Hedge_::Hedge_(Hedge_ *p)

```
{ next=p; adder=0; factor=1; expon= 1; ID= new char[15]; }

//              +++++++++++++++++++++++++++++++++++++++++++++++++++++
```

void Hedge_::Tran(FzSet& bs)

```
{
for(int i=0; i<VECMAX; i++ )
  {
  bs.mu_wrk[i]= (adder+bs.mu[i])*factor;
  if( bs.mu_wrk[i] > 1.0 ) bs.mu_wrk[i]=1.0;
    bs.mu_wrk[i]= pow( bs.mu_wrk[i],expon );
  }
};    // Block: 134

//              +++++++++++++++++++++++++++++++++++++++++++++++++++++
```

istream& operator>>(istream& in, Hedge_& t)

```
{
char tmp[15];
in >> t.ID;
in >> tmp;
if( strcmp( tmp,"add" ) == 0 )
  in >> t.adder;
else if( strcmp( tmp,"multiply" ) == 0 )
  in >> t.factor;
else if( strcmp( tmp,"power" ) == 0 )
  in >> t.expon;
else
  cerr << "Error: (Hedge_ Input) not an input option : " << tmp << "\n" << flush;
                  return in;
};    // Block: 135
//              +++++++++++++++++++++++++++++++++++++++++++++++++++++
```

ostream& operator<<(ostream& out, Hedge_& t)

```
{
out << " " << t.ID << " Add: " << t.adder<< " Prod: "<< t.factor << " Exp: "
<< t.expon << "\n";
return out;
};    // Block: 136
```

```
//              +++++ Fuzzy Issue Control  ( Issue_ ) +++++

   class Issue_

  {
//              ++++++++++++++++++++++++++++++++++++++++++++++++++++
//
// Com 44       Defines Characteristics And Function Of Fuzzy Issues.
//
//              Issue_ consist of blocks of rules associated with an issue.
//              Issues are part of the "FzDecision class" variable list and are
//              elements of a linked list.
//
//              Member functions for rules define into and output of a issues.
//              i.e. Load_issue & Dump_issue
//
//              Issue format is as follows:
//
//              issues risk a1 a2 a3 a4 a5 eoi
//
//              Keyword is "issues " (keyed on in Load_Model of FzDecision)
//              The variable following the keyword is an ID term descriptive of the
//              issue at hand.  The five(5) a1...a5 are the ID's associated with a
//              rule.
//
//              RO_ is the structure inernal to ISSUE_ which contains the set
//              of rules comprising an issue.  Detail usage is discussed in
//              FzDecision.
//              ++++++++++++++++++++++++++++++++++++++++++++++++++++
   public:
   char ID[40];
   Issue_ *next;
   Issue_ *prev;

   struct RO_

  {
           char* r_ID;
           void* add_;
           RO_ *prev;
           RO_ *next;
           RO_( RO_ *p )
           {prev=p; next=0; r_ID= new char[5]; }
  };    // Block: 137

   RO_ *rln;
```

```
 RO_ *rl1;
                    Issue_( Issue_ *p );
 void               Load_issue(istream& in);
 void               Dump_issue(ostream& out);
};      // Block: 138          }
//          +++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Issue_::Issue_(Issue_ *p)

```
 {  prev=p; next=0; rl1=0; rln=0; }
//          +++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

void Issue_::Load_issue(istream& in)

```
{
 char tmp[15];
 RO_ *rl;

 in >> ID;
 in >> tmp;
 while( strcmp( tmp,"eoi" ) != 0 )
   {
   rl= new RO_( rln );
   rln= rl;
   strcpy( rl->r_ID, tmp );
   in >> tmp;
   };

 for( rl =rln; rl->prev !=0; rl =rl->prev )
   (rl->prev)->next =rl;
 rl1= rl;
 return;
};      // Block: 139
//          +++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

void Issue_::Dump_issue(ostream& out)

```
{
 RO_ *rl;
 for( rl=rl1; rl !=0; rl=rl->next )
   out << " " << rl->r_ID << " ";
 return;
};      // Block: 140
```

```
//          +++++ Fuzzy Rule Operations ( Rules_ ) +++++

    class Rules_

  {
//              +++++++++++++++++++++++++++++++++++++++++++++++++++++++
//
// Com 45       Class defines the structure of a rule.
//              Rules_ consist of "rule atoms" RA_ie the parts of a rule
//              Rules which are part of the "FzDecision class" variable list are
//              elements of a linked list.  The rule atoms makeing up a rule are
//              also structured as linked lists.
//
//              Member functions for rules define into and output of a rule sets.
//              i.e. Load_rule & Dump_rule
//
//              Rule format is as follows:
//
//              rule c1 < if SERVCIE_LIFE is LONG then MATERIALS is EXOTIC >
//
//              Keyword input is "rule " (keyed on in Load_Model of FzDecision)
//              follwed by the rule enclosed in angle brackets "<  >"
//              if, is, then, and : are the logic elements of the rule
//
//              +++++++++++++++++++++++++++++++++++++++++++++++++++++++
   public:
   char ID[5];
   int Aggregation_Op ;
   Rules_ *next;
   Rules_ *prev;

    struct RA_

   {
          char *atom;
          RA_ *nm1;
          RA_ *np1;
          RA_( RA_ *p )
            {nm1=p; np1=0; atom= new char[15];}
          friend istream& operator>>(istream& in, RA_& t)
            {
            in >> t.atom;
            return in;
            }
          friend ostream& operator<<(ostream& out, RA_& t)
            {
```

```
        out << " & " << t.atom << " & " ;
        return out;
        }
 };     // Block: 141

 RA_ *at1;
 RA_ *atn;
 RA_ *at;
 Rules_( Rules_ *p );
 void Load_rule(istream& in);
 void Dump_rule(ostream& out);
};     // Block: 142

//            +++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Rules_::Rules_(Rules_ *p)

```
{prev=p; next=0; at1=0; atn=0; Aggregation_Op=MIN_MAX; }

//            +++++++++++++++++++++++++++++++++++++++++++++++++++++
```

void Rules_::Load_rule(istream& in)

```
{
 char tmp[15];
 in >> ID;
 in >> tmp;
 while( strcmp( tmp,">" ) != 0 )
   {
   if( strcmp( tmp,"Op" ) ==0 )
     {
     in >> tmp;
     if( strcmp( tmp,"MIN_MAX" ) )
       Aggregation_Op=MIN_MAX;
     else if( strcmp( tmp,"MIN_MIN" ) )
       Aggregation_Op =MIN_MIN;
     else if( strcmp( tmp,"ADDITIVE" ) )
       Aggregation_Op =ADDITIVE;
     else
       Aggregation_Op=MIN_MAX;

     in >> tmp;
     }
  else if( strcmp( tmp,"<" ) ==0 )
     in >> tmp;
  else if( strcmp( tmp,"is" ) == 0 || strcmp( tmp,"IS" ) == 0 )
```

```
        in >> tmp;
      else
        {
        at= new RA_( atn );
        atn= at;
        strcpy( at->atom, tmp );
        in >> tmp;
        }

      };
//
//                  Next go through and define next atom np1
//
    for( at =atn; at->nm1 !=0; at =at->nm1 )
      (at->nm1)->np1 =at;
    at1= at;
    return;
   } ;     // Block: 143

//                  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

     void Rules_::Dump_rule(ostream& out)

   {
    for( at=at1; at !=0; at=at->np1 )
      out << " " << at->atom << " ";
    return;
   } ;     // Block: 144
```

//   +++++ Fuzzy Sets Operations (FzSet) +++++

class FzSet

```
{
//          ++++++++++++++++++++++++++++++++++++++++++++++++++
//
// Com 46   Define The Characteristics Of Fuzzy Sets
//
//          Fuzzy sets capture the levels associated with a semantic variable.
//          They are stored as linked lists.  Fuzzy sets capture aspects of a
//          semantic or qualitative problem by defining levels to be associated
//          with a variable.  Fuzzy sets are members of the FzVariable class.
//
//          Member functions include:
//
//          A significant number of functions such as; Setup_Beta( ),
//          Setup_Custom( ), Setup_Gauss( ), Setup_Linear( ), Setup_Pi( ),
//          Setup_Sigmoid( ), Setup_Tri( ) ,Setup_Uniform( ) are used to
//          define the shape of the membership function. Note:  not all are
//          defined at this point in time(2/27/98).
//
//          The second block of functions define defuzzification methods;
//          Centroid(),  Ave_Max() & Max_Edge()
//
//          The remaining functions address the following functions:
//
//          Alfacut_Above  sets values associated with an element to zero
//          beyond a specified level.
//          Degree_of_Truth
//          Fz_Initialize simply sets values to the membership function over
//          the range specified in domain.
//          Get_Max
//          Normalize_FZS
//          Reset_Wrk  resets the working vector of membership levels back to
//          the original level.
//
//          Input and output is acheived through the overloaded redirection
//          operators.
//
//          Input format is as follows:
//
//          fuzzy_set
//          HIGH domain 0 50 func_form sigmoid 5 25 45 increase
//          eoi
```

```
//
//              Keyword is "fuzzy_set" ( keyed on in Variable_Setup of FzVariable)
//              The ID for member of the set follows the keyword, "HIGH" in this
//              case.  "domain" defines the range of the variable for the set element,
//              while "func_form" triggers input of the shape of the member
//              element.  In this case the shape is an increasing sigmoid function
//              with low, mid, and high points defined to be 5, 25, & 45.
//              The levels associated with the shape function are stored in
//              the variable  parms
//
//              ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  public:
  int exists;
//
//              Fuzzy Set Descriptor Block (FSDB)
//
  char* ID;                      // FZS id or name
  char* C_typ;                   // FZS curve type
  int conv_trend;                // Converted trend for linear & sigmoid func
  int set_stat;                  // FZS status,set or not set
  int fz_order;                  // FZS order ?
  double domain[2];              // FZS min and max domain values
  double parms[4];
  double alfacut;                // FZS alfacut
  double mu[VECMAX];             // FZS membership array
  double mu_wrk[VECMAX];         // FZS membership working array
  FzSet* next;                   // Linked list, last fuzzy set was...

//
//              overload the input and output operators
//
  friend istream& operator>>( istream& in,FzSet&);
  friend ostream& operator<<( ostream& out, FzSet&);
//
//              Define additional member functions:
//
                FzSet( FzSet* );
  void          Help();
  void          Alfacut_Above( double );
  double        Centroid();
  double        Ave_Max();
  double        Max_Edge();
  double        Degree_of_Truth( double );
  void          Fz_Initialize( );
  double        Get_Max();
  void          Normalize_FZS( );
```

```
  void          Reset_Wrk( );
  void          Setup_Beta( );
  void          Setup_Custom( );
  void          Setup_Gauss( );
  void          Setup_Linear( );
  void          Setup_Pi( );
  void          Setup_Sigmoid( );
  void          Setup_Tri( );
  void          Setup_Uniform( );
 };    // Block: 145

//           ++++++++++++++++++++++++++++++++++++++++++++++++++
//           NOTE:The integers in the constructor must be adjusted to reflect
//           the number of variables and arguments in the model

  FzSet::FzSet( FzSet *p )

 {
  exists=FALSE;
  ID = new char[15];
  C_typ = new char[15];
  set_stat=FALSE;
  next= p;
 };    // Block: 146

//           ++++++++++++++++++++++++++++++++++++++++++++++++++
//           Note:Provide output to the terminal to assist in defining the object
//           model and the information required

  void FzSet::Help( )

 {
  return;
 };    // Block: 147

//           ++++++++++++++++++++++++++++++++++++++++++++++++++

  istream& operator>>( istream& in, FzSet& adum )

 {
  char tmp_in[15];
  char trend[15];

  adum.exists=TRUE;
  in >> adum.ID;
  in >> tmp_in;
  while( strcmp( tmp_in,"eoi" ) !=0 )
```

```
            {
            if( strcmp( tmp_in,"domain" ) == 0 )
              in >> adum.domain[0] >> adum.domain[1];
            else if( strcmp( tmp_in,"alfa_cut" ) == 0 )
              in >> adum.alfacut;
            else if( strcmp( tmp_in,"func_form" ) == 0 )
              {
              in >> adum.C_typ;
              if( strcmp( adum.C_typ,"gauss" ) == 0 )
                in >> adum.parms[0] >> adum.parms[1];
              else if( strcmp( adum.C_typ,"linear" )== 0 )
                {
                in >> adum.parms[0] >> adum.parms[1];
                in >> trend;
                }
              else if( strcmp( adum.C_typ,"sigmoid" )== 0 )
                {
                in >> adum.parms[0] >> adum.parms[1] >> adum.parms[2];
                in >> trend;
                }
              else if( strcmp( adum.C_typ,"triangular" )== 0 )
                in >> adum.parms[0] >> adum.parms[1] >> adum.parms[2];
              else if( strcmp( adum.C_typ,"uniform" )== 0 )
                in >> adum.parms[0] >> adum.parms[1];
              else
                cerr << "Error: (FzSet Input) unaceptable form option : " << tmp_in
                 << "\n" << flush;
              }
            else
              {
              cerr << "Error: (FzSet Input) unaceptable input option : " << tmp_in
               << "\n" << flush;
              }
            in >> tmp_in;
            }
//
//                  Initialize The Fuzzy Set
//
          if( strcmp( adum.C_typ,"linear" )== 0 || strcmp( adum.C_typ,"sigmoid" )== 0 )
            {
            if( strcmp( trend, "increase" )== 0 )
              adum.conv_trend= INCREASE;
            else if( strcmp( trend, "decrease" )== 0 )
              adum.conv_trend= DECREASE;
            else
              cerr << "Error: (FzSet Input) not a trend option : " << trend  << "\n" << flush;
```

```
  }
 adum.Fz_Initialize( );
 return in;
} ;    // Block: 148

//              ++++++++++++++++++++++++++++++++++++++++++++++++++++

  ostream& operator<<( ostream& out, FzSet& adum )

 {
//
//              Provide Coding To Represent The Ouput Desired For This Object
//
  out << adum.ID << "\n";
  double domain_width;
  double member_value;
  int i;
  domain_width= adum.domain[1]-adum.domain[0];
  for( i=0; i<VECMAX; i++)
    {
    member_value = adum.domain[0] + (float)i * domain_width / VECMAX;
    out << " " << member_value << " " << adum.mu[i] << " "
    << adum.mu_wrk[i]
    << "\n";
    }
  return out;
 } ;    // Block: 149
//              ++++++++++++++++++++++++++++++++++++++++++++++++++++
//              Note:Modify fuzzy set with an alfa cut
//

  void FzSet::Alfacut_Above( double cut_val )

 {
  int i;
  for( i=0; i<VECMAX; i++ )
    if( mu[i] < cut_val ) mu[i] = 0.0;
  return;
 } ;    // Block: 150
//              ++++++++++++++++++++++++++++++++++++++++++++++++++++
//              Note:determine the maximum membership value
//

  double FzSet::Centroid( )

 {
  int i;
```

```
double domain_width;
double member_value;
double s1=0.0;
double s2=0.0;
domain_width= domain[1]-domain[0];
for( i=0; i<VECMAX; i++)
  {
  member_value = domain[0] + (float)i * domain_width / VECMAX;
  s1+= mu_wrk[i];
  s2+= mu_wrk[i]*member_value;
  }
if( s1 == 0 ) s1 =999999.9;
return( s2/s1 );
};     // Block: 151
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++
//          Note:determine the maximum membership value
//
```

double FzSet::Ave_Max( )

```
{
int i;
double mx_mem;
mx_mem= Get_Max();
double domain_width;
double member_value;
double s1=0.0;
double s2=0.0;
domain_width= domain[1]-domain[0];
for( i=0; i<VECMAX; i++)
  {
  member_value = domain[0] + (float)i * domain_width / VECMAX;
  if( mu_wrk[i] >= mx_mem )
    {
    s1+= mu_wrk[i];
    s2+= mu_wrk[i]*member_value;
    }
  }
if( s1 == 0 ) s1 =999999.9;
return( s2/s1 );
};     // Block: 152

//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++
//          Note:determine The Maximum Membership Value
//
```

```
double FzSet::Max_Edge( )

{
 int i;
 double mx_mem;
 mx_mem= Get_Max();
 double domain_width;
 double member_value;
 domain_width= domain[1]-domain[0];
 i=0;
 member_value = domain[0];
 while( mu_wrk[i] < mx_mem )
   {
   member_value = domain[0] + (float)i * domain_width / VECMAX;
   i++;
   }
 return( member_value );
};     // Block: 153
//            +++++++++++++++++++++++++++++++++++++++++++++++++++
//            Note:assess The Degree Of Truth Given A Scaler Value
//

double FzSet::Degree_of_Truth( double scaler )

{
 double domain_width;
 double member_value;
 double truth_val;
 int i;
 truth_val= 0.0;
 domain_width= domain[1]-domain[0];
 for( i=0; i<VECMAX; i++)
   {
   member_value = domain[0] + (float)i * domain_width / VECMAX;
   if( member_value > scaler )
   return( truth_val );
   truth_val= mu_wrk[i];
   }
 return( truth_val );
};     // Block: 154
//            +++++++++++++++++++++++++++++++++++++++++++++++++++
//            Note: Determine The Maximum Membership Value
//
```

```
void FzSet::Fz_Initialize()

{
 if( strcmp( C_typ,"gauss" ) == 0 )
   Setup_Gauss();
 else if( strcmp( C_typ,"linear" ) == 0 )
   Setup_Linear();
 else if( strcmp( C_typ,"sigmoid" ) == 0 )
   Setup_Sigmoid();
 else if( strcmp( C_typ,"triangular" ) == 0 )
   Setup_Tri();
 else if( strcmp( C_typ,"uniform" ) == 0 )
   Setup_Uniform();
 else
   cerr << "Error: (FzSet Input) unaceptable form option : " << C_typ  << "\n"
   << flush;
 return;
};     // Block: 155
//            +++++++++++++++++++++++++++++++++++++++++++++++++
//            Note:determine the maximum membership value
//

double FzSet::Get_Max( )

{
 int i;
 double max_mem=0.0;
 for( i=0; i<VECMAX; i++ )
   if( mu_wrk[i] > max_mem ) max_mem = mu_wrk[i];
 return( max_mem );
};     // Block: 156
//            +++++++++++++++++++++++++++++++++++++++++++++++++
//            Note:renormalize the fuzzy set
//

void FzSet::Normalize_FZS( )

{
 int i;
 double norm_val;
 norm_val = FzSet::Get_Max();
 for( i=0; i<VECMAX; i++ )
   mu_wrk[i] = mu[i]/norm_val;
 return;
};     // Block: 157
//            +++++++++++++++++++++++++++++++++++++++++++++++++
```

```
//              Note:renormalize the fuzzy set
//

   void FzSet::Reset_Wrk( )

 {
  int i;
  for( i=0; i<VECMAX; i++ )
    mu_wrk[i] = mu[i];
  return;
 } ;     // Block: 158
//              ++++++++++++++++++++++++++++++++++++++++++++++++++
//              Note:intiialize membership function to a beta function
//

   void FzSet::Setup_Beta( )

 {
  return;
 } ;     // Block: 159
//              ++++++++++++++++++++++++++++++++++++++++++++++++++
//              Note:intiialize membership function to a customized input function
//

   void FzSet::Setup_Custom( )

 {
  return;
 } ;     // Block: 160
//              ++++++++++++++++++++++++++++++++++++++++++++++++++
//              Note:intiialize membership function to a gauss function
//

   void FzSet::Setup_Gauss( )

 {
  double domain_width;
  double thiscaler,gausspt;
  double center, sigma, tmp;
  int i;
  center= parms[0];
  sigma = parms[1];
  tmp = sigma*pow(2*PI,0.5);
  domain_width= domain[1]-domain[0];
  for( i=0; i<VECMAX; i++)
    {
    thiscaler= domain[0] + (float)i * domain_width / (VECMAX-1);
```

```
    gausspt= -pow( ((center-thiscaler)/sigma),2);
    mu[i]= exp( gausspt/2 );
    }
  return;
 };     // Block: 161
//              ++++++++++++++++++++++++++++++++++++++++++++++++
//              Note:intiialize membership function to a linear function;also provides
//              for the capability of puting sholders on the linear functions.
```

void FzSet::Setup_Linear()

```
 {
  double slope_width;
  double domain_width;
  double member_value;
  double lo, hi;
  int i;
  lo= parms[0];
  hi= parms[1];
  slope_width = hi-lo;
  domain_width= domain[1]-domain[0];
  for( i=0; i<VECMAX; i++)
    {
    member_value = domain[0] + (float)i * domain_width / VECMAX;

    if( member_value > hi )
    mu[i]= 1.0;
    else if( member_value > lo && member_value <= hi )
    mu[i]= (member_value - lo) / slope_width;
    else
    mu[i]= 0.0;
    }
  if( conv_trend == DECREASE )
    for( i=0; i<VECMAX; i++ )
      mu[i]= 1.0 - mu[i];
  return;
 };     // Block: 162
//              ++++++++++++++++++++++++++++++++++++++++++++++++
//              Note:intiialize membership function to a pi function
//
```

void FzSet::Setup_Pi( )

```
 {
```

```
 return;
} ;     // Block: 163

//          ++++++++++++++++++++++++++++++++++++++++++++++++++++
//          Note:intiialize membership function to a sigmoid function
//
```

void FzSet::Setup_Sigmoid( )

```
{
 double slope_width;
 double domain_width;
 double member_value;
 double tmp1, left, flexpoint, right;
 int i;

 left= parms[0];
 flexpoint= parms[1];
 right= parms[2];
 slope_width = right-left;
 domain_width= domain[1]-domain[0];

 for( i=0; i<VECMAX; i++ )
   {
   member_value = domain[0] + (double)i * domain_width / VECMAX;
   if( member_value >= right )
     mu[i]=1.0;
   else if( member_value > flexpoint && member_value < right )
     {
     tmp1 = (member_value-right)/slope_width;
     mu[i]= 1-(2*(pow(tmp1,2)));
     }
   else if( member_value >= left && member_value <= flexpoint )
     {
     tmp1 = (member_value-left)/slope_width;
     mu[i]= (2*(pow(tmp1,2)));
     }
   else
     mu[i]=0.0;
   }

 if( conv_trend == DECREASE )
   for( i=0; i<VECMAX; i++ )
     mu[i]= 1.0 - mu[i];
```

```
 return;
 };     // Block: 164
//          ++++++++++++++++++++++++++++++++++++++++++++++++++
//          Note:intiialize membership function to a triangular function
//

  void FzSet::Setup_Tri( )

 {
  double domain_width;
  double member_value;
  double tmp1,tmp2, left, flexpoint, right;
  int i;

  left= parms[0];
  flexpoint= parms[1];
  right= parms[2];
  tmp1 = right-flexpoint;
  tmp2 = flexpoint-left;
  domain_width= domain[1]-domain[0];
  for( i=0; i<VECMAX; i++)
    {
    member_value = domain[0] + (float)i * domain_width / VECMAX;
    if( member_value > left&& member_value <= flexpoint )
      mu[i]= (member_value - left) / tmp2;
    else if( member_value > flexpoint&& member_value < right )
      mu[i]= 1.0 - (member_value - flexpoint) / tmp1;
    else
      mu[i]= 0.0;
    }
  return;
 };     // Block: 165
//          ++++++++++++++++++++++++++++++++++++++++++++++++++
//          Note:intiialize membership function to a uniform distribution function
//

  void FzSet::Setup_Uniform( )

 {
  double domain_width;
  double member_value;
  double lo, hi;
  int i;
  lo= parms[0];
  hi= parms[1];
  domain_width= domain[1]-domain[0];
```

```
for( i=0; i<VECMAX; i++ )
  {
  member_value = domain[0] + (float)i * domain_width / VECMAX;
  mu[i]= 0.0;
  if( member_value <= hi && member_value >= lo )
    mu[i]= 1.0;
  }
 return;
};    // Block: 166
```

// +++++ Fuzzy Variable Operations (FzVariable) +++++

class FzVariable

```
{
//              ++++++++++++++++++++++++++++++++++++++++++++++++
//
// Com 47       Defines The Characteristics of Fuzzy Variables
//
//              Fuzzy variables, FzVariable, are the descriptors for the model
//              being developed. They may represent time, temperature, or
//              service life. Fuzzy variables are typicallly used in the "FzDecisiion
//              class" and are defined as a linked list. Fuzzy variables are
//              associated with fuzzy sets described by the "FxSet class".
//
// Com 48       Member functions include:
//              Find_FzHedge, Find_FzSet which are used to locate
//              characteristics for hedges and fuzzy sets used in the rule model.
//              Reset_FS
//              Variable_Output, & Variable_Setup, which provide functions
//              for loading and printing fuzzy variables.
//
//              Fuzzy variables represent a characteristic of the problem. We
//              normally recognize that the real set of numbers are members of
//              the set temperature we do not think in terms os sets. In the case
//              of fuzzy variables we also must define the space and membership
//              associated with a variable. For this reason we define the
//              levels/descritization of the variable through fuzzy sets, identifying
//              a range of applicability, modifiers or hedges, and in this specific
//              implementation, the values used in the exercise of the system
//              model.
//              Loading of data is controlled or orchastrated by this class.
//
// Com 49       Input format is as follows:
//
//              variables
//              MISSION_PROFILE
//              scaler 6  10 30 50 70 90  70
//              fuzzy_set
//              COMPLEX domain 0 100 func_form linear 0 100 increase
//              eoi
//              fuzzy_set
//              ROUTINE domain 0 100 func_form linear 0 100 decrease
//              eoi
//              hedge
```

```
//              VERY power 2.0
//              eoi
//
// Com 50       Keyword is "variables" (keyed on in Load_Model of FzDecision)
//              In this case "MISSION_PROFILE" is the variable name.  "scaler"
//              is a local input keyword indicating that 6 test runs will be conducted
//              and the quantitative values to be assigned to MISSION_PROFILE
//              will be, 10 30 50 70 90  70.
//
// Com 51       The next local keyword, "fuzzy_set" triggers input of set information
//              and is described in FxSet class headers.  Suffice to say COMPLEX
//              and ROUTINE represent linguistic levels associated with the
//              variable "MISSION_PROFILE".
//
// Com 52       The last local keyword is "hedge" which triggers input of hedge
//              type information.  Hedges act as modifiers in the semantic fuzzy
//              model being developed.  In this case the modifier allows us to
//              examine linguistic rules associated with
//              VERY COMPLEX MISSION_PROFILE s.
//
//              +++++++++++++++++++++++++++++++++++++++++++++++++++++
  public:
//
//              Define model parameters:
//
  int result_var;
  int result_;
  char* Fz_Var_ID;
  int num_scaler;
  double* scaler;
  double* D_o_T;

  FzVariable* next;
  FzSet* fs;
  FzSet* last_fs;
  FzSet* fs_result;

  Hedge_*hedge;
  Hedge_*last_hedge;
//
//              Define addiitonal member functions:
//
                FzVariable( FzVariable* );
  Hedge_*       Find_FzHedge( char* );
  FzSet*        Find_FzSet( char* );
  void          Help();
```

```
  void          Reset_FS( );
  void          Variable_Output( ostream& out );
  void          Variable_Setup( istream& in );
 };    // Block: 167

//          ++++++++++++++++++++++++++++++++++++++++++++++++
//          NOTE:The integers in the constructor must be adjusted to reflect
//          the number of variables and arguments in the model
//
```

FzVariable::FzVariable(FzVariable *p)

```
 {
//          Define default conditions
//
  result_var=FALSE;
  result_ =FALSE;
  Fz_Var_ID = new char[25];
  last_fs=0;
  last_hedge= 0;
  next= p;
  scaler= 0;
  D_o_T= 0;
//
//          Setup A Result Fuzzy Set For Use In The De-fuzzification Process
//
  fs = new FzSet( last_fs );
  last_fs= fs;
  fs_result= fs;
  strcpy( fs->ID, "RESULT_" );
 };    // Block: 168
//          ++++++++++++++++++++++++++++++++++++++++++++++++
```

Hedge_* FzVariable::Find_FzHedge(char* v_nam)

```
 {
  Hedge_* rtn_v;
  rtn_v=0;
  hedge=last_hedge;
  while( hedge!=0 )
    {
    if( strcmp( hedge->ID, v_nam ) == 0 )
      rtn_v=hedge;
    hedge=hedge->next;
    }
```

```
  return( rtn_v );
 };    // Block: 169
//          ++++++++++++++++++++++++++++++++++++++++++++++++
```

FzSet* FzVariable::Find_FzSet(char* v_nam)

```
 {
  FzSet* rtn_v;
  rtn_v=0;
  fs=last_fs;
    while( fs!=0 )
    {
    if( strcmp( fs->ID, v_nam ) == 0 )
    rtn_v=fs;
    fs=fs->next;
    }
  return( rtn_v );
 };    // Block: 170
//          ++++++++++++++++++++++++++++++++++++++++++++++++
//          Note:Provide output to the terminal to assist in defining the object
//          model and the information required
```

void FzVariable::Help( )

```
 {
  return;
 };    // Block: 171
//          ++++++++++++++++++++++++++++++++++++++++++++++++
```

void FzVariable::Reset_FS( )

```
 {
  fs=last_fs;
  while( fs!=0 )
    {
    fs->Reset_Wrk();
    fs=fs->next;
    }
  return;
 };    // Block: 172
//          ++++++++++++++++++++++++++++++++++++++++++++++++
```

void FzVariable::Variable_Output(ostream& out)

```
 {
 out << "\n Fuzzy Variable:" << Fz_Var_ID;
 for( fs=last_fs; fs != 0; fs=fs->next )
```

```
    {
    out << "\nFuzzy set: ";
    out << *fs;
    }

  out << "\n Fuzzy Variable:" << Fz_Var_ID << "\n";
  for( int i=0; i<num_scaler; i++ )
  out << " " << Fz_Var_ID << "[ " << i+1 << " ] " << scaler[i] << "\n";

  out << "\n Fuzzy Variable:" << Fz_Var_ID;
  for( hedge=last_hedge; hedge != 0; hedge=hedge->next )
    {
    out << "\n Fuzzy Hedges: ";
    out << *hedge;
    }
  return;
 };     // Block: 173
//              ++++++++++++++++++++++++++++++++++++++++++++++++++
```

void FzVariable::Variable_Setup(istream& in)

```
 {
  char tmp_in[15];
  in >> Fz_Var_ID;
  in >> tmp_in;
  while( strcmp( tmp_in,"eoi" ) !=0 )
    {
    if( strcmp( tmp_in,"scaler" ) == 0 )
      {
      in >> num_scaler;
      scaler = new double[num_scaler];
      D_o_T= new double[num_scaler];
      int icas=0;
      in >> tmp_in;
      while( strcmp( tmp_in,"eoi" ) !=0 )
        {
        scaler[icas]= double( atof(tmp_in) ) ;
        icas++;
        in >> tmp_in;
        if( icas > num_scaler )
          cerr << "Error: (Scaler Input) too many values : " << icas << "\n" << flush;
        }
      }
    else if( strcmp( tmp_in,"fuzzy_set" ) == 0 )
      {
      fs = new FzSet( last_fs );
```

```
  in >> *fs;
  last_fs= fs;
  }
 else if( strcmp( tmp_in,"hedge" ) == 0 )
  {
  hedge = new Hedge_( last_hedge );
  in >> *hedge;
  last_hedge= hedge;
  }
 else
  {
  cerr << "Error: (FzVariable Input) unaceptable input option : " << tmp_in
   << "\n" << flush;
  }
 in >> tmp_in;
 }
return;
};     // Block: 174
```

DISTRIBUTION:

1 Michael Maglich, SP282
Strategic Systems Programs
1931 Jefferson Davis Hwy.
Arlington, Va 22241-5362

1 Mike Wagner
ITTS
1500 Garden of the Gods Rd.
PO Box 7463
Colorado Springs, CO 80933

1 Dwayne Curtiss
ITTS
1111 Jefferson Davis Hwy.
Suite 700
Arlington, VA 22202

1 Rick Verbanec
Lockheed-Martin Missiles & Space
PO Box 3504
89-20/157
Sunnyvale, CA 94089-3504

1 Alex Loewenthal
Dept. 2514, Bldg. 611
Plant 10
Lockheed Martin Skunkworks
1011 Lockheed Way
Palmdale, CA 93599-2514

1 Wallace Louie
NSWC DL
Code K-44
17320 Dahlgren Rd.
Dahlgren, VA 22448

1 Doug P. Anson
LANL
TSA-5 MS F602
Los Alamos, NM 87545

1 Dr. C. Christopher Reed
The Aerospace Corporation
PO Box 92957- M4/943
Los Angeles, CA 90009-2957

1	MS0457	W.J. Tedeschi, 2001
1	MS0429	J.H. Stichman, 2100
1	MS0475	R.C. Hartwig, 2105
1	MS0475	J.B. Godfrey, 2105
2	MS0481	T.F. Hendrickson, 2167
1	MS9005	J.B. Wright, 2200
2	MS0501	M.K. Lau, 2338
	Attn:	T.E. Owen, 2338
1	MS1221	J.S. Rottler, 5400
1	MS1221	A.B. Cox, 5405
1	MS0417	W.H. Ling, 5413
	Attn:	D.M. Fordham, 5413
10	MS0455	M.E. Senglaub, 6232
1	MS9018	Central Technical Files, 8940-2
2	MS0899	Technical Library, 4916
1	MS0619	Review & Approval Desk, 15102 For Doe/OSTI