");
  break;

}

op_pk_destroy( RsvpApplPkptr );
```

transition **Arr -> Idle**			
attribute	*value*	*type*	*default value*
name	tr_7	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline

...
...

forced state **Join**			
attribute	*value*	*type*	*default value*
name	Join	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **Join**

```
if(!Sess)
{
	GrpId = op_dist_outcome( GrpDist );

	/** Make Session call **/
	if(CurrSessionId == 0)
	{
		Packet* SessionPkptr;
		/*  GrpId = 1;  */
		SessionPkptr = op_pk_create_fmt("SessionCall");
		op_pk_nfd_set( SessionPkptr, "SessionId", 0 );
		op_pk_nfd_set( SessionPkptr, "DestNode", GrpId );
		op_pk_nfd_set( SessionPkptr, "DestNet", MULTICAST );
		op_pk_nfd_set( SessionPkptr, "DestPort", DEFAULT_DEST_PORT );

		RsvpApplPkptr = op_pk_create_fmt("RSVP_APP_CNTRL");
		op_pk_nfd_set( RsvpApplPkptr, "AppCall", SessionPkptr );
		op_pk_nfd_set( RsvpApplPkptr, "ApplCallType", SESSION_MSG );

		op_pk_send( RsvpApplPkptr, RSVP_STREAM );
	}

	/** Make Sender Call **/
	if(IsSender && CurrSessionId > 0)
	{
		Packet* SenderPkptr;
		SenderPkptr = op_pk_create_fmt("SenderCall");

		SessDirectory[GrpId].SenderAddr = LocalAddress[0];
		SessDirectory[GrpId].SenderPort = DEFAULT_SRC_PORT;
		SessDirectory[GrpId].DestAddr.NetNum = 500;
		SessDirectory[GrpId].DestAddr.NodeNum = GrpId;

		op_pk_nfd_set( SenderPkptr, "SessionId", CurrSessionId );
		op_pk_nfd_set( SenderPkptr, "SourceNet", LocalAddress[0].NetNum );
		op_pk_nfd_set( SenderPkptr, "SourceNode", LocalAddress[0].NodeNum );
		op_pk_nfd_set( SenderPkptr, "SourcePort", DEFAULT_SRC_PORT );
		op_pk_nfd_set( SenderPkptr, "ttl", DEFAULT_SEND_TTL );
		op_pk_nfd_set( SenderPkptr, "BandWidth", DEFAULT_SEND_BW );
		op_pk_nfd_set( SenderPkptr, "Delay", DEFAULT_SEND_DELAY );

		SessDirectory[GrpId].Bandwidth = DEFAULT_SEND_BW;
		SessDirectory[GrpId].Delay = DEFAULT_SEND_DELAY;

		RsvpApplPkptr = op_pk_create_fmt("RSVP_APP_CNTRL");
		op_pk_nfd_set( RsvpApplPkptr, "AppCall", SenderPkptr );
		op_pk_nfd_set( RsvpApplPkptr, "ApplCallType", SENDER_MSG );
		op_pk_send( RsvpApplPkptr, RSVP_STREAM );
		Sess = 1;
```

```
        IgmpApplPkptr = op_pk_create_fmt("MC_APP_CNTRL");
        op_pk_nfd_set( IgmpApplPkptr, "join", 1 );
        op_pk_nfd_set( IgmpApplPkptr, "terminate", 0 );
        op_pk_nfd_set( IgmpApplPkptr, "GRP_ID", GrpId );
        op_pk_send( IgmpApplPkptr, IGMP_STREAM );
    }

    /** Make the IGMP Join Call **/
    if( IsReceiver && CurrSessionId > 0 )
    {
        Packet* ReservePkptr;
        int sender_node, sender_net, sender_port;

        if( !(SessDirectory[GrpId].SenderAddr.NodeNum <= 0 ||
              SessDirectory[GrpId].SenderAddr.NetNum <= 0 ||
              SessDirectory[GrpId].DestAddr.NodeNum <=0) )
        {
            sender_node = SessDirectory[GrpId].SenderAddr.NodeNum;
            sender_net = SessDirectory[GrpId].SenderAddr.NetNum;
            sender_port = SessDirectory[GrpId].SenderPort;

            ReservePkptr = op_pk_create_fmt("ReserveCall");
            op_pk_nfd_set( ReservePkptr, "SessionId", CurrSessionId );
            op_pk_nfd_set( ReservePkptr, "SenderNode", sender_node );
            op_pk_nfd_set( ReservePkptr, "SenderNet", sender_net );
            op_pk_nfd_set( ReservePkptr, "SenderPort", sender_port );
            op_pk_nfd_set( ReservePkptr, "BandWidth", DEFAULT_RECV_BW );
            op_pk_nfd_set( ReservePkptr, "Delay", DEFAULT_RECV_DELAY );

            RsvpApplPkptr = op_pk_create_fmt("RSVP_APP_CNTRL");
            op_pk_nfd_set( RsvpApplPkptr, "AppCall", ReservePkptr );
            op_pk_nfd_set( RsvpApplPkptr, "ApplCallType", RESERVE_MSG );
            op_pk_send( RsvpApplPkptr, RSVP_STREAM );
            Sess = 1;
            IgmpApplPkptr = op_pk_create_fmt("MC_APP_CNTRL");
            op_pk_nfd_set( IgmpApplPkptr, "join", 1 );
            op_pk_nfd_set( IgmpApplPkptr, "terminate", 0 );
            op_pk_nfd_set( IgmpApplPkptr, "GRP_ID", GrpId );
            op_pk_send( IgmpApplPkptr, IGMP_STREAM );
        }
    }

    VL  RADL   printf( "Sending the join packet to the IGMP Layer \n" );

    /** Schedule the Interputs **/
    op_intrpt_schedule_self( (op_sim_time() + DataIat), 1);
    Time = op_dist_outcome( TermDist );
    op_intrpt_schedule_self( op_sim_time()+Time, 0);
}
else
{
    IgmpApplPkptr = op_pk_create_fmt( "MC_APP_CNTRL" );
    op_pk_nfd_set( IgmpApplPkptr, "join", 0 );
    op_pk_nfd_set( IgmpApplPkptr, "terminate", 1 );
    op_pk_nfd_set( IgmpApplPkptr, "GRP_ID", GrpId );
    op_pk_send( IgmpApplPkptr, IGMP_STREAM );
    VL  RADL   printf( "Sending the terminate packet to the IGMP Layer \n" );

    Sess = 0;
```

```
    /** Make Release Call **/
    RsvpApplPkptr = op_pk_create_fmt("RSVP_APP_CNTRL");
    op_pk_nfd_set( RsvpApplPkptr, "ApplCallType", RELEASE_MSG );
    op_pk_nfd_set( RsvpApplPkptr, "SessionId", &SessionId );
    op_pk_send( RsvpApplPkptr, RSVP_STREAM );
    VL  RADL   printf( "Made a release call to the local RSVP Daemon \n" );

    /** Schedule the interupts **/
    Time = op_dist_outcome( TermDist );
    op_intrpt_schedule_self( op_sim_time()+Time, 0);
}
```

transition **Join -> Idle**			
attribute	*value*	*type*	*default value*
name	tr_4	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline

forced state **Data_Send**			
attribute	*value*	*type*	*default value*
name	Data_Send	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **Data_Send**

```
if(IsSender)
{
   App_Data DataSent;
   Packet* ip_pkptr;
   sprintf( DataSent.Message, "This was data sending to the host %d.%d by %d.%d \n",
          dest_net, dest_node, LocalAddress.NetNum, LocalAddress.NodeNum );
   pkptr = op_pk_create_fmt("Unicast_Appl");

   op_pk_nfd_set(pkptr, "UniData", &DataSent, op_prg_mem_alloc,
             op_prg_mem_free, sizeof(App_Data) );

   data_len = op_pk_total_size_get( pkptr )/8;
   ip_pkptr = op_pk_create_fmt("ip_dgram");
   op_pk_nfd_set(ip_pkptr, "data", pkptr );
   op_pk_nfd_set(ip_pkptr, "prot_id", UDP_PROTOCOL_ID );
   op_pk_bulk_size_set(ip_pkptr, data_len*8 );
   op_pk_nfd_set(ip_pkptr, "dest_net", 500 );
   op_pk_nfd_set(ip_pkptr, "dest_node", grp_id );

   op_pk_nfd_set(ip_pkptr, "src_net", LocalAddress.NetNum );
   op_pk_nfd_set(ip_pkptr, "src_node", LocalAddress.NodeNum );

   op_pk_nfd_set(ip_pkptr, "orig_len", data_len );
   op_pk_nfd_set(ip_pkptr, "frag_len", data_len );
   op_pk_nfd_set(ip_pkptr, "ttl", 2 );
```

```
    op_pk_nfd_set(ip_pkptr, "frag", 0 );

      VL
      {
          printf("Sending data packet from Resv-MCAST Appl at %f\n", op_sim_time() );
    }

    data_send++;
    op_stat_local_write( data_sent, data_send);

    op_pk_send(ip_pkptr, IP_STREAM);
}
op_intrpt_schedule_self( op_sim_time()+(DataIat), 1);
```

transition **Data_Send -> Idle**			
attribute	*value*	*type*	*default value*
name	tr_5	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline

unforced state **EndSim**			
attribute	*value*	*type*	*default value*
name	EndSim	string	st
enter execs	(empty)	textlist	(empty)
exit execs	(empty)	textlist	(empty)
status	unforced	toggle	unforced