ModEco V2.xxα
Absolutely Conservative ModEco-Based Economy
The Cyclic Economic Engine
Documentation of The Initialization and Transition Rule

Author: G. H. Boyle
Date: 10th March, 2014

TABLE OF CONTENTS

ModEco V2.xxα
Absolutely Conservative ModEco-Based Economy
The Cyclic Economic Engine
Documentation of The Initialization and Transition Rule

The Economic Engine

References:

[1] Boyle, G H; Larson, I; Liu, S; The Distribution of Wealth In an Absolutely Conservative Economy in ModEco, 27 April 2011
[2] Boyle, G H; Note To File; Conservative Economy – Transient Vs Steady-state Behaviour; May 16th, 2011
[3] Boyle, G H; Note To File; Conservative Economy – Distribution of Wealth; May 16th, 2011
[4] ACE website, http://www2.econ.iastate.edu/tesfatsi/ace.htm, 16 June, 2011

Scenario:

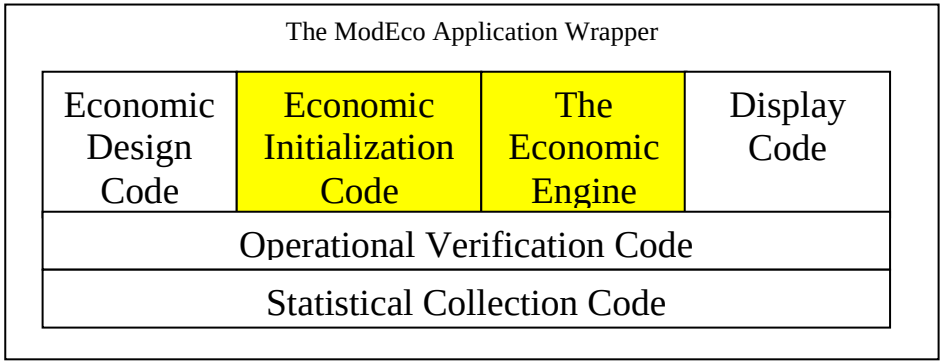
ModEco V2.xxA, Cyclic Engine, Crowded Model, Seed=1, PMM (Perpetual Motion Machine) On (Pmm=On)
This scenario exhibits an absolutely conservative sustainable economy, as defined at [1], that has achieved a run of over 20 million ticks prior to user termination.

Purpose:

- To describe the algorithm used in a ModEco-based economy, by reference to cropped source code from which all non-algorithmic code has been elided.

Preliminary Information:

ModEco is a windows application which provides a laboratory in which a user may design and run different economies, watch real-time developments, and collect data for later analysis with an analytical package of some kind such as MS Excel. There are over 150,000 lines of code, most of which are required to make a windows application run, or to enabled user control of the design of the economy. This note is only interested in a subset of that code. The following diagram characterises types of code that are, or are not, of interest:



The economic design code gives the user the ability to change certain parameters. As the ModEco application was developed and tested certain features were added to enable studies of behaviour that lead to further improvements, and the addition of further features. In the present version, V2.XXA, there are some vestigial parameters that the user has access to which have no current value, and there are a few that are totally untested. The A in the version number means the software is in the 'alpha-test' phase, and so has lots of warts and missing bits. However, the design code does give the user substantial ability to enable or disable features, or vary parametric values. Most of this code is associated with the ModEco Initiation Wizard, Economic Wizard, the random number seed assignment, or the selection of township size through the scenario command. None of that is the topic of this note.

The display code is of two types: real-time display, and data export. The display code has been carefully separated from the economic engine. No display data feeds back into the engine. The display extracts data from the engine and displays it in real time or exports it to a file for later study. The display code is normally intermingled with the code for the economic engine, but it plays no role in the algorithm and has been removed in the code shown below.

The verification code is used to ensure that the economic is performing as intended. It is used in debug mode only, and is not part of the release version of the software. However, the code shown below originally included the verification code intermingled with the algorithmic code. The data collection code is also intermingled with the economic engine code. The verification code and the statistical collection code is not the topic of this note, and has been removed in the code shown below.

The economic initialization is important because a ModEco economy is essentially a complex FSM, and so it must be specified as a set of initial conditions, and a transition rule. The initialization code specifies the initial conditions for the FSM that is a ModEco-based economy. This is part of the topic of this note.

The economic engine itself embodies the transition rule for the FSM that is a ModEco-based economy. This is also part of the topic of this note.

Definitions:

For a relatively detailed description of the ModEco scenario of interest here, see [1]. Here are a few key definitions:

- **Township:** the rectangular area of green space on the screen in which the economy develops. It is formed of square lots which tile the area.
- **MMgr:** the materiel manager, a unique agent of the township who buys scrap and sells raw materials.
- **EMgr:** the estate manager, a unique agent of the township who receives estate assets and doles out municipal grants to deserving agents.
- **Agent:** an independent logical entity that buys, sells, produces, consumes, receives or disburses assets. Types of agents are Frmrs, Wrkrs, the MMgr, the EMgr.
- **Frmr:** a producer of goods, buying raw materiel, hiring workers, producing inventory for sale to consumers.
- **Wrkr:** a worker, selling labour, working to produce inventory for the employer.
- **Consumer:** A Frmr or Wrkr, buying inventory from Frmrs and storing it in its supply cupboard, consuming supplies, selling scrap, using/selling obtained vigour to produce more inventory.
- **Materiel units:** An MbMEu is composed of one MbMz and one MbEu. It represents a produced good. The MbMz represents the matter. The MbEu represents the added value that accrues from work done by a Frmr and the worker hired by the Frmr.
- **Conserved quantities:** four quantities are conserved in all ModEco-based economies. Cash (measured in dollars), intrinsic value (measured in dollars), matter (measured in MbMu) and infrastructure/health (measured in MbEu) are conserved.
  - **Cash:** is the medium of exchange, all commercial transactions involve an exchange of cash for goods or services.
  - **Intrinsic Value:** all cash and all types of materiel units have an intrinsic value which is conserved in all transactions. The intrinsic value draws from the nature and needs of human life, and the need for access to certain substances.
  - **Matter:** resource-based materiel units (MbMu), a proxy for physical mass.
  - **Infrastructure/health:** work-based materiel units (MbEu) represent added value, and is a proxy for physical energy.
- **Stores:** types of intrinsic value held by an agent, such as cash ($), raw materiel (MbMu), infrastructure (MbEu), health (MbEu), inventory (MbMEu), supplies (MbMEu), scrap (MbMu).
- Unit price: all goods and services are sold at their intrinsic value. One MbMz is worth $2. One MbEu is worth $8. One MbMEu is worth $10.

Verification and Validation

According to the website of the Agent-based Computational Economics (ACE) special interest group of the Society for Computational Economics (SCE), verification and validation involve two significantly different activities:

- Verification: measures taken to ensure that the model performs as the designers intended.
- Validation: measures taken to ensure that the model correctly models the real world.

Validation

ModEco has not been validated and it is not intended that it be validated. ModEco was designed to abstract from reality the most basic characteristics of a commercial economy in an effort to understand the nature of a sustainable commercial economy, and determine what makes it sustainable. As such, it is intended that it be treated as a valid demonstration economy in its own right, and studied as such. Should lessons learned from the study of a ModEco economy be applicable in a real-world economy, that would be a good, and hoped-for, benefit. Such a pleasant happenstance would be some kind of validation of the effort taken to build it. But it is not reasonable or useful to look at some feature of ModEco, decide that it does not model reality correctly, and declare ModEco invalid.

Verification

It is intended that ModEco be verified, and verifiable by users. All of the debugging tools developed during the creation of ModEco have been left active for the user. Some are quite technical, and not well-documented at this point, but they are nevertheless available, as follows:

- **ModEco Control Panel:** available under the TechStuff menu item, this command starts a dialogue in which the user can inspect the setting of all parameters under user control, and all features under user control. This method of validation is only available before the first tick of the economy is taken.
- **View Agents:** available under the TechStuff menu item, this command starts a dialogue in which the user can inspect the content of all of the stores of each agent, as well as a number of other technical characteristics.
- View Agents Genes: available under the TechStuff menu item, this command starts a dialogue in which the user can inspect the value of all pricing genes. The pricing genes play no role in the sustainable economy being studied in this exercise, and will not be explained further here.
- **View Contact Pool:** available under the TechStuff menu item, this command starts a dialogue in which the user can inspect the details of all contact lists such as the union list, employer list, customer list, supplier list of each agent, or the common pool in which all of these logical rolodex files exist.
- **Left, Right, Status and Notes panels:** Available via the black buttons on the toolbar, allowing the user to have real-time visual presentation of macro-economic and micro-economic data.
- **CSV Exports:** available under the Excel Export command, there are fifteen different file formats that can be dumped to file. CSV stands for comma-separated values. These files provide exceptional and redundant detail on every type of significant transaction. Most files have been used extensively to debug the economic engine, and to reconcile expected results with actual results.

ModEco contains approximately 150,000 lines of code. This includes over 1,600 ASSERT statements. An ASSERT is like a mathematical tautology. If a variable x should never be negative, each time it is calculated the programmer can enter an ASSERT statement like this:

```
ASSERT( X >= 0.0 );
```

When the software is run in debug mode, any failure to match such an ASSERT will cause the program to halt and forces the programmer to find and fix the fault. Liberal usage of ASSERT statements ensures that all variables stay within their intended domain.

ModEco has been run in debug mode up to 20,000,000 ticks without any ASSERTs happening. The economy was sustainable, and no abnormalities were seen.

There are still a number of minor bugs in the display code, and these are going on a list for later correction. There are no known bugs in the economic engine, or in the initializaiton code.

There is no guarantee that the economic engine is without errors. Given the large number of lines of code, and the complexity of some of the sub-algorithms, there is some reasonable probability that there are still bugs in the economic engine. One area of concern is the potential for edge effects to cause faults. This will be reviewed in some detail and addressed in a later version of ModEco.

The problem of edge effects deserves some explanation. In a computer system, every variable has an upper and lower cutoff, and a resolution cutoff. For example, a computer may not be able to represent a number larger than $+10^{50}$ or smaller than -10^{50}. These are the upper and lower cutoffs. It also may not be able to handle numbers of magnitude less than $+/- 10^{-50}$. This is a resolution cutoff. These cutoff values determine the range and density of values that a variable can have. If a dollar variable is rounded to the nearest penny, that restricts the resolution of the variable, and is an edge effect. Every variable has edge effects. Since there are thousands of variables in a ModEco economy, there are thousands of edges that can cause faults. Happily, many of them are easy to manage.

For example, the township has a width and a height. The variable x that specifies the horizontal position of a lot cannot access a position outside of the township. The left and right edges of the township need to be observed.

The most difficult edge effect to manage is rounding of variables which must be conserved in aggregate. All dollar values, and intrinsic values, are rounded to two digits after the decimal. All MbMEu, MbMu and MbEu are rounded to three digits after the decimal. Once every tick, discrepancies are found and some control variables are adjusted to ensure that conserved quantities suffer no slippage.

In short, every effort has been made to ensure that the economic engine has been verified and is functioning as intended.

The Initialization Process

In the following presentation, the code is not executable. All declarations, ASSERTions, and verification code has been removed. Most non-relevant return codes have been removed. In addition, all code associated with display, status reporting or statistical data collection has been removed. Finally, all code that handles scenarios and switchable features other than those used in the scenario of interest has been deleted. What remains is the code that defines the transition rule of the FSM of interest.

The program that initializes the absolutely conservative (sustainable) economy is DoScenario() when invoked with the level 1, scenario 2 code number. It, in turn, invokes the proper scenario builder. The following list of programs are involved in initialization of the scenario of interest.

Initialization Code Function Names

```
bool DoScenario( long ScenarioNo );
bool DoL1CrowdedModel( void );
bool BuildFarm( long Col, long Row );
bool BuildResidence( long Col, long Row, long Number );
bool StaggerAges( void );
bool BuildUnionList( BFrmr* FrmrPtr );
bool BuildCustomerList( BFrmr* FrmrPtr );
bool AddThisWrkrToUnionLists( BWrkr* ThisWrkrPtr );
bool AddThisWrkrToCustomerLists( BWrkr* ThisWrkrPtr );
bool AddThisFrmrToEmployerLists( BFrmr* ThisFrmrPtr );
bool AddThisFrmrToSupplierLists( BFrmr* ThisFrmrPtr );
bool AddThisFrmrToCustomerLists( BFrmr* ThisFrmrPtr );
```

bool BModEcoSys::DoScenario(long newScenarioNo)

```
{
```

```
    switch( newScenarioNo )
    {
    case _ME_ModelNo_LEVEL1_02:
        TempViewPtr = ViewPtr;
        Toggles.SetTwpHeight( 15 );
        Toggles.SetTwpWidth( 20 );
        InitBModEcoSys( RsPtr );
        ScenarioNo = _ME_ModelNo_LEVEL1_02;
        ViewPtr = TempViewPtr;
        Title = "ModEco1B";
        ViewPtr->GetDocument()->SetTitle( Title );
        RegKeys.SetTitle( &Title );
        NextAgentSerNo = 0;
        CurrentFunction = _ME_Fn_Setup;
        Toggles.VersionNo = _ME_VersionNo;
        Toggles.LevelNo = 1;
        Toggles.ScenarioNo = _ME_ModelNo_LEVEL1_02;
        Toggles.SeedNo = RsPtr->GetCurrentSeed();
        DoL1CrowdedModel();
        break;
    }
}
```

bool BModEcoSys::DoL1CrowdedModel(void)

```
{
    NwFactor = 5.0;
    Township.DistributeNetWorth( Toggles.GetAppropriateMMgr_Nwpa(), Toggles.Global_Pm_Switch );
    Township.MMgr.ZeroMMgrGenes( &ERules );
    for( X = 0; X < 20; X = X + 5 )
    {
        for( Y = 0; Y < 15; Y = Y + 3 )
        {
            BuildResidence( X + 0, Y + 0, _ME_HiresMax_PostInd );
            BuildResidence( X + 0, Y + 1, _ME_HiresMax_PostInd );
            BuildFarm( X + 1, Y + 0 );
            BuildFarm( X + 1, Y + 1 );
            BuildFarm( X + 2, Y + 0 );
            BuildFarm( X + 2, Y + 1 );
            BuildResidence( X + 3, Y + 0, _ME_HiresMax_PostInd );
            BuildResidence( X + 3, Y + 1, _ME_HiresMax_PostInd );
        }
    }
```

```
    StaggerAges();

    // Baseline the Macro-Economic Aggregator.
    Aggregator.ZeroCurAggregates();
    Aggregator.ShiftAggregates();
    Aggregator.ComputeAggregateDifs();
    Aggregator.CollectAggregates( &FrmrList, &WrkrList, &Township );
    Aggregator.TakeBaselineReading();
    Aggregator.ForceToBaseline( &Township );

    BuildLists();
}
```

```
bool BModEcoSys::BuildFarm( long Col, long Row )
{
        MuType = _ME_LotDevType_Farm;

        LotSlotNo = Township.GetLotSlotNo( Col, Row );
        TwpLotPtr = Township.GetLotPtr( LotSlotNo );
        TwpLotPtr->SetLotDevType( MuType );
        TwpLotPtr->Location.SetPLocation( CPoint( Col, Row ) );
        TwpLotPtr->ComputeLotBaseColors();

        FrmrPtr = FrmrList.AddFrmr( IncNextAgentSerNo() );
        FrmrPtr->TogglesPtr = &Toggles;
        FrmrPtr->TransmitRsPtr( RsPtr );
        FrmrPtr->UnionList.PoolPtr = (&ContactPool);
        FrmrPtr->CustomerList.PoolPtr = (&ContactPool);
        FrmrPtr->SupplierList.PoolPtr = (&ContactPool);
        FrmrPtr->BirthDate = Township.Age;
        FrmrPtr->Generation = 1;

        FrmrPtr->FrmrType = MuType;
        FrmrPtr->ComputeFrmrsOwnColor();

        FrmrPtr->EconomicStatus = _ME_EconomicStatus_PostInd;
        FrmrPtr->NoOfHiresMax = _ME_HiresMax_PostInd;
        FrmrPtr->NoOfHires = 0;

        // Initialize the raw mu pool.
        FrmrPtr->RecycledMbMzOnHand.MuPool[ MuType ].SetMuType( MuType );
        FrmrPtr->DistributeNetWorth( Toggles.GetAppropriateFrmr_Nwpa() );
```

```
	// Now, initialize the Supply mu pool.
	// Not req'd.

	LotVdPtr = (void*) TwpLotPtr;
	FrmrVdPtr = (void*) FrmrPtr;
	FrmrPtr->LotPtrPair.PtrPairActiveFlag = _ME_PtrPairActiveFlag_Yes;
	FrmrPtr->LotPtrPair.ObjectVdPtr = LotVdPtr;
	FrmrPtr->LotPtrPair.ObjectSlotNo = LotSlotNo;
	FrmrPtr->LotPtrPair.AddrSlotNo = -1;    // Not in a slot.
	TwpLotPtr->SetFrmrPtrPair( FrmrPtr->SlotNo, FrmrVdPtr, _ME_PtrPairActiveFlag_Yes );
	TwpLotPtr->Location.SetWireFrame();
	IncludeInSums( FrmrPtr );
}
```

bool BModEcoSys::BuildResidence(long Col, long Row, long Number)

```
{
	LotSlotNo = Township.GetLotSlotNo( Col, Row );
	TwpLotPtr = Township.GetLotPtr( LotSlotNo );
	TwpLotPtr->SetLotDevType( _ME_LotDevType_Residence );
	TwpLotPtr->Location.SetPLocation( CPoint( Col, Row ) );
	TwpLotPtr->ComputeLotBaseColors();
	LotVdPtr = (void*) TwpLotPtr;

	for( WrkrNo = 0; WrkrNo < Number; WrkrNo++ )
	{
		WrkrPtr = WrkrList.AddWrkr( IncNextAgentSerNo() );
		WrkrVdPtr = (void*) WrkrPtr;
		// Serial number and dynasty assigned by AddWrkr.
		WrkrPtr->TogglesPtr = &Toggles;
		WrkrPtr->TransmitRsPtr( RsPtr );
		WrkrPtr->EmployerList.PoolPtr = (&ContactPool);
		WrkrPtr->SupplierList.PoolPtr = (&ContactPool);
		WrkrPtr->BirthDate = Township.Age;
		WrkrPtr->Generation = 1;

		WrkrPtr->ComputeWrkrsOwnColor();
		WrkrPtr->DrawingColor = WrkrPtr->WrkrsOwnColor;

		//WrkrPtr->CashOnHand = _ME_CashOnHand_Dflt;
		WrkrPtr->Age.InitSAge();
		WrkrPtr->DateLastHired = Township.Age.GetInSeconds() - 1;

		AddResident( TwpLotPtr, WrkrPtr );
```

```
            WrkrPtr->ReproStatus = 0;
            WrkrPtr->DistributeNetWorth( Toggles.GetAppropriateWrkr_Nwpa() );
            IncludeInSums( WrkrPtr );
      }
}
```

bool BModEcoSys::StaggerAges(void)

```
{
      // Establish a random order for processing the Frmrs.
      NoOfFrmrs = FrmrList.GetNoOfFrmrs();
      DeltaAge = (double) ( ( (double) FrmrPtr->Genes.ART ) / ( (double) NoOfFrmrs ) );
      Age = 0; DoubleAge = 0;
      for( FrmrNo = 0; FrmrNo < NoOfFrmrs; FrmrNo++ )
      {
            FrmrPtr = FrmrList.GetFrmrPtr( SlotNoList.GetRandomSlotNo() );

            ExcludeFromSums( FrmrPtr );
            FrmrPtr->Age.SetInSeconds( Age );
            IncludeInSums( FrmrPtr );

            DoubleAge += DeltaAge;
            Age = (long) DoubleAge;

            FrmrPtr = FrmrPtr->NextPtr;
      }

      NoOfWrkrs = WrkrList.GetNoOfWrkrs();
      DeltaAge = (double) ( ( (double) WrkrPtr->Genes.ART ) / ( (double) NoOfWrkrs ) );
      Age = 0; DoubleAge = 0;
      for( WrkrNo = 0; WrkrNo < NoOfWrkrs; WrkrNo++ )
      {
            WrkrPtr = WrkrList.GetWrkrPtr( SlotNoList.GetRandomSlotNo() );

            ExcludeFromSums( WrkrPtr );
            WrkrPtr->Age.SetInSeconds( Age );
            IncludeInSums( WrkrPtr );

            DoubleAge += DeltaAge;
            Age = (long) DoubleAge;

            WrkrPtr = WrkrPtr->NextPtr;
      }
}
```

```
bool BModEcoSys::BuildUnionList( BFrmr* FrmrPtr )
{
   // Check if UnionList is already in existence.
   if( FrmrPtr->UnionList.GetNoOfContacts() > 0 ) return TRUE;

   // For this Frmr, canvass all Wrkrs in the
   //   commuting area and add to union list.
   TwpLotPtr = (BTwpLot*) FrmrPtr->GetLotVdPtr();
   CommuteAreaPtr = &(TwpLotPtr->CommuteArea);
   CaWidth = CommuteAreaPtr->CaWidth;
   CaHeight = CaWidth;
   for( CaRow = 0; CaRow < CaHeight; CaRow++ )
   {
      for( CaCol = 0; CaCol < CaWidth; CaCol++ )
      {
         TrgtTwpLotPtr = (BTwpLot*) CommuteAreaPtr->GetTwpLotPtr( CaCol, CaRow );
         // If this place is not residential, add no contacts.
         if( TrgtTwpLotPtr->IsResidential() == TRUE )
         {
            // Residential township lots must have at least one worker.
            NoOfResidents = TrgtTwpLotPtr->ResidentList.GetNoOfResidents();
            for( ResNo = 0; ResNo < _ME_ResSlotsMax; ResNo++ )
            {
               long   PtrPairActiveFlag;
               PtrPairActiveFlag = TrgtTwpLotPtr->ResidentList.GetResidentPtr( ResNo )->PtrPairActiveFlag;
               if( PtrPairActiveFlag == _ME_PtrPairActiveFlag_Yes )
               {
                  WrkrVdPtr = TrgtTwpLotPtr->ResidentList.GetWrkrVdPtr( ResNo );
                  WrkrPtr = (BWrkr*) WrkrVdPtr;
                  WrkrsLotVdPtr = (void*) WrkrPtr->LotPtrPair.ObjectVdPtr;
                  FrmrVdPtr = (void*) FrmrPtr;
                  FrmrsLotVdPtr = (void*) TwpLotPtr;
                  FrmrMtchPtr = FrmrPtr->UnionList.LinkNewWrkrAtBottom( _ME_CI_AgentType_Frmr,
                                                                        FrmrPtr->SlotNo,
                                                                        FrmrVdPtr,
                                                                        WrkrPtr->SlotNo,
                                                                        WrkrVdPtr,
                                                                        TrgtTwpLotPtr->LotSlotNo,
                                                                        WrkrsLotVdPtr );
                  if( FrmrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
                  // Add this Frmr to the Wrkr's employer list.
                  WrkrMtchPtr = WrkrPtr->EmployerList.LinkNewFrmrAtBottom( _ME_CI_AgentType_Wrkr,
                                                                           WrkrPtr->SlotNo,
```

```
                                                                WrkrVdPtr,
                                                                FrmrPtr->SlotNo,
                                                                FrmrVdPtr,
                                                                TwpLotPtr->LotSlotNo,
                                                                FrmrsLotVdPtr );
                        if( WrkrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
                        FrmrMtchPtr->MtchPtr = WrkrMtchPtr;
                        FrmrMtchPtr->MtchSlotNo = WrkrMtchPtr->SlotNo;
                        WrkrMtchPtr->MtchPtr = FrmrMtchPtr;
                        WrkrMtchPtr->MtchSlotNo = FrmrMtchPtr->SlotNo;
                     }  // End of if PtrPairActiveFlag.
                  }  // End of for ResNo.
               }  // End of if IsResidential.
            }  // End of for CaCol.
         }  // End of for CaRow.
      }
```

bool BModEcoSys::RemoveThisWrkrFromUnionLists(BWrkr* ThisWrkrPtr)

```
{
   // Go through the list of employers within the old commuting
   //   area and delete ThisWrkr from each union list.
   NoOfEmployers = ThisWrkrPtr->EmployerList.GetNoOfContacts();
   ContactPtr = ThisWrkrPtr->EmployerList.GetTopContactPtr();
   for( EmployerNo = 0; EmployerNo < NoOfEmployers; EmployerNo++ )
   {
      // Get a pointer to the Frmr from the employer contact list.
      FrmrPtr = (BFrmr*) (ContactPtr->AgentVdPtr);
      // Get a pointer to the matching BContact item in the
      //   employer's union list.
      DelThisWrkrContactPtr = ContactPtr->MtchPtr;
      DelThisFrmrContactPtr = ContactPtr;
      // Step the ContactPtr back one in the employer list.
      ContactPtr = ContactPtr->PrevPtr;

      // Delete the Wrkr from the union list.
      Result = FrmrPtr->UnionList.DelContact( DelThisWrkrContactPtr );
      // Delete the contact.
      Result = ThisWrkrPtr->EmployerList.DelContact( DelThisFrmrContactPtr );
   }
}
```

bool BModEcoSys::AddThisWrkrToUnionLists(BWrkr* ThisWrkrPtr)

```
{
```

```
// Package Wrkr data for use.
WrkrVdPtr = (void*) ThisWrkrPtr;
TwpLotPtr = (BTwpLot*) ThisWrkrPtr->LotPtrPair.ObjectVdPtr;
WrkrsLotVdPtr = (void*) TwpLotPtr;

// ThisWrkr's new TwpLotPtr is already set.  We need to go
//   through the commuting area and add ThisWrkr to the
//   union list of each Frmr in the commuting area.
CaPtr = &TwpLotPtr->CommuteArea;
CaWidth = CaPtr->GetCaWidth();
for( CaRow = 0; CaRow < CaWidth; CaRow++ )
{
   for( CaCol = 0; CaCol < CaWidth; CaCol++ )
   {
      TrgtTwpLotPtr = (BTwpLot*) CaPtr->GetTwpLotPtr( CaCol, CaRow );
      LotDevType = TrgtTwpLotPtr->GetLotDevType();
      if( TrgtTwpLotPtr->IsCommercial() == TRUE )
      {
         FrmrPtr = (BFrmr*) TrgtTwpLotPtr->GetFrmrVdPtr();
         FrmrVdPtr = (void*) FrmrPtr;
         FrmrsLotVdPtr = (void*) TrgtTwpLotPtr;
         // This is a potential employer.  Add the Wrkr to the union list.
         FrmrMtchPtr = FrmrPtr->UnionList.LinkNewWrkrAtBottom( _ME_CI_AgentType_Frmr,
                                                FrmrPtr->SlotNo,
                                                FrmrVdPtr,
                                                ThisWrkrPtr->SlotNo,
                                                WrkrVdPtr,
                                                TwpLotPtr->LotSlotNo,
                                                WrkrsLotVdPtr );
         if( FrmrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
         // Add this Frmr to the Wrkr's employer list.
         WrkrMtchPtr = ThisWrkrPtr->EmployerList.LinkNewFrmrAtBottom( _ME_CI_AgentType_Wrkr,
                                                      ThisWrkrPtr->SlotNo,
                                                      WrkrVdPtr,
                                                      FrmrPtr->SlotNo,
                                                      FrmrVdPtr,
                                                      TrgtTwpLotPtr->LotSlotNo,
                                                      FrmrsLotVdPtr );
         if( WrkrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
         FrmrMtchPtr->MtchPtr = WrkrMtchPtr;
         FrmrMtchPtr->MtchSlotNo = WrkrMtchPtr->SlotNo;
         WrkrMtchPtr->MtchPtr = FrmrMtchPtr;
         WrkrMtchPtr->MtchSlotNo = FrmrMtchPtr->SlotNo;
      }
```

```
      }
   }
}
```

```
bool BModEcoSys::BuildCustomerList( BFrmr* FrmrPtr )
{
   // Check if CustomerList is already in existence.
   if( FrmrPtr->CustomerList.GetNoOfContacts() > 0 ) return TRUE;

   // For this Frmr, canvass all Wrkrs and Frmrs in the
   //   commuting area and add to customer list.
   TwpLotPtr = (BTwpLot*) FrmrPtr->GetLotVdPtr();
   CommuteAreaPtr = &(TwpLotPtr->CommuteArea);
   CaWidth = CommuteAreaPtr->CaWidth;
   CaHeight = CaWidth;
   for( CaRow = 0; CaRow < CaHeight; CaRow++ )
   {
      for( CaCol = 0; CaCol < CaWidth; CaCol++ )
      {
         TrgtTwpLotPtr = (BTwpLot*) CommuteAreaPtr->GetTwpLotPtr( CaCol, CaRow );

         // If this place is residential, add Wrkr contacts.
         if( TrgtTwpLotPtr->IsResidential() == TRUE )
         {
            NoOfResidents = TrgtTwpLotPtr->ResidentList.GetNoOfResidents();
            for( ResNo = 0; ResNo < _ME_ResSlotsMax; ResNo++ )
            {
               long   PtrPairActiveFlag;
               PtrPairActiveFlag = TrgtTwpLotPtr->ResidentList.GetResidentPtr( ResNo )->PtrPairActiveFlag;
               if( PtrPairActiveFlag == _ME_PtrPairActiveFlag_Yes )
               {
                  WrkrVdPtr = TrgtTwpLotPtr->ResidentList.GetWrkrVdPtr( ResNo );
                  WrkrPtr = (BWrkr*) WrkrVdPtr;
                  WrkrsLotVdPtr = WrkrPtr->LotPtrPair.ObjectVdPtr;
                  FrmrVdPtr = (void*) FrmrPtr;
                  FrmrsLotVdPtr = (void*) TwpLotPtr;
                  FrmrMtchPtr = FrmrPtr->CustomerList.LinkNewWrkrAtBottom( _ME_CI_AgentType_Frmr,
                                                                           FrmrPtr->SlotNo,
                                                                           FrmrVdPtr,
                                                                           WrkrPtr->SlotNo,
                                                                           WrkrVdPtr,
                                                                           TrgtTwpLotPtr->LotSlotNo,
                                                                           WrkrsLotVdPtr );
```

```
            if( FrmrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
            // Add this Frmr to the Wrkr's supplier list.
            WrkrMtchPtr = WrkrPtr->SupplierList.LinkNewFrmrAtBottom( _ME_CI_AgentType_Wrkr,
                                                    WrkrPtr->SlotNo,
                                                    WrkrVdPtr,
                                                    FrmrPtr->SlotNo,
                                                     FrmrVdPtr,
                                                     TwpLotPtr->LotSlotNo,
                                                     FrmrsLotVdPtr );
            if( WrkrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
            FrmrMtchPtr->MtchPtr = WrkrMtchPtr;
            FrmrMtchPtr->MtchSlotNo = WrkrMtchPtr->SlotNo;
            WrkrMtchPtr->MtchPtr = FrmrMtchPtr;
            WrkrMtchPtr->MtchSlotNo = FrmrMtchPtr->SlotNo;
         }
      }  // End of for ResNo.
   }  // End of if Residential.


   // If this place is Commercial, add a Frmr contact.
   if( TrgtTwpLotPtr->IsCommercial() )    // Can (must) sell to one's self.
   {
      TrgtFrmrVdPtr = TrgtTwpLotPtr->FrmrPtrPair.ObjectVdPtr;
      TrgtFrmrPtr = (BFrmr*) TrgtFrmrVdPtr;
      TrgtFrmrsLotVdPtr = TrgtFrmrPtr->LotPtrPair.ObjectVdPtr;
      FrmrVdPtr = (void*) FrmrPtr;
      FrmrsLotVdPtr = (void*) TwpLotPtr;
      FrmrMtchPtr = FrmrPtr->CustomerList.LinkNewFrmrAtBottom( _ME_CI_AgentType_Frmr,
                                                  FrmrPtr->SlotNo,
                                                  FrmrVdPtr,
                                                  TrgtFrmrPtr->SlotNo,
                                                  TrgtFrmrVdPtr,
                                                  TrgtTwpLotPtr->LotSlotNo,
                                                  TrgtFrmrsLotVdPtr );
      if( FrmrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
      // Add this Frmr to the target Frmr's supplier list.
      TrgtMtchPtr = TrgtFrmrPtr->SupplierList.LinkNewFrmrAtBottom( _ME_CI_AgentType_Frmr,
                                                     TrgtFrmrPtr->SlotNo,
                                                     TrgtFrmrVdPtr,
                                                     FrmrPtr->SlotNo,
                                                     FrmrVdPtr,
                                                     TwpLotPtr->LotSlotNo,
                                                     FrmrsLotVdPtr );
      if( TrgtMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
```

```
            FrmrMtchPtr->MtchPtr = TrgtMtchPtr;
            FrmrMtchPtr->MtchSlotNo = TrgtMtchPtr->SlotNo;
            TrgtMtchPtr->MtchPtr = FrmrMtchPtr;
            TrgtMtchPtr->MtchSlotNo = FrmrMtchPtr->SlotNo;
         }  // End of if Commercial.
      }  // End of for CaCol.
   }  // End of for CaRow.
}
```

bool BModEcoSys::RemoveThisWrkrFromCustomerLists(BWrkr* ThisWrkrPtr)

```
{
   // ThisWrkr has a SupplierList with contacts for Frmrs.
   //    Contact each Frmr on the SupplierList.
   //    Tell them to delete ThisWrkr from their CustomerList.
   //    Then delete them from ThisWrkr's SupplierList.

   // Go through the list of suppliers within the old commuting
   //   area and delete this Wrkr from each union list.
   NoOfSuppliers = ThisWrkrPtr->SupplierList.GetNoOfContacts();
   ContactPtr = ThisWrkrPtr->SupplierList.GetTopContactPtr();
   for( SupplierNo = 0; SupplierNo < NoOfSuppliers; SupplierNo++ )
   {
      SupplierListContactPtr = ContactPtr;
      // Get a pointer to the Frmr from the supplier contact list.
      SupplierPtr = (BFrmr*) (ContactPtr->AgentVdPtr);
      // Get a pointer to the matching BContact item in the
      //   supplier's customer list.
      CustomerListContactPtr = ContactPtr->MtchPtr;
      // Step the ContactPtr back one in the employer list.
      ContactPtr = SupplierListContactPtr->PrevPtr;

      // Delete the Wrkr from the customer list.
      Result = SupplierPtr->CustomerList.DelContact( CustomerListContactPtr );
      // Delete the contact.
      Result = ThisWrkrPtr->SupplierList.DelContact( SupplierListContactPtr );
   }
}
```

bool BModEcoSys::RemoveThisFrmrFromSupplierLists(BFrmr* ThisFrmrPtr)

```
{
   // This Frmr has a CustomerList with contacts for Frmrs and Wrkrs.
   //    Contact each Agent on the CustomerList.
   //    Tell them to delete ThisFrmr from their SupplierList.
```

```
    //    Then delete them from ThisFrmr's CustomerList.

    // Go through the list of customers within the old commuting
    //   area and delete this Frmr from each supplier list.

    NoOfCustomers = ThisFrmrPtr->CustomerList.GetNoOfContacts();
    ContactPtr = ThisFrmrPtr->CustomerList.GetTopContactPtr();
    for( CustomerNo = 0; CustomerNo < NoOfCustomers; CustomerNo++ )
    {
        CustomerListContactPtr = ContactPtr;
        // The customer may be a Wrkr or a Frmr.
        if( CustomerListContactPtr->AgentType == _ME_CI_AgentType_Frmr )
        {
            // Get a pointer to the Frmr from the employer contact list.
            FrmrCustomerPtr = (BFrmr*) (CustomerListContactPtr->AgentVdPtr);
            // Get a pointer to the matching BContact item in the
            //   suppliers SupplierList.
            SupplierListContactPtr = CustomerListContactPtr->MtchPtr;
            // Step the ContactPtr back one in the customer list.
            ContactPtr = CustomerListContactPtr->PrevPtr;

            // Delete the customer from the supplier list.
            Result = FrmrCustomerPtr->SupplierList.DelContact( SupplierListContactPtr );
            // Delete the contact.
            Result = ThisFrmrPtr->CustomerList.DelContact( CustomerListContactPtr );
        }
        else
        {
            // Get a pointer to the Frmr from the employer contact list.
            WrkrCustomerPtr = (BWrkr*) (CustomerListContactPtr->AgentVdPtr);
            // Get a pointer to the matching BContact item in the
            //   suppliers SupplierList.
            SupplierListContactPtr = CustomerListContactPtr->MtchPtr;
            // Step the ContactPtr back one in the customer list.
            ContactPtr = CustomerListContactPtr->PrevPtr;

            // Delete the customer from the supplier list.
            Result = WrkrCustomerPtr->SupplierList.DelContact( SupplierListContactPtr );
            // Delete the contact.
            Result = ThisFrmrPtr->CustomerList.DelContact( CustomerListContactPtr );
        }
    }
}
```

bool BModEcoSys::AddThisWrkrToCustomerLists(BWrkr* ThisWrkrPtr)

```
{
   // Package Wrkr data for use.
   WrkrVdPtr = (void*) ThisWrkrPtr;
   TwpLotPtr = (BTwpLot*) ThisWrkrPtr->LotPtrPair.ObjectVdPtr;
   WrkrsLotVdPtr = (void*) TwpLotPtr;

   // ThisWrkr's new TwpLotPtr is already set.  We need to go
   //   through the commuting area and add ThisWrkr to the
   //   customer list of each Frmr in the commuting area.
   CaPtr = &TwpLotPtr->CommuteArea;
   CaWidth = CaPtr->GetCaWidth();
   for( CaRow = 0; CaRow < CaWidth; CaRow++ )
   {
      for( CaCol = 0; CaCol < CaWidth; CaCol++ )
      {
         TrgtTwpLotPtr = (BTwpLot*) CaPtr->GetTwpLotPtr( CaCol, CaRow );
         LotDevType = TrgtTwpLotPtr->GetLotDevType();
         if( TrgtTwpLotPtr->IsCommercial() == TRUE )
         {
            FrmrPtr = (BFrmr*) TrgtTwpLotPtr->GetFrmrVdPtr();
            FrmrVdPtr = (void*) FrmrPtr;
            FrmrsLotVdPtr = (void*) TrgtTwpLotPtr;
            // This is a potential supplier.  Add ThisWrkr to the customer list.
            FrmrMtchPtr = FrmrPtr->CustomerList.LinkNewWrkrAtBottom( _ME_CI_AgentType_Frmr,
                                                                     FrmrPtr->SlotNo,
                                                                     FrmrVdPtr,
                                                                     ThisWrkrPtr->SlotNo,
                                                                     WrkrVdPtr,
                                                                     TwpLotPtr->LotSlotNo,
                                                                     WrkrsLotVdPtr );
            if( FrmrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
            // Add this supplier to ThisWrkr's supplier list.
            WrkrMtchPtr = ThisWrkrPtr->SupplierList.LinkNewFrmrAtBottom( _ME_CI_AgentType_Wrkr,
                                                                         ThisWrkrPtr->SlotNo,
                                                                         WrkrVdPtr,
                                                                         FrmrPtr->SlotNo,
                                                                         FrmrVdPtr,
                                                                         TrgtTwpLotPtr->LotSlotNo,
                                                                         FrmrsLotVdPtr );
            if( WrkrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
            FrmrMtchPtr->MtchPtr = WrkrMtchPtr;
            FrmrMtchPtr->MtchSlotNo = WrkrMtchPtr->SlotNo;
```

```
            WrkrMtchPtr->MtchPtr = FrmrMtchPtr;
            WrkrMtchPtr->MtchSlotNo = FrmrMtchPtr->SlotNo;
         }
      }
   }
}
```

bool BModEcoSys::AddThisFrmrToCustomerLists(BFrmr* ThisFrmrPtr)

```
{
   // ThisFrmr wants to get onto the CustomerList of all of the Frmrs
   //   in the commuting area.
   // Package Wrkr data for use.
   ThisFrmrVdPtr = (void*) ThisFrmrPtr;
   ThisFrmrsTwpLotPtr = (BTwpLot*) ThisFrmrPtr->LotPtrPair.ObjectVdPtr;
   ThisFrmrsLotVdPtr = (void*) ThisFrmrsTwpLotPtr;

   // ThisFrmr's new TwpLotPtr is already set.  We need to go
   //   through the commuting area and add ThisFrmr to the
   //   customer list of each Frmr in the commuting area.
   CaPtr = &ThisFrmrsTwpLotPtr->CommuteArea;
   CaWidth = CaPtr->GetCaWidth();
   for( CaRow = 0; CaRow < CaWidth; CaRow++ )
   {
      for( CaCol = 0; CaCol < CaWidth; CaCol++ )
      {
         TrgtTwpLotPtr = (BTwpLot*) CaPtr->GetTwpLotPtr( CaCol, CaRow );
         LotDevType = TrgtTwpLotPtr->GetLotDevType();
         if( TrgtTwpLotPtr->IsCommercial() == TRUE )
         {
            TrgtFrmrsLotVdPtr = (void*) TrgtTwpLotPtr;
            TrgtFrmrVdPtr = TrgtTwpLotPtr->GetFrmrVdPtr();
            TrgtFrmrPtr = (BFrmr*) TrgtFrmrVdPtr;
            // TrgtFrmr is a potential supplier.  Add ThisFrmr to the customer list.
            TrgtFrmrMtchPtr = TrgtFrmrPtr->CustomerList.LinkNewFrmrAtBottom( _ME_CI_AgentType_Frmr,
                                                                             TrgtFrmrPtr->SlotNo,
                                                                             TrgtFrmrVdPtr,
                                                                             ThisFrmrPtr->SlotNo,
                                                                             ThisFrmrVdPtr,
                                                                             ThisFrmrsTwpLotPtr->LotSlotNo,
                                                                             ThisFrmrsLotVdPtr );
            if( TrgtFrmrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
            // Add TrgtFrmr to ThisFrmr's supplier list.
            ThisFrmrMtchPtr = ThisFrmrPtr->SupplierList.LinkNewFrmrAtBottom( _ME_CI_AgentType_Frmr,
```

```
                                                                ThisFrmrPtr->SlotNo,
                                                                ThisFrmrVdPtr,
                                                                TrgtFrmrPtr->SlotNo,
                                                                TrgtFrmrVdPtr,
                                                                TrgtTwpLotPtr->LotSlotNo,
                                                                TrgtFrmrsLotVdPtr );
            if( ThisFrmrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
            TrgtFrmrMtchPtr->MtchPtr = ThisFrmrMtchPtr;
            TrgtFrmrMtchPtr->MtchSlotNo = ThisFrmrMtchPtr->SlotNo;
            ThisFrmrMtchPtr->MtchPtr = TrgtFrmrMtchPtr;
            ThisFrmrMtchPtr->MtchSlotNo = TrgtFrmrMtchPtr->SlotNo;
         }  // End of if Commercial.
      }  // End of for( CaCol.
   }  // End of for( CaRow.
}
```

bool BModEcoSys::AddThisFrmrToSupplierLists(BFrmr* ThisFrmrPtr)

```
{
   // ThisFrmr wants to get onto the supplier lists of all of the customers
   //   in the commuting area.
   // Package Wrkr data for use.
   ThisFrmrVdPtr = (void*) ThisFrmrPtr;
   ThisFrmrsLotVdPtr = ThisFrmrPtr->LotPtrPair.ObjectVdPtr;
   ThisFrmrsLotPtr = (BTwpLot*) ThisFrmrsLotVdPtr;

   // ThisFrmrsLotPtr is already set.  We need to go
   //   through the commuting area and add this Frmr to the
   //   supplier list of each Wrkr and Frmr in the commuting area.
   CaPtr = &(ThisFrmrsLotPtr->CommuteArea);
   CaWidth = CaPtr->GetCaWidth();
   for( CaRow = 0; CaRow < CaWidth; CaRow++ )
   {
      for( CaCol = 0; CaCol < CaWidth; CaCol++ )
      {
         TrgtTwpLotPtr = (BTwpLot*) CaPtr->GetTwpLotPtr( CaCol, CaRow );
         LotDevType = TrgtTwpLotPtr->GetLotDevType();
         if( TrgtTwpLotPtr->IsCommercial() )    // Can (must) sell to one's self.
         {
            CustomerVdPtr = TrgtTwpLotPtr->GetFrmrVdPtr();
            FrmrCustomerPtr = (BFrmr*) CustomerVdPtr;
            CustomersLotVdPtr = (void*) TrgtTwpLotPtr;
            // That Trgt Frmr is a potential customer.  Add ThisFrmr to the SupplierList of the TrgtFrmr.
            CustomerMtchPtr = FrmrCustomerPtr->SupplierList.LinkNewFrmrAtBottom( _ME_CI_AgentType_Frmr,
```

```
                                                        FrmrCustomerPtr->SlotNo,
                                                        CustomerVdPtr,
                                                        ThisFrmrPtr->SlotNo,
                                                        ThisFrmrVdPtr,
                                                        ThisFrmrsLotPtr->LotSlotNo,
                                                        ThisFrmrsLotVdPtr );
    if( CustomerMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
    // Add Trgt Frmr to the CustomerList of ThisFrmr.
    ThisFrmrMtchPtr = ThisFrmrPtr->CustomerList.LinkNewFrmrAtBottom( _ME_CI_AgentType_Frmr,
                                                        ThisFrmrPtr->SlotNo,
                                                        ThisFrmrVdPtr,
                                                        FrmrCustomerPtr->SlotNo,
                                                        CustomerVdPtr,
                                                        TrgtTwpLotPtr->LotSlotNo,
                                                        CustomersLotVdPtr );
    if( ThisFrmrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
    CustomerMtchPtr->MtchPtr = ThisFrmrMtchPtr;
    CustomerMtchPtr->MtchSlotNo = ThisFrmrMtchPtr->SlotNo;
    ThisFrmrMtchPtr->MtchPtr = CustomerMtchPtr;
    ThisFrmrMtchPtr->MtchSlotNo = CustomerMtchPtr->SlotNo;
}

if( TrgtTwpLotPtr->IsResidential() )
{
    for( ResNo = 0; ResNo < _ME_ResSlotsMax; ResNo++ )
    {
        BPtrPair* ActivePtrPairPtr = NULL;
        ActivePtrPairPtr = TrgtTwpLotPtr->ResidentList.GetResidentPtr( ResNo );
        if( ActivePtrPairPtr->PtrPairActiveFlag == _ME_PtrPairActiveFlag_Yes )
        {
            CustomerVdPtr = ActivePtrPairPtr->ObjectVdPtr;
            WrkrCustomerPtr = (BWrkr*) CustomerVdPtr;
            CustomersLotVdPtr = (void*) TrgtTwpLotPtr;
            // That TrgtWrkr is a potential customer.  Add ThisFrmr to the SupplierList of the TrgtWrkr.
            CustomerMtchPtr = WrkrCustomerPtr->SupplierList.LinkNewFrmrAtBottom( _ME_CI_AgentType_Wrkr,
                                                        WrkrCustomerPtr->SlotNo,
                                                        CustomerVdPtr,
                                                        ThisFrmrPtr->SlotNo,
                                                        ThisFrmrVdPtr,
                                                        ThisFrmrsLotPtr->LotSlotNo,
                                                        ThisFrmrsLotVdPtr );
            if( CustomerMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
            // Add Trgt Wrkr to the customer list of ThisFrmr.
            ThisFrmrMtchPtr = ThisFrmrPtr->CustomerList.LinkNewWrkrAtBottom( _ME_CI_AgentType_Frmr,
```

```
                                                                    ThisFrmrPtr->SlotNo,
                                                                    ThisFrmrVdPtr,
                                                                    WrkrCustomerPtr->SlotNo,
                                                                    CustomerVdPtr,
                                                                    TrgtTwpLotPtr->LotSlotNo,
                                                                    CustomersLotVdPtr );
                  if( ThisFrmrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
                  CustomerMtchPtr->MtchPtr = ThisFrmrMtchPtr;
                  CustomerMtchPtr->MtchSlotNo = ThisFrmrMtchPtr->SlotNo;
                  ThisFrmrMtchPtr->MtchPtr = CustomerMtchPtr;
                  ThisFrmrMtchPtr->MtchSlotNo = CustomerMtchPtr->SlotNo;
               }  // End of _ME_PtrPairActiveFlag_Yes
            }  // End of for( ResNo;.
         }  // End of IsResidential.
      }  // End of for( CaCol;.
   }  // End of for( CaRow;.
}
```

bool BModEcoSys::RemoveThisFrmrFromEmployerLists(BFrmr* ThisFrmrPtr)

```
{
   // This Frmr has a UnionList with contacts for Workers (Wrkrs).
   //    Contact each Worker on the UnionList.
   //    Tell them to delete ThisFrmr from their EmployerList.
   //    Then delete the Worker from the UnionList.

   // Go through the list of workers within the old commuting
   //   area and delete ThisFrmr from each employer list.
   NoOfWorkers = ThisFrmrPtr->UnionList.GetNoOfContacts();
   ContactPtr = ThisFrmrPtr->UnionList.GetTopContactPtr();
   for( WorkerNo = 0; WorkerNo < NoOfWorkers; WorkerNo++ )
   {
      UnionListContactPtr = ContactPtr;
      // Get a pointer to the worker from the union contact list.
      WorkerPtr = (BWrkr*) (UnionListContactPtr->AgentVdPtr);
      // Get a pointer to the matching BContact item in the
      //   employer contact list.
      EmployerListContactPtr = UnionListContactPtr->MtchPtr;
      // Step the ContactPtr back one in the union list.
      ContactPtr = UnionListContactPtr->PrevPtr;

      // Delete the Frmr from the employer list.
      Result = WorkerPtr->EmployerList.DelContact( EmployerListContactPtr );
      // Delete the contact.
```

```
      Result = ThisFrmrPtr->UnionList.DelContact( UnionListContactPtr );
   }
}
```

bool BModEcoSys::RemoveThisFrmrFromCustomerLists(BFrmr* ThisFrmrPtr)

```
{
   // ThisFrmr has a SupplierList with contacts for supplier Frmrs.
   //    Contact each supplier on the SupplierList.
   //    Tell them to delete ThisFrmr from their CustomerList.
   //    Then delete the supplier from the SupplierList.

   // Go through the list of workers within the old commuting
   //   area and delete ThisFrmr from each employer list.
   NoOfSuppliers = ThisFrmrPtr->SupplierList.GetNoOfContacts();
   ContactPtr = ThisFrmrPtr->SupplierList.GetTopContactPtr();
   for( SupplierNo = 0; SupplierNo < NoOfSuppliers; SupplierNo++ )
   {
      SupplierListContactPtr = ContactPtr;
      // Get a pointer to the supplier from the SupplierList.
      SupplierPtr = (BFrmr*) (SupplierListContactPtr->AgentVdPtr);
      // Get a pointer to the matching BContact item in the
      //   CustomerList.
      CustomerListContactPtr = SupplierListContactPtr->MtchPtr;
      // Step the ContactPtr back one in the SupplierList.
      ContactPtr = SupplierListContactPtr->PrevPtr;

      // Delete the Frmr from the CustomerList.
      Result = SupplierPtr->CustomerList.DelContact( CustomerListContactPtr );
      // Delete the contact in the SupplierList.
      Result = ThisFrmrPtr->SupplierList.DelContact( SupplierListContactPtr );
   }
}
```

bool BModEcoSys::AddThisFrmrToEmployerLists(BFrmr* ThisFrmrPtr)

```
{
   // ThisFrmr wants to get onto the EmployerList of each of the workers
   //   in the commuting area, and put those workers onto its UnionList.

   // Package Wrkr data for use.
   ThisFrmrVdPtr = (void*) ThisFrmrPtr;
   ThisFrmrsTwpLotPtr = (BTwpLot*) ThisFrmrPtr->LotPtrPair.ObjectVdPtr;
   ThisFrmrsLotVdPtr = (void*) ThisFrmrsTwpLotPtr;
```

```
// ThisFrmr's new TwpLotPtr is already set.  We need to go
//   through the commuting area and add ThisFrmr to the
//   EmployerList of each Wrkr in the commuting area.
CaPtr = &ThisFrmrsTwpLotPtr->CommuteArea;
CaWidth = CaPtr->GetCaWidth();
for( CaRow = 0; CaRow < CaWidth; CaRow++ )
{
   for( CaCol = 0; CaCol < CaWidth; CaCol++ )
   {
      TrgtTwpLotPtr = (BTwpLot*) CaPtr->GetTwpLotPtr( CaCol, CaRow );
      LotDevType = TrgtTwpLotPtr->GetLotDevType();
      // If this place is residential, add Wrkr contacts.
      if( TrgtTwpLotPtr->IsResidential() == TRUE )
      {
         // Residential township lots must have at least one consumer.
         NoOfResidents = TrgtTwpLotPtr->ResidentList.GetNoOfResidents();
         for( ResNo = 0; ResNo < _ME_ResSlotsMax; ResNo++ )
         {
            long   PtrPairActiveFlag;
            PtrPairActiveFlag = TrgtTwpLotPtr->ResidentList.GetResidentPtr( ResNo )->PtrPairActiveFlag;
            if( PtrPairActiveFlag == _ME_PtrPairActiveFlag_Yes )
            {
               WorkerVdPtr = TrgtTwpLotPtr->ResidentList.GetWrkrVdPtr( ResNo );
               WorkerPtr = (BWrkr*) WorkerVdPtr;
               WorkersLotVdPtr = WorkerPtr->LotPtrPair.ObjectVdPtr;
               ThisFrmrVdPtr = (void*) ThisFrmrPtr;
               ThisFrmrsLotVdPtr = (void*) ThisFrmrsTwpLotPtr;
               // Add this Wrkr to the Frmr's union list.
               ThisFrmrMtchPtr = ThisFrmrPtr->UnionList.LinkNewWrkrAtBottom( _ME_CI_AgentType_Frmr,
                                                                  ThisFrmrPtr->SlotNo,
                                                                  ThisFrmrVdPtr,
                                                                  WorkerPtr->SlotNo,
                                                                  WorkerVdPtr,
                                                                  TrgtTwpLotPtr->LotSlotNo,
                                                                  WorkersLotVdPtr );
               if( ThisFrmrMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
               // Add this Frmr to the Wrkr's employer list.
               WorkerMtchPtr = WorkerPtr->EmployerList.LinkNewFrmrAtBottom( _ME_CI_AgentType_Wrkr,
                                                                 WorkerPtr->SlotNo,
                                                                 WorkerVdPtr,
                                                                 ThisFrmrPtr->SlotNo,
                                                                  ThisFrmrVdPtr,
                                                                  ThisFrmrsTwpLotPtr->LotSlotNo,
                                                                  ThisFrmrsLotVdPtr );
```

```
                    if( WorkerMtchPtr == NULL ) { Sounds.ContactOverflow(); return FALSE; }
                    ThisFrmrMtchPtr->MtchPtr = WorkerMtchPtr;
                    ThisFrmrMtchPtr->MtchSlotNo = WorkerMtchPtr->SlotNo;
                    WorkerMtchPtr->MtchPtr = ThisFrmrMtchPtr;
                    WorkerMtchPtr->MtchSlotNo = ThisFrmrMtchPtr->SlotNo;
                }
            }  // End of for ResNo.
        }  // End of if Residential.
    }
  }
}
```

The Economic Engine Algorithm

In the following presentation, the code is not executable. All declarations, ASSERTions, and verification code has been removed. In addition, all code associated with display, status reporting or statistical data collection has been removed. Finally, all code that handles scenarios and switchable features other than those used in the scenario of interest has been deleted. What remains is the code that defines the transition rule of the FSM of interest.

The program that causes the FSM to apply the transition rule once is AdvanceOneTick(). It is run by an independent thread of execution in an endless loop which terminates only when all agents are dead, the STOP button is pressed, or the HaltAt time is reached. The following list of programs define the main functions performed during one tick.

Economic Engine Function Names

```
bool AdvanceOneTick( void );

long DoSetup( long CurrentFunction );
long DoJobOffers( long CurrentFunction );
long DoWrkrsMove( long CurrentFunction );
long DoSellInventory( long CurrentFunction );
long DoConsumeSupplyMbMEu( long CurrentFunction );
long DoSellWasteMbMu( long CurrentFunction );
long DoBuyRecycledMbMu( long CurrentFunction );
long DoAgentsRepro( long CurrentFunction );
long DoAgentsDeath( long CurrentFunction );
long DoCleanup( long CurrentFunction );
```

These programs are subsidiary to the previous list. These procedural sub-functions are roughly in order of first appearance, and are grouped under the function in which they first appear.

```
// Subsidiary to DoSetup.
bool BuildLists( void );  // Already seen in initialization.

// Subsidiary to DoJobOffers.
bool IssueJobOffers( BFrmr* FrmrPtr );
bool IssueJobOffer( BFrmr* FrmrPtr, BWrkr* WrkrPtr );
```

```
double ApplyForWrkr_MbEuGrants( double MbEuAvailable, double MbEuNeeded );
double ApplyForWrkr_CashGrants( double CashAvailable, double CashNeeded );
double ApplyForFrmr_MbEuGrants(double MbEuAvailable, double MbEuNeeded, double MbEuCommitted );
double ApplyForFrmr_MbMzGrants(double MbMuRecycledAvailable, double MbMuRecycledNeeded, long MuType );
double ApplyForFrmr_CashGrants(double CashAvailable, double CashNeeded );
bool DrawWrkr_MbEuGrants( BWrkr* WrkrPtr, double Wrkr_MbEuGrantsNeeded );
bool DrawWrkr_CashGrants( BWrkr* WrkrPtr, double Wrkr_CashGrantsNeeded );
bool DrawFrmr_MbEuGrants( BFrmr* FrmrPtr, double Frmr_MbEuGrantsNeeded );
bool DrawFrmr_MbMzGrants( BFrmr* FrmrPtr, double Frmr_MbMzGrantsNeeded );
bool DrawFrmr_CashGrants( BFrmr* FrmrPtr, double Frmr_CashGrantsNeeded );
double ConvertRecycledToInventory();

// Subsidiary to DoWrkrsMove.
bool MoveWrkr( BWrkr* WrkrPtr );
bool FindVacantResidence( BTwpLot* TwpLotPtr, BPtrPair* LaPtr );
bool AddResident( BTwpLot* TwpLotPtr, BWrkr* WrkrPtr );
bool DelResident( BTwpLot* TwpLotPtr, BWrkr* WrkrPtr );
bool SearchThisFrmrInLists( BFrmr* ThisFrmrPtr );
bool RemoveThisWrkrFromUnionLists( BWrkr* ThisWrkrPtr );
bool RemoveThisWrkrFromCustomerLists( BWrkr* ThisWrkrPtr );
bool RemoveThisFrmrFromEmployerLists( BFrmr* ThisFrmrPtr );
bool RemoveThisFrmrFromSupplierLists( BFrmr* ThisFrmrPtr );
bool RemoveThisFrmrFromCustomerLists( BFrmr* ThisFrmrPtr );

// Subsidiary to DoSellInventory.
bool IssueSalesPitches( BFrmr* FrmrPtr );
bool IssueSalesPitch( BFrmr* FrmrPtr, BWrkr* WrkrPtr );
bool IssueSalesPitch( BFrmr* FrmrPtr, BFrmr* TrgtFrmrPtr );

// Subsidiary to DoConsumeSupplyMbMEu.
bool ConsumeMateriel( BMaterielUnit* SupdMuPtr, BWrkr* WrkrPtr );
bool ConsumeMateriel( BMaterielUnit* SupdMuPtr, BFrmr* FrmrPtr );

// Subsidiary to DoSellWasteMbMu.
bool SellWrkrWaste( BWrkr* WrkrPtr );
bool SellFrmrWaste( BFrmr* FrmrPtr );

// Subsidiary to DoBuyRecycledMbMu.
bool BuyRecycledMbMu( BFrmr* FrmrPtr );

// Subsidiary to DoAgentsRepro.
bool ReproduceFrmr( BFrmr* FrmrPtr );
bool ReproduceWrkr( BWrkr* WrkrPtr );
bool FindVacantLot( BTwpLot* TwpLotPtr, BPtrPair* LaPtr );
```

```
// Subsidiary to DoAgentsDeath.
bool KillFrmr( BFrmr* FrmrPtr );
bool ClearUnionList( BFrmr* FrmrPtr );
bool ClearCustomerList( BFrmr* ThisFrmrPtr );
bool ClearSupplierList( BFrmr* ThisFrmrPtr );
bool ClearFrmrInventory( BFrmr* FrmrPtr );
bool KillWrkr( BWrkr* WrkrPtr );
bool ClearEmployerList( BWrkr* WrkrPtr );
bool ClearSupplierList( BWrkr* WrkrPtr );
bool ClearWrkrInventory( BWrkr* WrkrPtr );

// Subsidiary to DoCleanup.
bool AgeAllAgents( void );
```

// Subsidiary to DoSetup.

long BModEcoSys::DoSetup(long CurrentFunction)

```
{
   // Build any contact lists, if missing.
   BuildLists();
}
```

BuildLists() creates contact lists for each private sector agent.

// Subsidiary to DoJobOffers.

long BModEcoSys::DoJobOffers(long CurrentFunction)

```
{
   // Establish a random order for processing the Frmrs.
   // Ensure the list is empty.
   FrmrSlotNoList.InitBSlotNoList();
   FrmrSlotNoList.RsPtr = RsPtr;
   NoOfFrmrs = FrmrList.GetNoOfFrmrs();
   FrmrPtr = FrmrList.GetTopFrmrPtr();
   for( FrmrNo = 0; FrmrNo < NoOfFrmrs; FrmrNo++ )
   {
      FrmrSlotNoList.AddSlotNo( FrmrPtr->SlotNo );
      FrmrPtr = FrmrPtr->NextPtr;
   }

   // Process all Frmrs.
```

```
    for( FrmrNo = 0; FrmrNo < NoOfFrmrs; FrmrNo++ )
    {
        FrmrPtr = FrmrList.GetFrmrPtr( FrmrSlotNoList.GetRandomSlotNo() );
        {
            FrmrPtr->NoOfHiresMax = 4;
            // For this Frmr, canvass all Wrkrs in the
            //   commuting area and make job offers.
            IssueJobOffers( FrmrPtr );
        }
    }
}
```

bool BModEcoSys::IssueJobOffers(BFrmr* FrmrPtr)

```
{
    // At this point, we assume the Frmr is still trying to
    //   hire someone.
    TodaysDate = Township.Age.GetInSeconds();

    // Ensure the SlotNoList is empty.
    ContactSlotNoList.InitBSlotNoList();
    ContactSlotNoList.RsPtr = RsPtr;
    NoOfContacts = FrmrPtr->UnionList.GetNoOfContacts();
    if( NoOfContacts <= 0 ) return TRUE;
    ContactPtr = FrmrPtr->UnionList.GetTopContactPtr();
    for( ContactNo = 0; ContactNo < NoOfContacts; ContactNo++ )
    {
        ContactSlotNoList.AddSlotNo( ContactPtr->SlotNo );
        ContactPtr = ContactPtr->NextPtr;
    }

    for( ContactNo = 0; ContactNo < NoOfContacts; ContactNo++ )
    {
        // The Frmr can only continue to make job offers while it has
        //   cash on hand, raw mus, and MbEu.
        if( FrmrPtr->GetCashOnHand() <= 0 ) return FALSE;

        // The Frmr can only continue to make job offers while it has
        //   access to Infra MbEu.  If the Frmr_MbEuGrants program is turned on,
        //   and the MMgr has MbEu available, this is access.
        if( ( FrmrPtr->MbEu.GetNoOfMu() == 0.0 ) &&            // Has no MbEu of its own.
            ( ( Toggles.Frmr_MbEuGrants_Switch == _ME_Frmr_MbEuGrants_Switch_Off ) ||  // No Frmr_MbEuGrants program
              ( Township.MbEuG_MuPool.GetNoOfMu() == 0.0 ) ) )         // No Grants available.
        {
```

```
   return FALSE;
}

// The Frmr can only continue to make job offers while it has
//   access to Recycled MbMu.  If the Frmr_MbMzGrants program is turned on,
//   and the MMgr has MbMu available, this is access.
if( ( FrmrPtr->RecycledMbMzOnHand.MuPool[ FrmrPtr->FrmrType ].GetNoOfMu() == 0.0 ) &&// Has no MbMu of its own.
   ( ( Toggles.Frmr_MbMzGrants_Switch == _ME_Frmr_MbMzGrants_Switch_Off ) ||      // No Frmr_MbMzGrants program
     ( Township.GetNoOfMbMuGMbMEu( FrmrPtr->FrmrType ) == 0.0 ) ) )        // No Grants available.
{
   return FALSE;
}

// The Frmr will only make a job offer if it has jobs open.
// The logic than ensures this is in a break statement at the bottom of the loop.

// Additionally, the Frmr will only continue to make job offers while
//   it is within planning parameter HPF.  In particular, the value of
//   its inventory cannot exceed a pre-determined factor of its total
//   net worth.  It may choose to decline to hire more.
if( !FrmrPtr->IsPlanningToHireStaff() ) return FALSE;

// Frmr is still good-to-go.  Make the next contact.
ContactPtr = ContactPool.GetContactPtr( ContactSlotNoList.GetRandomSlotNo() );
// Contact the Wrkr.
if( ContactPtr->Dlc < TodaysDate )
{
   WrkrPtr = (BWrkr*) ContactPtr->AgentVdPtr;
   // A Wrkr cannot be hired twice in one day,
   //   and must be able to work.
   if( WrkrPtr->DateLastHired < TodaysDate )     // Has not worked yet today
   {
      if( ( WrkrPtr->MbEu.GetNoOfMu() > 0.0 ) ||          // Has MbEu of its own.
         ( ( Toggles.Wrkr_MbEuGrants_Switch == _ME_Wrkr_MbEuGrants_Switch_On ) &&// Wrkr_MbEuGrants program
           ( Township.MbEuG_MuPool.GetNoOfMu() > 0.0 ) ) )       // Grants available.
      {  // Wrkr wants job offer
         ResultOfJobOffer = IssueJobOffer( FrmrPtr, WrkrPtr );
         // Mark the contact as completed.
         ContactPtr->Dlc = TodaysDate;
         ContactPtr->MtchPtr->Dlc = TodaysDate;
         if( ResultOfJobOffer == TRUE )
         {
            WrkrPtr->DateLastHired = TodaysDate;
         }
```

```
            }
         }
      }
      if( FrmrPtr->NoOfHires >= FrmrPtr->NoOfHiresMax ) break;
   }
}
```

bool BModEcoSys::IssueJobOffer(BFrmr* FrmrPtr, BWrkr* WrkrPtr)

```
{
   // In this function a Frmr issues a job offer to a Wrkr.
   TodaysDate = Township.Age.GetInSeconds();
   if( WrkrPtr->DateLastHired == TodaysDate ) return FALSE;

   // Establish pointers to the relevant resources.
   WrkrMbEuPtr = (&WrkrPtr->MbEu);
   FrmrMbEuPtr = (&FrmrPtr->MbEu);
   MuType = FrmrPtr->FrmrType;
   RecycledMbMzPtr = (&FrmrPtr->RecycledMbMzOnHand.MuPool[ MuType ]);

   // A Wrkr must be healthy enough to complete some work.
   // A Frmr must have enough supplies and infrastructure to support some work.
   // The Frmr must have enough raw mus on hand to keep the worker
   //   busy for a while.
   ProductionLimit = _ME_DailyWorkRate_MbEu + _ME_DailyWorkRate_MbEu;

   // The Frmr and Wrkr must now negotiate a price for a day's wage.

   UnitIValue = ComputeUnitPrice( WrkrMbEuPtr->GetIntrinsicValue(), WrkrMbEuPtr->GetNoOfMu() );
   // The intrinsic price for a unit of time is now determined.
   // The Frmr will make an offer, and the Wrkr will
   //   determine if it is acceptable.
   FrmrPriceQuoteBuyLabour = FrmrPtr->GetQuoteBuyLabour( UnitMValue );
   WrkrPriceQuoteSellLabour = WrkrPtr->GetQuoteSellLabour( UnitMValue );

   // If the Wrkr wants less than the Frmr is willing to pay, deal.
   if( WrkrPriceQuoteSellLabour <= FrmrPriceQuoteBuyLabour )
   {   // Always happens in absolutely conservative scenario
      // If you get to here, the decision has been made to hire.

      // The worker gets more than he asked for, and the Frmr pays less than it offered.
      UnitPrice = ( FrmrPriceQuoteBuyLabour + WrkrPriceQuoteSellLabour ) / 2.0;
      // This is the price per 'hour' of work.
```

```
      // First, the worker performs the work using the Frmr's infrastructure.
      // Draw raw mus from the raw mu pool, time from the Wrkr's time pool,
      //   and infrastructure units from the Frmr's infra pool.
      WrkrMbEuUsed = ConvertRecycledToInventory( FrmrPtr, WrkrPtr, &MuSupplyByWrkr, UnitPrice );

      // The day's wage is the average of the two quotes, times the number of time units sold.
      // DaysWage is the ExtendedPrice of labour.
      DaysWage = Round_As_Cash( WrkrMbEuUsed * UnitPrice );

      // Pay the worker and mark his Wrkr log.
      if( FrmrPtr->GetCashOnHand() - DaysWage < 0 ) DaysWage = FrmrPtr->GetCashOnHand();
      FrmrPtr->SetCashOnHand( FrmrPtr->GetCashOnHand() - DaysWage );
      WrkrPtr->SetCashOnHand( WrkrPtr->GetCashOnHand() + DaysWage );

      FrmrPtr->NoOfHires++;
      WrkrPtr->DateLastHired = Township.Age.GetInSeconds();
   }
}
```

```
double BModEcoSys::ConvertRecycledToInventory( BFrmr* FrmrPtr, BWrkr* WrkrPtr, BMaterielUnit* BillPtr, double HourlyUnitPrice )
{
   MuType = FrmrPtr->FrmrType;

   // Pointers to relevant resource pools.
   WrkrMbEuPtr = &WrkrPtr->MbEu;
   FrmrMbEuPtr = &FrmrPtr->MbEu;
   RecycledMbMzPtr = &FrmrPtr->RecycledMbMzOnHand.MuPool[ MuType ];
   InventoryUnitPtr = &FrmrPtr->InventoryOnHand.MuPool[ MuType ];

   // At this point there are two fairly complicated paths we can take.
   // If the township has an "Every man works" program of some type, we can apply
   //   for Frmr_MbEuGrants, Frmr_MbMzGrants or for Wrkr_MbEuGrants.
   // If the township does not have such a program, we take the second path.

   // If grants are switched on, take the "Every man works" path.
   if( Wrkr_MbEuGrants_Avail || Frmr_MbEuGrants_Avail || Frmr_MbMzGrants_Avail )
   {
      // Township has an "Every man works" program wherein a Wrkr may seek
      //   Wrkr_MbEuGrants to make them capable to work (top up the MbEu),
      //   or a Frmr may seek Frmr_MbEuGrants or Frmr_MbMzGrants to make them capable to
      //   hire (top up the MbEu or MbMu).  In truth, these optional programs are provided
      //   to recycle MbEu and MbMu from the township to the Frmrs and Wrkrs.  This
      //   distributes the "Estate" MbMEu which went to the township on the
```

```
//   death of agents.
// These are free of charge, given to the agents, when needed,
//   that have earned them by successfully negotiating work.  This technique
//   would hopefuly also support quick evolution of the genes.
// They are free, because they went to the MM freely, and to balance
//   the accounts (to avoid a net transfer of value to the MM) we
//   need to put them back into the private sector for free.

// The MbEu used must be matched by MbMu from Frmr stock.
ProductionLimit = _ME_DailyWorkRate_MbEu + _ME_DailyWorkRate_MbEu;
WrkrMbEuNeeded = ProductionLimit / 2.0;
FrmrMbEuNeeded = ProductionLimit / 2.0;
FrmrMbMuNeeded = ProductionLimit;

// The Frmr must be able to afford to hire the Wrkr.
MbEuFrmrCanAfford = Round_As_Mu( FrmrPtr->GetCashOnHand() / HourlyUnitPrice );
if( WrkrMbEuNeeded > MbEuFrmrCanAfford )
{
   // Downsize the number of hours to be purchased to an affordable level.
   WrkrMbEuNeeded = MbEuFrmrCanAfford;
   FrmrMbEuNeeded = WrkrMbEuNeeded;
   FrmrMbMuNeeded = WrkrMbEuNeeded + FrmrMbEuNeeded;
}

// How many MbMu and MbEu are available for use?
WrkrMbEuAvailable = WrkrPtr->MbEu.GetNoOfMu();
if( WrkrMbEuAvailable > WrkrMbEuNeeded ) WrkrMbEuAvailable = WrkrMbEuNeeded;
FrmrMbEuAvailable = FrmrPtr->MbEu.GetNoOfMu();
if( FrmrMbEuAvailable > FrmrMbEuNeeded ) FrmrMbEuAvailable = FrmrMbEuNeeded;
FrmrMbMuAvailable = RecycledMbMzPtr->GetNoOfMu();
if( FrmrMbMuAvailable > FrmrMbMuNeeded ) FrmrMbMuAvailable = FrmrMbMuNeeded;

if( Toggles.Wrkr_MbEuGrants_Switch == _ME_Wrkr_MbEuGrants_Switch_On )
{
   Wrkr_MbEuGrantsNeeded = ApplyForWrkr_MbEuGrants( WrkrMbEuAvailable, WrkrMbEuNeeded );
   Wrkr_MbEuGrantsNeeded = Round_As_Mu( Wrkr_MbEuGrantsNeeded );
   if( Wrkr_MbEuGrantsNeeded > 0 ) Wrkr_MbEuGrants_Needed = TRUE;
}

if( _ME_Frmr_MbEuGrants_Switch_Dflt == _ME_Frmr_MbEuGrants_Switch_On )
{
   Frmr_MbEuGrantsNeeded = ApplyForFrmr_MbEuGrants( FrmrMbEuAvailable, FrmrMbEuNeeded, Wrkr_MbEuGrantsNeeded );
   Frmr_MbEuGrantsNeeded = Round_As_Mu( Frmr_MbEuGrantsNeeded );
   if( Frmr_MbEuGrantsNeeded > 0 ) Frmr_MbEuGrants_Needed = TRUE;
```

```
}

if( _ME_Frmr_MbMzGrants_Switch_Dflt == _ME_Frmr_MbMzGrants_Switch_On )
{
   Frmr_MbMzGrantsNeeded = ApplyForFrmr_MbMzGrants( FrmrMbMuAvailable, FrmrMbMuNeeded, MuType );
   Frmr_MbMzGrantsNeeded = Round_As_Mu( Frmr_MbMzGrantsNeeded );
   if( Frmr_MbMzGrantsNeeded > 0 ) Frmr_MbMzGrants_Needed = TRUE;
}

// Adjust downwards to make Wrkr and Frmr contributions equal again.
if( ( WrkrMbEuAvailable + Wrkr_MbEuGrantsNeeded ) > ( FrmrMbEuAvailable + Frmr_MbEuGrantsNeeded ) )
{
   Wrkr_MbEuGrantsNeeded = FrmrMbEuAvailable + Frmr_MbEuGrantsNeeded - WrkrMbEuAvailable;
   Wrkr_MbEuGrantsNeeded = Round_As_Mu( Wrkr_MbEuGrantsNeeded );
   if( Wrkr_MbEuGrantsNeeded < 0 )
   {
      WrkrMbEuAvailable += Wrkr_MbEuGrantsNeeded;
      Wrkr_MbEuGrantsNeeded = 0;
   }
}
else if( ( FrmrMbEuAvailable + Frmr_MbEuGrantsNeeded ) > ( WrkrMbEuAvailable + Wrkr_MbEuGrantsNeeded ) )
{
   Frmr_MbEuGrantsNeeded = WrkrMbEuAvailable + Wrkr_MbEuGrantsNeeded - FrmrMbEuAvailable;
   Frmr_MbEuGrantsNeeded = Round_As_Mu( Frmr_MbEuGrantsNeeded );
}

// The two amounts must now be equal.
WrkrMbEuNeeded = Round_As_Mu( WrkrMbEuAvailable + Wrkr_MbEuGrantsNeeded );
FrmrMbEuNeeded = Round_As_Mu( FrmrMbEuAvailable + Frmr_MbEuGrantsNeeded );

// However, the MbMu must match.
FrmrMbMuNeeded = ( FrmrMbMuAvailable + Frmr_MbMzGrantsNeeded );
if( FrmrMbMuNeeded >= ( WrkrMbEuNeeded + FrmrMbEuNeeded ) )
{
   FrmrMbMuNeeded = ( WrkrMbEuNeeded + FrmrMbEuNeeded );
}
else
{
   // We have too many MbEu lined up, and not enough MbMu.
   Adjustment = ( ( WrkrMbEuNeeded + FrmrMbEuNeeded ) - FrmrMbMuNeeded ) / 2.0;
   Adjustment = Round_As_Mu( Adjustment );
   // Reduce the WrkrEu amount, including any grants.
   WrkrMbEuNeeded = Round_As_Mu( WrkrMbEuNeeded - Adjustment );
   if( Adjustment > Wrkr_MbEuGrantsNeeded ) Wrkr_MbEuGrantsNeeded = 0;
```

```
    // Reduce the FrmrEu amount, including any grants.
    FrmrMbEuNeeded = Round_As_Mu( FrmrMbEuNeeded - Adjustment );
    if( Adjustment > Frmr_MbEuGrantsNeeded ) Frmr_MbEuGrantsNeeded = 0;
}

// Now that we know exactly how many MbEu and MbMu must be drawn from the township as Wrkr_MbEuGrants,
//   Frmr_MbEuGrants and Frmr_MbMzGrants, we can draw those amounts from the township.

if( Wrkr_MbEuGrantsNeeded > 0 )
{
    DrawWrkr_MbEuGrants( WrkrPtr, Wrkr_MbEuGrantsNeeded );
    Wrkr_MbEuGrants_Granted = TRUE;
}
else
{
    Wrkr_MbEuGrantsNeeded = 0.0;
    Wrkr_MbEuGrants_Granted = FALSE;
}
if( Frmr_MbEuGrantsNeeded > 0 )
{
    DrawFrmr_MbEuGrants( FrmrPtr, Frmr_MbEuGrantsNeeded );
    Frmr_MbEuGrants_Granted = TRUE;
}
else
{
    Frmr_MbEuGrantsNeeded = 0.0;
    Frmr_MbEuGrants_Granted = FALSE;
}

if( Frmr_MbMzGrantsNeeded > 0 )
{
    DrawFrmr_MbMzGrants( FrmrPtr, Frmr_MbMzGrantsNeeded );
    Frmr_MbMzGrants_Granted = TRUE;
}
else
{
    Frmr_MbMzGrantsNeeded = 0.0;
    Frmr_MbMzGrants_Granted = FALSE;
}

// They must be the same size.
if( WrkrMbEuNeeded > FrmrMbEuNeeded ) WrkrMbEuNeeded = FrmrMbEuNeeded;
if( FrmrMbEuNeeded > WrkrMbEuNeeded ) FrmrMbEuNeeded = WrkrMbEuNeeded;
ProductionLimit = WrkrMbEuNeeded + FrmrMbEuNeeded;
```

```
      if( ProductionLimit > RecycledMbMzPtr->GetNoOfMu() )
      {
         // Fix rounding error.
         ProductionLimit = ProductionLimit - ( 2 * ( 1 / _ME_Rounding_Factor_MbMEu ) );
         WrkrMbEuNeeded = WrkrMbEuNeeded - ( 1 / _ME_Rounding_Factor_MbMEu );
         FrmrMbEuNeeded = FrmrMbEuNeeded - ( 1 / _ME_Rounding_Factor_MbMEu );
      }
      FrmrMbMuNeeded = ProductionLimit;
   }
   else
   {
      // The township has no such grants program.  It's every agent for himself.
      // The details are not important to the scenario under study.  Deleted.
   }

   if( Is_EQ_WT( ProductionLimit, 0 ) ) ProductionLimit = 0;
   if( Is_EQ_WT( WrkrMbEuNeeded, 0 ) ) WrkrMbEuNeeded = 0;
   if( Is_EQ_WT( FrmrMbEuNeeded, 0 ) ) FrmrMbEuNeeded = 0;
   if( Is_EQ_WT( FrmrMbMuNeeded, 0 ) ) FrmrMbMuNeeded = 0;

   if( WrkrMbEuNeeded < 0 ) WrkrMbEuNeeded = 0;
   if( FrmrMbEuNeeded < 0 ) FrmrMbEuNeeded = 0;
   if( ProductionLimit < 0 ) ProductionLimit = 0;

   WrkrMbEuNeeded = Round_As_Mu( WrkrMbEuNeeded );
   FrmrMbEuNeeded = Round_As_Mu( FrmrMbEuNeeded );
   ProductionLimit = Round_As_Mu( ProductionLimit );

   BillOfLading.ZeroAllValues();
   BillOfLading.SetMuType( MuType );
   BillOfLading.DrawMateriel( WrkrMbEuNeeded, WrkrMbEuPtr );
   BillOfLading.DrawMateriel( FrmrMbEuNeeded, FrmrMbEuPtr );
   BillOfLading.DrawMateriel( ProductionLimit, RecycledMbMzPtr );
   InventoryUnitPtr->StoreMateriel( &BillOfLading );
   (*BillPtr) = BillOfLading;

   return WrkrMbEuNeeded;
}
```

double BModEcoSys::ApplyForWrkr_MbEuGrants(double MbEuAvailable, double MbEuNeeded)

```
{
   // Default is zero.
   Wrkr_MbEuGrantsAllowed = 0;
```

```
   if( Township.MbEuG_MuPool.GetNoOfMu() < 0.0 ) return Wrkr_MbEuGrantsAllowed;

   // Bump this up, if there is need.
   if( MbEuAvailable < MbEuNeeded )
      Wrkr_MbEuGrantsAllowed = MbEuNeeded - MbEuAvailable;

   // Reduce this if there is a shortage in the township (Closed system).
   if( Township.MbEuG_MuPool.GetNoOfMu() < Wrkr_MbEuGrantsAllowed )
      Wrkr_MbEuGrantsAllowed = Township.MbEuG_MuPool.GetNoOfMu() ;

   return Round_As_Mu( Wrkr_MbEuGrantsAllowed );
}
```

```
double BModEcoSys::ApplyForWrkr_CashGrants( double CashAvailable, double CashNeeded )
{
   // Default is zero.
   Wrkr_CashGrantsAllowed = 0;
   if( Township.GetCash_Grants() == 0 ) return Wrkr_CashGrantsAllowed;

   // Bump this up, if there is need.
   if( CashAvailable < CashNeeded )
      Wrkr_CashGrantsAllowed = Round_As_Cash( CashNeeded - CashAvailable );

   // Reduce this if there is a shortage in the township (Closed system).
   if( Township.GetCash_Grants() < Wrkr_CashGrantsAllowed )
      Wrkr_CashGrantsAllowed = Township.GetCash_Grants() ;

   return Round_As_Cash( Wrkr_CashGrantsAllowed );
}
```

```
double BModEcoSys::ApplyForFrmr_MbEuGrants( double MbEuAvailable, double MbEuNeeded, double MbEuCommitted )
{
   // Default is zero.
   Frmr_MbEuGrantsAllowed = 0;
   if( Township.MbEuG_MuPool.GetNoOfMu() < 0.0 ) return Frmr_MbEuGrantsAllowed;

   // Bump this up, if there is need.
   if( MbEuAvailable < MbEuNeeded )
      Frmr_MbEuGrantsAllowed = MbEuNeeded - MbEuAvailable;

   // Reduce this if there is a shortage in the township (Closed system).
   if( Township.MbEuG_MuPool.GetNoOfMu() < ( Frmr_MbEuGrantsAllowed + MbEuCommitted ) )
      Frmr_MbEuGrantsAllowed = Township.MbEuG_MuPool.GetNoOfMu() - MbEuCommitted;
```

```
   if( Frmr_MbEuGrantsAllowed < 0 ) Frmr_MbEuGrantsAllowed = 0;

   return Round_As_Mu( Frmr_MbEuGrantsAllowed );
}
```

double BModEcoSys::ApplyForFrmr_MbMzGrants(double MbMuRecycledAvailable, double MbMuRecycledNeeded, long MuType)

```
{
   // Default is zero.
   Frmr_MbMzGrantsAllowed = 0;
   if( Township.GetNoOfMbMuGMbMEu( MuType ) < 0.01 ) return Frmr_MbMzGrantsAllowed;

   // Bump this up, if there is need.
   if( MbMuRecycledAvailable < MbMuRecycledNeeded )
      Frmr_MbMzGrantsAllowed = MbMuRecycledNeeded - MbMuRecycledAvailable;

   // Reduce this if there is a shortage in the township (Closed system).
   if( Township.GetNoOfMbMuGMbMEu( MuType ) < Frmr_MbMzGrantsAllowed )
      Frmr_MbMzGrantsAllowed = Township.GetNoOfMbMuGMbMEu( MuType );

   if( Frmr_MbMzGrantsAllowed < 0 ) Frmr_MbMzGrantsAllowed = 0;

   return Round_As_Mu( Frmr_MbMzGrantsAllowed );
}
```

double BModEcoSys::ApplyForFrmr_CashGrants(double CashAvailable, double CashNeeded)

```
{
   // Default is zero.
   Frmr_CashGrantsAllowed = 0;
   if( Township.GetCash_Grants() == 0 ) return Frmr_CashGrantsAllowed;

   // Bump this up, if there is need.
   if( CashAvailable < CashNeeded )
      Frmr_CashGrantsAllowed = CashNeeded - CashAvailable;

   // Reduce this if there is a shortage in the township (Closed system).
   if( Township.GetCash_Grants() < Frmr_CashGrantsAllowed )
      Frmr_CashGrantsAllowed = Township.GetCash_Grants() ;

   return Round_As_Cash( Frmr_CashGrantsAllowed );
}
```

```
bool BModEcoSys::DrawWrkr_MbEuGrants( BWrkr* WrkrPtr, double Wrkr_MbEuGrantsNeeded )
{
   // Transfer social assistance to Wrkr.
   Wrkr_MbEuGrants.FillRequisitionForm( _ME_MU_MuType_MbEu, Wrkr_MbEuGrantsNeeded );
   Township.MbEuG_MuPool.RequisitionHours( &Wrkr_MbEuGrants );
   WrkrPtr->MbEu.StoreHours( &Wrkr_MbEuGrants );
}

bool BModEcoSys::DrawWrkr_CashGrants( BWrkr* WrkrPtr, double Wrkr_CashGrantsNeeded )
{
   // Transfer social assistance to Wrkr.
   WrkrPtr->SetCashOnHand( WrkrPtr->GetCashOnHand() + Wrkr_CashGrantsNeeded );
   Township.SetCash_Grants( Township.GetCash_Grants() - Wrkr_CashGrantsNeeded );
}

bool BModEcoSys::DrawFrmr_MbEuGrants( BFrmr* FrmrPtr, double Frmr_MbEuGrantsNeeded )
{
   // Transfer social assistance to Wrkr.
   Frmr_MbEuGrants.FillRequisitionForm( _ME_MU_MuType_MbEu, Frmr_MbEuGrantsNeeded );
   Township.MbEuG_MuPool.RequisitionHours( &Frmr_MbEuGrants );
   FrmrPtr->MbEu.StoreHours( &Frmr_MbEuGrants );
}

bool BModEcoSys::DrawFrmr_MbMzGrants( BFrmr* FrmrPtr, double Frmr_MbMzGrantsNeeded )
{
   MuType = FrmrPtr->FrmrType;

   if( Toggles.Global_Pm_Switch == _ME_Global_PriceMode_Generic )
   {
      // Transfer social assistance to Wrkr.
      Frmr_MbMzGrants.FillRequisitionForm( MuType, Frmr_MbMzGrantsNeeded );
      Township.MbMuG_MuStore.RequisitionRecycledMbMu( &Frmr_MbMzGrants );
      FrmrPtr->RecycledMbMzOnHand.MuPool[ MuType ].StoreRecycledMbMu( &Frmr_MbMzGrants );
   }
   else
   {
      // Transfer social assistance to Wrkr.
      Frmr_MbMzGrants.FillRequisitionForm( MuType, Frmr_MbMzGrantsNeeded );
      Township.MbMuG_MuPool.MuPool[ MuType ].RequisitionRecycledMbMu( &Frmr_MbMzGrants );
      FrmrPtr->RecycledMbMzOnHand.MuPool[ MuType ].StoreRecycledMbMu( &Frmr_MbMzGrants );
   }
}
```

```
bool BModEcoSys::DrawFrmr_CashGrants( BFrmr* FrmrPtr, double Frmr_CashGrantsNeeded )
{
   // Transfer social assistance to Frmr.
   FrmrPtr->SetCashOnHand( FrmrPtr->GetCashOnHand() + Frmr_CashGrantsNeeded );
   Township.SetCash_Grants( Township.GetCash_Grants() - Frmr_CashGrantsNeeded );
}
```

// Subsidiary to DoWrkrsMove.

```
long BModEcoSys::DoWrkrsMove( long CurrentFunction )
{
	NoOfWrkrs = WrkrList.GetNoOfWrkrs();
	WrkrPtr = WrkrList.GetTopWrkrPtr();
	WrkrSlotNoList.InitBSlotNoList();
	WrkrSlotNoList.RsPtr = RsPtr;
	for( WrkrNo = 0; WrkrNo < NoOfWrkrs; WrkrNo++ )
	{
		WrkrSlotNoList.AddSlotNo( WrkrPtr->SlotNo );
		WrkrPtr = WrkrPtr->NextPtr;
	}

	// Unpack today's date.
	TodaysDate = Township.Age.GetInSeconds();

	// Process all Wrkrs.
	for( WrkrNo = 0; WrkrNo < NoOfWrkrs; WrkrNo++ )
	{
		WrkrPtr = WrkrList.GetWrkrPtr( WrkrSlotNoList.GetRandomSlotNo() );
		// Determine if this Wrkr was hired today.
		DateLastHired = WrkrPtr->DateLastHired;
		if( ( DateLastHired < TodaysDate ) &&
			( WrkrPtr->IsPlanningToMove() ) )
		{
			// This Wrkr was not hired today, and is running out of
			//   money.  He/she should move.
			// For this Wrkr, canvass all residential lots in the
			//   commuting area and make a list of available slots.
			bResult = MoveWrkr( WrkrPtr );
		}
	}	// End of for WrkrNo
}
```

bool BModEcoSys::FindVacantResidence(BTwpLot* TwpLotPtr, BPtrPair* LaPtr)

```
{
    CaPtr = &TwpLotPtr->CommuteArea;
    CaWidth = CaPtr->CaWidth;
    for( CaRow = 0; CaRow < CaWidth; CaRow++ )
    {
        for( CaCol = 0; CaCol < CaWidth; CaCol++ )
        {
            TrgtTwpLotPtr = (BTwpLot*) CaPtr->GetTwpLotPtr( CaCol, CaRow );
            if( TrgtTwpLotPtr->GetLotDevType() == _ME_LotDevType_UnSpec )
                VacantLotList.AddLotSlotNo( TrgtTwpLotPtr->LotSlotNo );
            if( ( TrgtTwpLotPtr->GetLotDevType() == _ME_LotDevType_Residence ) &&
                ( TrgtTwpLotPtr->ResidentList.GetNoOfResidents() < _ME_UnitsMax_PostInd ) )
                VacantLotList.AddLotSlotNo( TrgtTwpLotPtr->LotSlotNo );
        }
    }

    if( VacantLotList.GetNoOfUnits() <= 0 )
    {
        // No vacant lots exist within the commuting area.
        LaPtr->InitBPtrPair();
        return FALSE;
    }

    NoOfVacantLots = VacantLotList.GetNoOfUnits();
    if( NoOfVacantLots > 0 )
    {
        // Declarations
        long LotSlotNo;
        long VacantLotSlotNo;

        // Randomly select one of the vacant rental lots in the list.
        RandomNumber = RsPtr->randDblExc();       // Real number (0,1)
        VacantLotSlotNo = (long) floor( RandomNumber * (double) NoOfVacantLots );
        // Figure out what lot this unit is on.
        LotSlotNo = VacantLotList.Units[ VacantLotSlotNo ];
        TrgtTwpLotPtr = Township.GetLotPtr( LotSlotNo );
        LaPtr->PtrPairActiveFlag = _ME_PtrPairActiveFlag_Yes;
        LaPtr->AddrSlotNo = -1;     // Only used with BResidentList.
        LaPtr->ObjectSlotNo = TrgtTwpLotPtr->LotSlotNo;
        LaPtr->ObjectVdPtr = (void*) TrgtTwpLotPtr;
    }
}
```

bool BModEcoSys::AddResident(BTwpLot* TwpLotPtr, BWrkr* WrkrPtr)

```
{
    // Ensure it is a residential lot.
    if( TwpLotPtr->GetLotDevType() == _ME_LotDevType_UnSpec )
    {
        TwpLotPtr->SetLotDevType( _ME_LotDevType_Residence );
    }

    // The lot must know this Wrkr is a resident.
    // Add the Wrkr to the resident list in the new lot.
    ResAddrPairPtr = TwpLotPtr->ResidentList.AddResident( WrkrPtr->SlotNo, (void*) WrkrPtr );
    TwpLotPtr->ResidentList.SetResColor( ResAddrPairPtr->AddrSlotNo, WrkrPtr->GetDrawingColorPtr()->GetRGBvalue() );
    TwpLotPtr->SetDotColor( ResAddrPairPtr->AddrSlotNo, WrkrPtr->GetDrawingColorPtr()->GetRGBvalue() );
    // The Wrkr must know the lot address of its residence.
    // Now, add the lot address to the Wrkr.
    WrkrPtr->LotPtrPair.PtrPairActiveFlag = _ME_PtrPairActiveFlag_Yes;
    WrkrPtr->LotPtrPair.ObjectVdPtr = (void*) TwpLotPtr;
    WrkrPtr->LotPtrPair.ObjectSlotNo = TwpLotPtr->LotSlotNo;
    WrkrPtr->LotPtrPair.AddrSlotNo = -1;  // Not in a slot.
    // The Wrkr must know which unit it occupies in the residence.
    // Note the UnitNo of this address pair in the Wrkr.
    WrkrPtr->ResUnitNo = ResAddrPairPtr->AddrSlotNo;
}
```

bool BModEcoSys::DelResident(BTwpLot* TwpLotPtr, BWrkr* WrkrPtr)

```
{
    // Get the unit number occupied by this Wrkr.
    ResUnitNo = WrkrPtr->ResUnitNo;
    // Delete this Wrkr from the resident list in this lot.
    TwpLotPtr->ResidentList.DelResident( ResUnitNo );
    TwpLotPtr->ResidentList.SetResColor( ResUnitNo, TwpLotPtr->GetLotBgColor() );
    TwpLotPtr->ComputeLotDotColors();
    // Remove the lot address from the Wrkr.
    WrkrPtr->LotPtrPair.InitBPtrPair();
    // Remove the UnitNo from the Wrkr.
    WrkrPtr->ResUnitNo = -1;

    if( TwpLotPtr->ResidentList.GetNoOfResidents() == 0 )
    {
        TwpLotPtr->SetLotDevType( _ME_LotDevType_UnSpec );
    }
}
```

```
bool BModEcoSys::MoveWrkr( BWrkr* WrkrPtr )
{
    TwpLotPtr = (BTwpLot*) WrkrPtr->LotPtrPair.ObjectVdPtr;
    LotSlotNo = TwpLotPtr->LotSlotNo;
    bResult = FindVacantResidence( TwpLotPtr, &VacantLotPtrPair );
    if( bResult == FALSE ) { return FALSE; }

    // This Wrkr now has a move location.
    TrgtLotSlotNo = VacantLotPtrPair.ObjectSlotNo;
    TrgtTwpLotPtr = Township.GetLotPtr( TrgtLotSlotNo );
    // Record the unit number for statistical reporting purposes.
    WasResUnitNo = WrkrPtr->ResUnitNo;
    // Remove the Wrkr from the residence list of the current lot.
    DelResident( TwpLotPtr, WrkrPtr );
    // This Wrkr is now moved out of that unit, but
    //   remains on the union lists of all Frmrs
    //   in the commuting area.
    RemoveThisWrkrFromUnionLists( WrkrPtr );
    // Now remove the Wrkr from the customer lists.
    RemoveThisWrkrFromCustomerLists( WrkrPtr );
    // Redraw the vacated lot.
    Township.DrawOneLot( LotSlotNo );

    // Add this Wrkr as a resident of the target township lot.
    AddResident( TrgtTwpLotPtr, WrkrPtr );
    // And get this Wrkr on the union list of all
    //   employers in the new commuting area.
    AddThisWrkrToUnionLists( WrkrPtr );
    // And get this Wrkr on the customer list of all
    //   suppliers in the new commuting area.
    AddThisWrkrToCustomerLists( WrkrPtr );
    // Redraw the occupied lot.
    Township.DrawOneLot( TrgtTwpLotPtr->LotSlotNo );
}


// Subsidiary to DoSellInventory.

long BModEcoSys::DoSellInventory( long CurrentFunction )
{
       for( FrmrNo = 0; FrmrNo < NoOfFrmrs; FrmrNo++ )
       {
              FrmrPtr = FrmrList.GetFrmrPtr( FrmrSlotNoList.GetRandomSlotNo() );
```

```
            // Is this Frmr ready to sell inventory?
            MuType = FrmrPtr->FrmrType;
            ValueAvailable = FrmrPtr->InventoryOnHand.MuPool[ MuType ].GetMonetaryValue();
            if( ValueAvailable > 0 )
            {
                  // For this Frmr, canvass all Wrkrs on the
                  //   customer list and make sales pitches.
                  BuildCustomerList( FrmrPtr );      // Only actions if needed.
                  IssueSalesPitches( FrmrPtr );
            }
      }
}
```

bool BModEcoSys::IssueSalesPitches(BFrmr* FrmrPtr)

```
{
   MuPtr = FrmrPtr->GetInventoryOnHandPtr();
   // At this point, we assume the Frmr has mus in inventory
   //   (i.e. materiel units to sell).
   TodaysDate = Township.Age.GetInSeconds();

   ContactSlotNoList.InitBSlotNoList();
   ContactSlotNoList.RsPtr = RsPtr;
   NoOfContacts = FrmrPtr->CustomerList.GetNoOfContacts();
   if( NoOfContacts <= 0 ) return TRUE;

   ContactPtr = FrmrPtr->CustomerList.GetTopContactPtr();
   for( ContactNo = 0; ContactNo < NoOfContacts; ContactNo++ )
   {
      ContactSlotNoList.AddSlotNo( ContactPtr->SlotNo );
      ContactPtr = ContactPtr->NextPtr;
   }

   for( ContactNo = 0; ContactNo < NoOfContacts; ContactNo++ )
   {
      // A Frmr can only make sales pitches while it still has inventory to sell.
      if( FrmrPtr->InventoryOnHand.MuPool[ FrmrPtr->FrmrType ].GetNoOfMu() <= 0 ) return FALSE;

      // The Frmr is still good-to-go.  Make the next contact.
      ContactPtr = ContactPool.GetContactPtr( ContactSlotNoList.GetRandomSlotNo() );

      // Contact the customer.
      if( ContactPtr->Dlc < TodaysDate )
      {
         long TrgtType;
         TrgtType = ContactPtr->AgentType;
```

```
            if( TrgtType == _ME_CI_AgentType_Frmr )
            {   // is a Frmr
                TrgtFrmrPtr = (BFrmr*) ContactPtr->AgentVdPtr;
                // A Frmr for which the value of its Supply mus
                //   exceeds the pre-determined level as per its planning
                //   genes will not choose to proceed.
                if( ( TrgtFrmrPtr->IsPlanningToBuySupply() ) &&
                    ( TrgtFrmrPtr->GetCashOnHand() > 0 ) )
                {   // is a needy Frmr
                    ResultOfSalesPitch = IssueSalesPitch( FrmrPtr, TrgtFrmrPtr );
                    // Mark the contact as completed.
                    ContactPtr->Dlc = TodaysDate;
                    ContactPtr->MtchPtr->Dlc = TodaysDate;
                    if( ResultOfSalesPitch == TRUE )
                    {
                        TrgtFrmrPtr->DateLastSupplied[ MuType ] = TodaysDate;
                    }
                }   // end of 'is a needy Frmr
            }   // end of 'is a Frmr
            else
            {   // is a Wrkr
                WrkrPtr = (BWrkr*) ContactPtr->AgentVdPtr;
                // A Wrkr for which the value of its
                //   Supply MbMEu exceeds the requirements of its
                //   genes will decline to purchase Supply MbMEu.
                if( ( WrkrPtr->IsPlanningToBuySupply() ) &&
                    ( WrkrPtr->GetCashOnHand() > 0 ) )
                {   // Is a needy Wrkr.
                    ResultOfSalesPitch = IssueSalesPitch( FrmrPtr, WrkrPtr );
                    // Mark the contact as completed.
                    ContactPtr->Dlc = TodaysDate;
                    ContactPtr->MtchPtr->Dlc = TodaysDate;
                    if( ResultOfSalesPitch == TRUE )
                    {
                        WrkrPtr->DateLastSupplied[ MuType ] = TodaysDate;
                    }
                }   // end of 'is a needy Wrkr
            }   // end of 'else, is a Wrkr.
        }
    }
}
```

```
bool BModEcoSys::IssueSalesPitch( BFrmr* FrmrPtr, BWrkr* WrkrPtr )
{
```

```
// In this function a Frmr attempts to sell MbMEu to a Wrkr.
MuPtr = FrmrPtr->GetInventoryOnHandPtr();
MuType = FrmrPtr->FrmrType;
TodaysDate = Township.Age.GetInSeconds();

if( MuPtr->GetNoOfMu() <= 0 )
{
    return FALSE;
}

UnitIValue = ComputeUnitPrice( MuPtr->GetIntrinsicValue(), MuPtr->GetNoOfMu() );

// The base prices are now determined.
// The Frmr will make an offer, and the Wrkr will
//   determine if it is acceptable.
FrmrPriceQuoteSellMbMEu = FrmrPtr->GetQuoteSellInvMbMEu( UnitIValue, MuType );
WrkrPriceQuoteBuyMbMEu = WrkrPtr->GetQuoteBuySupdMbMEu( UnitIValue, MuType );

// If the Frmr wants less than the Wrkr is willing to pay, deal.
if( FrmrPriceQuoteSellMbMEu <= WrkrPriceQuoteBuyMbMEu )
{
    // Deal!
    // If you get to here, the Wrkr has decided to make a purchase.
    // The purchase price is the average of the two quotes.  The seller gets more
    //   than he asked for, and the buyer pays less than it offered.
    UnitPrice = ( FrmrPriceQuoteSellMbMEu + WrkrPriceQuoteBuyMbMEu ) / 2.0;

    // Now we must determine how much materiel changes hands.
    // There is an upper limit on what can be sold, like the 40
    //   units of Nrg per colony in PSoup.
    NoOfMuSold = _ME_MaxMbMEuSold;

    // NoOfMuSold cannot be more than the Frmr has available in inventory.
    AvailableMbMEu = FrmrPtr->InventoryOnHand.MuPool[ MuType ].GetNoOfMu();
    if( NoOfMuSold > AvailableMbMEu ) NoOfMuSold = AvailableMbMEu;

    // What can the Wrkr afford?
    AvailableCash = WrkrPtr->GetCashOnHand();

    // Are Cash grants available.
    if( Township.GetCash_Grants() > 0 ) CashGrantsAreAvailable = TRUE;
    else CashGrantsAreAvailable = FALSE;

    // NoOfMuSold cannot be more than what can be afforded.
    if( NoOfMuSold > ( AvailableCash / UnitPrice ) )
    {
```

```
    // The Wrkr has insufficient cash to purchase the entire lot available.
    CashGrantsAreNeeded = TRUE;
    // Apply for grants, if available.
    if( ( Toggles.Cons_CashGrants_Switch == _ME_Cons_CashGrants_Switch_On ) &&
        ( CashGrantsAreAvailable ) )
    {
        // Consumer Cash Grants program is on.
        CashNeeded = NoOfMuSold * UnitPrice;
        Cash_Grant_Allowed = ApplyForWrkr_CashGrants( AvailableCash, CashNeeded );
        if( Cash_Grant_Allowed > 0.0 )
        {
            CashGrantsAreGranted = TRUE;
            DrawWrkr_CashGrants( WrkrPtr, Cash_Grant_Allowed );
            AvailableCash = WrkrPtr->GetCashOnHand();
        }
    }
    NoOfMuSold = Round_As_Mu( AvailableCash / UnitPrice );
}

// Calculate the extended price of this amount of materiel.
ExtendedPrice = Round_As_Cash( UnitPrice * NoOfMuSold );
// Address rounding issue.
while( Is_LT_WT( ( WrkrPtr->GetCashOnHand() - ExtendedPrice ), 0 ) )
{
    ExtendedPrice -= ( 1 / _ME_Rounding_Factor_Cash );
}

// Draw the MbMEu from the correct pool of MbMEu.
BillOfLading.FillRequisitionForm( MuType, NoOfMuSold );
FrmrPtr->InventoryOnHand.MuPool[ MuType ].RequisitionMateriel( &BillOfLading );

// Transfer product to the Wrkr via a bill of lading.
// But adjust the monetary value to equal the extended price.
OldMbMuMValue = BillOfLading.GetMbMuMValue();
OldMbEuMValue = BillOfLading.GetMbEuMValue();
OldMValue = BillOfLading.GetMonetaryValue();
if( OldMValue == 0 )
{
    NewMbMuMValue = 0;
}
else
{
    NewMbMuMValue = Round_As_Cash( OldMbMuMValue * ( ExtendedPrice / OldMValue ) );
}
NewMbEuMValue = ExtendedPrice - NewMbMuMValue;
BillOfLading.SetMbMuMValue( NewMbMuMValue );
```

```
        BillOfLading.SetMbEuMValue( NewMbEuMValue );
        WrkrPtr->SupplyMuOnHand.MuPool[ MuType ].StoreMateriel( &BillOfLading );
        // Make a payment.
        FrmrPtr->SetCashOnHand( FrmrPtr->GetCashOnHand() + ExtendedPrice );
        WrkrPtr->SetCashOnHand( WrkrPtr->GetCashOnHand() - ExtendedPrice );
        // Record the time and type of transaction.
        WrkrPtr->DateLastSupplied[ MuType ] = Township.Age.GetInSeconds();
    }
}
```

bool BModEcoSys::IssueSalesPitch(BFrmr* FrmrPtr, BFrmr* TrgtFrmrPtr)

```
{
    // In this function a Frmr attempts to sell MbMEu to another Frmr.
    MuPtr = FrmrPtr->GetInventoryOnHandPtr();
    MuType = FrmrPtr->FrmrType;
    TodaysDate = Township.Age.GetInSeconds();

    if( MuPtr->GetNoOfMu() <= 0 )
    {
        // No Deal!  Write Statistics
        return FALSE;
    }

    UnitIValue = ComputeUnitPrice( MuPtr->GetIntrinsicValue(), MuPtr->GetNoOfMu() );

    // The base prices are now determined.
    // The Frmr will make an offer, and the target Frmr will
    //   determine if it is acceptable.
    FrmrPriceQuoteSellMbMEu = FrmrPtr->GetQuoteSellInvMbMEu( UnitIValue, MuType );
    TrgtFrmrPriceQuoteBuyMbMEu = TrgtFrmrPtr->GetQuoteBuySupdMbMEu( UnitIValue, MuType );

    // If the Frmr wants less than the Wrkr is willing to pay, deal.
    if( FrmrPriceQuoteSellMbMEu <= TrgtFrmrPriceQuoteBuyMbMEu )
    {
        // Deal!
        // If you get to here, the Wrkr has decided to make a purchase.
        // The purchase price is the average of the two quotes.  The seller gets more
        //   than he asked for, and the buyer pays less than it offered.
        UnitPrice = ( FrmrPriceQuoteSellMbMEu + TrgtFrmrPriceQuoteBuyMbMEu ) / 2.0;

        // Now we must determine how much materiel changes hands.
        // There is an upper limit on what can be sold, like the 40
        //   units of Nrg per colony in PSoup.
        NoOfMuSold = _ME_MaxMbMEuSold;
```

```
// NoOfMuSold cannot be more than the selling Frmr has available in inventory.
AvailableMbMEu = FrmrPtr->InventoryOnHand.MuPool[ MuType ].GetNoOfMu();
if( NoOfMuSold > AvailableMbMEu ) NoOfMuSold = AvailableMbMEu;

// What can the target consumer Frmr afford?
AvailableCash = TrgtFrmrPtr->GetCashOnHand();

if( Township.GetCash_Grants() > 0.0 ) CashGrantsAreAvailable = TRUE;
else CashGrantsAreAvailable = FALSE;

// NoOfMuSold cannot be more than what can be afforded.
if( NoOfMuSold > ( AvailableCash / UnitPrice ) )
{
    // The TrgtFrmr has insufficient cash to purchase the entire lot available.
    CashGrantsAreNeeded = TRUE;
    // Apply for grants, if available.
    if( ( Toggles.Cons_CashGrants_Switch == _ME_Cons_CashGrants_Switch_On ) &&
        ( CashGrantsAreAvailable ) )
    {
        // Consumer Cash Grants program is on.

        CashNeeded = Round_As_Cash( NoOfMuSold * UnitPrice );
        Cash_Grant_Allowed = ApplyForFrmr_CashGrants( AvailableCash, CashNeeded );
        if( Cash_Grant_Allowed > 0.0 )
        {
            CashGrantsAreGranted = TRUE;
            DrawFrmr_CashGrants( TrgtFrmrPtr, Cash_Grant_Allowed );
            AvailableCash = TrgtFrmrPtr->GetCashOnHand();
        }
    }
    NoOfMuSold = Round_As_Mu( AvailableCash / UnitPrice );
}
// Calculate the extended price of this amount of materiel.
ExtendedPrice = Round_As_Cash( UnitPrice * NoOfMuSold );
// Address rounding issue.
while( Is_LT_WT( ( TrgtFrmrPtr->GetCashOnHand() - ExtendedPrice ), 0 ) )
{
    ExtendedPrice -= ( 1 / _ME_Rounding_Factor_Cash );
}

// Draw the MbMEu from the correct pool of MbMEu.
BillOfLading.FillRequisitionForm( MuType, NoOfMuSold );
FrmrPtr->InventoryOnHand.MuPool[ MuType ].RequisitionMateriel( &BillOfLading );

// Transfer product to the Frmr via a bill of lading.
// But adjust the monetary value to equal the extended price.
```

```
        OldMbMuMValue = BillOfLading.GetMbMuMValue();
        OldMbEuMValue = BillOfLading.GetMbEuMValue();
        OldMValue = BillOfLading.GetMonetaryValue();
        if( OldMValue == 0.0 )
        {
            NewMbMuMValue = 0;
        }
        else
        {
            NewMbMuMValue = OldMbMuMValue * ( ExtendedPrice / OldMValue );
        }
        NewMbEuMValue = ExtendedPrice - NewMbMuMValue;
        BillOfLading.SetMbMuMValue( NewMbMuMValue );
        BillOfLading.SetMbEuMValue( NewMbEuMValue );
        TrgtFrmrPtr->SupplyMuOnHand.MuPool[ MuType ].StoreMateriel( &BillOfLading );

        // Make a payment.
        FrmrPtr->SetCashOnHand( FrmrPtr->GetCashOnHand() + ExtendedPrice );
        TrgtFrmrPtr->SetCashOnHand( TrgtFrmrPtr->GetCashOnHand() - ExtendedPrice );

        // Record the time and type of transaction.
        TrgtFrmrPtr->DateLastSupplied[ MuType ] = Township.Age.GetInSeconds();
    }
}
```

// Subsidiary to DoConsumeSupplyMbMEu.

long BModEcoSys::DoConsumeSupplyMbMEu(long CurrentFunction)

```
{
        TodaysDate = Township.Age.GetInSeconds();
        for( WrkrNo = 0; WrkrNo < NoOfWrkrs; WrkrNo++ )
        {
                WrkrPtr = WrkrList.GetWrkrPtr( WrkrSlotNoList.GetRandomSlotNo() );

                // MANDATORY SCRAPPING
                // For each Wrkr, daily life requires consumption (scrapping) of MbMEu.
                MuType = GetMuTypeToConsume( WrkrPtr );
                MaxMbMEuToSpend = WrkrPtr->Genes.MPT;     // Materiel Per Tick.
                MbMEuOnHand = WrkrPtr->SupplyMuOnHand.MuPool[ MuType ].GetNoOfMu();
                if( MbMEuOnHand < MaxMbMEuToSpend )
                {
                        MbMEuToSpend = MbMEuOnHand;
                        WrkrPtr->HasStarved_Flag = _ME_HasStarved_Flag_Yes;
                }
```

```
		else MbMEuToSpend = MaxMbMEuToSpend;
		MaterielWasteped.InitBMaterielUnit();
		MaterielWasteped.FillRequisitionForm( MuType, MbMEuToSpend );
		WrkrPtr->MoveSupplyToWaste( MuType, &MaterielWasteped );
		StatsPack.TakeSp04SwRdg( TodaysDate, WrkrPtr, &MaterielWasteped );
		Sounds.WasteSupd();
		// DISCRETIONARY SCRAPPING
		// For each Wrkr, additional units may be scrapped to enable work.
		if( WrkrPtr->IsPlanningToWasteAdditional() )
		{
			// This mode of recycling MbEu can be activated by an ISR switch.
			MuType = GetMuTypeToConsume( WrkrPtr );
			MaxMbMEuToSpend = WrkrPtr->Genes.MPT;	// Materiel Per Tick.
			MbMEuOnHand = WrkrPtr->SupplyMuOnHand.MuPool[ MuType ].GetNoOfMu();
			if( MbMEuOnHand < MaxMbMEuToSpend ) MbMEuToSpend = MaxMbMEuToSpend;
			MaterielWasteped.FillRequisitionForm( MuType, MbMEuToSpend );
			WrkrPtr->MoveSupplyToWaste( MuType, &MaterielWasteped );
			StatsPack.TakeSp04SwRdg( TodaysDate, WrkrPtr, &MaterielWasteped );
			Sounds.WasteSupd();
		}
	}

	for( FrmrNo = 0; FrmrNo < NoOfFrmrs; FrmrNo++ )
	{
		FrmrPtr = FrmrList.GetFrmrPtr( FrmrSlotNoList.GetRandomSlotNo() );

		// MANDATORY CONSUMPTION
		// For each Frmr, daily life requires consumption (scrapping) of MbMEu.
		MuType = GetMuTypeToConsume( FrmrPtr );
		MbMEuOnHand = FrmrPtr->SupplyMuOnHand.MuPool[ MuType ].GetNoOfMu();
		// Default mandatory consumption (scrapping) level.
		MaxMbMEuToSpend = FrmrPtr->Genes.MPT;		// Materiel Per Tick.
		if( MbMEuOnHand < MaxMbMEuToSpend )
		{
			MbMEuToSpend = MbMEuOnHand;
			FrmrPtr->HasStarved_Flag = _ME_HasStarved_Flag_Yes;
		}
		else MbMEuToSpend = MaxMbMEuToSpend;
		MaterielWasteped.FillRequisitionForm( MuType, MbMEuToSpend );
		FrmrPtr->MoveSupplyToWaste( MuType, &MaterielWasteped );
		StatsPack.TakeSp04SwRdg( TodaysDate, FrmrPtr, &MaterielWasteped );
		Sounds.WasteSupd();
		// DISCRETIONARY CONSUMPTION
		if( FrmrPtr->IsPlanningToWasteAdditional() )
		{
			// Allow more consumption (scrapping) if the Frmr is an active employer.
```

```
                    // Start with one plus hires.  This is a look-ahead amount
                    //   that enables a Frmr to hire more in the next round if
                    //   the economy is good.
                    // This mode of recycling MbEu can be activated by a MbEuRecycle switch.
                    AdditionalConsumption = FrmrPtr->NoOfHires + 1;
                    if( AdditionalConsumption > _ME_HiresMax_PostInd )
                           AdditionalConsumption = _ME_HiresMax_PostInd;
                    // Expense is mandatory consumption plus consumption towards hiring.
                    MaxMbMEuToSpend = _ME_DailyWorkRate_MbEu * AdditionalConsumption;
                    // This enables an active Frmr to consume at a higher rate,
                    //   thereby producing MbEu (Infra units) at a higher rate
                    //   allowing future hiring at a higher rate, up to the maximum
                    //   hiring rate.
                    // This is a type of investment in the future.  The net worth
                    //   drops, because the MbEu are not counted, but the ability
                    //   to participate in the market is increased.  The risk is that
                    //   Supply units might be consumed too fast, and the Frmr could
                    //   starve for lack of mandatory consumption (mandatory scrapping).
                    if( MbMEuOnHand < MaxMbMEuToSpend ) MbMEuToSpend = MbMEuOnHand;
                    else MbMEuToSpend = MaxMbMEuToSpend;
                    MaterielWasteped.FillRequisitionForm( MuType, MbMEuToSpend );
                    FrmrPtr->MoveSupplyToWaste( MuType, &MaterielWasteped );
             }
      }
}

long BModEcoSys::GetMuTypeToConsume( BWrkr* WrkrPtr )
{
   MuType = _ME_MU_MuType_Wood;
}

long BModEcoSys::GetMuTypeToConsume( BFrmr* FrmrPtr )
{
   MuType = _ME_MU_MuType_Wood;
}

bool ConsumeMateriel( BMaterielUnit* SupdMuPtr, BWrkr* WrkrPtr )
{
   // SupdMuPtr points to a bill of lading containing Supply MbMEu
   //   which are to be scrapped.  The work-based value must go
   //   into the MbEu of the Wrkr, building up their health
   //   and allowing them to perform labours during their next job.
   //   The resource-based value must go into the scrap bin, to be
   //   sold to the township's materiel manager as scrap for whatever
```

```
    //   they can get.

    // Establish the pointers.
    MuType = SupdMuPtr->GetMuType();
    TuPtr = &WrkrPtr->MbEu;
    SuPtr = &WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ];

    // Unpack the data.
    NoOfMuToMove = SupdMuPtr->GetNoOfMu();
    MbEuMValue = SupdMuPtr->GetMbEuMValue();
    MbEuIValue = SupdMuPtr->GetMbEuIValue();
    MbMuMValue = SupdMuPtr->GetMbMuMValue();
    MbMuIValue = SupdMuPtr->GetMbMuIValue();

    // Put the scrap into the scrapped bin.
    SuPtr->SetMbMuMValue( MbMuMValue + SuPtr->GetMbMuMValue() );
    SuPtr->SetMbMuIValue( MbMuIValue + SuPtr->GetMbMuIValue() );

    // Put the work-based value into the time units bin.
    TuPtr->SetMbEuMValue( MbEuMValue + TuPtr->GetMbEuMValue() );
    TuPtr->SetMbEuIValue( MbEuIValue + TuPtr->GetMbEuIValue() );
}
```

bool ConsumeMateriel(BMaterielUnit* SupdMuPtr, BFrmr* FrmrPtr)

```
{
    // SupdMuPtr points to a bill of lading containing Supply MbMEu
    //   which are to be scrapped.  The work-based value must go
    //   into the MbEu of the Frmr, building up their capability
    //   and allowing them to support labours during their next job.
    //   The resource-based value must go into the scrap bin, to be
    //   sold to the township's materiel manager as scrap for whatever
    //   they can get.

    // Establish the pointers.
    MuType = SupdMuPtr->GetMuType();
    IuPtr = &FrmrPtr->MbEu;
    SuPtr = &FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ];

    // Unpack the data.
    NoOfMuToMove = SupdMuPtr->GetNoOfMu();
    MbEuMValue = SupdMuPtr->GetMbEuMValue();
    MbEuIValue = SupdMuPtr->GetMbEuIValue();
    MbMuMValue = SupdMuPtr->GetMbMuMValue();
    MbMuIValue = SupdMuPtr->GetMbMuIValue();
```

```
    // Put the scrap into the scrapped bin.
    SuPtr->SetMbMuMValue( MbMuMValue + SuPtr->GetMbMuMValue() );
    SuPtr->SetMbMuIValue( MbMuIValue + SuPtr->GetMbMuIValue() );

    // Put the work-based value into the time units bin.
    IuPtr->SetMbEuMValue( MbEuMValue + IuPtr->GetMbEuMValue() );
    IuPtr->SetMbEuIValue( MbEuIValue + IuPtr->GetMbEuIValue() );
}
```

// Subsidiary to DoSellWasteMbMu.

long BModEcoSys::DoSellWasteMbMu(long CurrentFunction)

```
{
        for( WrkrNo = 0; WrkrNo < NoOfWrkrs; WrkrNo++ )
        {
                WrkrPtr = WrkrList.GetWrkrPtr( WrkrSlotNoList.GetRandomSlotNo() );
                SellWrkrWaste( WrkrPtr );
        }

        for( FrmrNo = 0; FrmrNo < NoOfFrmrs; FrmrNo++ )
        {
                FrmrPtr = FrmrList.GetFrmrPtr( FrmrSlotNoList.GetRandomSlotNo() );
                SellFrmrWaste( FrmrPtr );
        }
}
```

bool BModEcoSys::SellWrkrWaste(BWrkr* WrkrPtr)

```
{
    // In this function a Wrkr sells scrap to the scrap dealer.

    TodaysDate = Township.Age.GetInSeconds();
    for( MuType = 0; MuType < _ME_MU_NoOfMuTypes; MuType++ )
    {
        if( WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu() > 0 )
        {
            MbMuUnitMValue = ComputeUnitPrice( WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetMbMuMValue(),
                                    WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu() );
            MbMuUnitIValue = ComputeUnitPrice( WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetMbMuIValue(),
                                    WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu() );

            if( Toggles.Global_Pm_Switch == _ME_Global_PriceMode_Generic )
            {
                MbMzPtr = &Township.MbMu_MuStore;
```

```
    }
    else
    {
        MbMzPtr = &Township.MbMzPool.MuPool[ MuType ];
    }

    // Waste is sold to the township's materiel manager at the values determined by
    //   the underlying resource.  The 'added value' associated with the Supply
    //   status of the mu was removed during consumption (scrapping).
    WrkrPriceQuoteSellWaste = WrkrPtr->GetQuoteSellWaste( MbMuUnitIValue, MuType );
    TownshipPriceQuoteBuyWaste = Township.GetQuoteBuyWaste( MbMuUnitIValue, MuType );

    // If the Wrkr wants less than the township is willing to pay, deal.
    if( WrkrPriceQuoteSellWaste <= TownshipPriceQuoteBuyWaste )
    {
        if( Toggles.Global_Db_Switch == _ME_DataBank_Switch_On )
        {
            DataBankMe.SampleWasteHiLo( WrkrPriceQuoteSellWaste, TownshipPriceQuoteBuyWaste );
        }
        // If you get to here, the sale is made.
        // The unit price is the average of the two quotes.  The consumer gets more
        //   than he asked for, and the township pays less than it offered.
        UnitPrice = ( TownshipPriceQuoteBuyWaste + WrkrPriceQuoteSellWaste ) / 2.0;

        // Now we must determine how much materiel changes hands.
        // There is an upper limit on what can be sold, like the 40
        //   units of Nrg per colony in PSoup.

        //NoOfMuSold = Round_As_Mu( _ME_MaxMbMEuSold / 4 );
        NoOfMuSold = WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu();

        if( Toggles.MMgr_QuEasing_Switch == _ME_MMgr_QuEasing_Switch_Off )
        {
            // What can the township afford?
            AvailableCash = Township.GetCashOnHand();

            // NoOfMuSold cannot be more than what can be afforded.
            if( NoOfMuSold > ( AvailableCash / UnitPrice ) )
                NoOfMuSold = Round_As_Mu( AvailableCash / UnitPrice );
        }

        // NoOfMuSold cannot be more than the Wrkr has on hand.
        AvailableMbMEu = WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu();
        if( NoOfMuSold > AvailableMbMEu ) NoOfMuSold = AvailableMbMEu;

        // Calculate the extended price of this amount of materiel.
```

```
            ExtendedPrice = Round_As_Cash( UnitPrice * NoOfMuSold );

            // Exchange the cash.
            Township.SetCashOnHand( Township.GetCashOnHand() - ExtendedPrice );  // Can be < 0
            WrkrPtr->SetCashOnHand( WrkrPtr->GetCashOnHand() + ExtendedPrice );

            // Waste is transferred to the RecycledMbMzPool.
            MbMuBillOfLading.FillRequisitionForm( MuType, NoOfMuSold );
            WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].RequisitionWaste( &MbMuBillOfLading );
            MbMzPtr->RecycleWaste( &MbMuBillOfLading );
        }
    }
  }
}
```

bool BModEcoSys::SellFrmrWaste(BFrmr* FrmrPtr)

```
{
  // In this function a Frmr sells scrap to the scrap dealer.
  TodaysDate = Township.Age.GetInSeconds();
  for( MuType = 0; MuType < _ME_MU_NoOfMuTypes; MuType++ )
  {
    if( FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu() > 0 )
    {
      MbMuUnitMValue = ComputeUnitPrice( FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetMbMuMValue(),
                                         FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu() );
      MbMuUnitIValue = ComputeUnitPrice( FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetMbMuIValue(),
                                         FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu() );

      if( Toggles.Global_Pm_Switch == _ME_Global_PriceMode_Generic )
      {
        MbMzPtr = &Township.MbMu_MuStore;
      }
      else
      {
        MbMzPtr = &Township.MbMzPool.MuPool[ MuType ];
      }

      // Waste is sold to the township's materiel manager at the values determined by
      //   the underlying resource.  The 'added value' associated with the Supply
      //   status of the mu was removed during consumption (scrapping).
      FrmrPriceQuoteSellWaste = FrmrPtr->GetQuoteSellWasteMbMu( MbMuUnitIValue, MuType );
      TownshipPriceQuoteBuyWaste = Township.GetQuoteBuyWaste( MbMuUnitIValue, MuType );

      // If the Frmr wants less than the township is willing to pay, deal.
      if( FrmrPriceQuoteSellWaste <= TownshipPriceQuoteBuyWaste )
```

```
            {
                // If you get to here, the sale is made.
                // The unit price is the average of the two quotes.  The consumer gets more
                //   than he asked for, and the township pays less than it offered.
                UnitPrice = ( TownshipPriceQuoteBuyWaste + FrmrPriceQuoteSellWaste ) / 2.0;

                // Now we must determine how much materiel changes hands.
                // There is an upper limit on what can be sold, like the 40
                //   units of Nrg per colony in PSoup.

                //NoOfMuSold = Round_As_Mu( _ME_MaxMbMEuSold / 4 );
                NoOfMuSold = FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu();

                if( Toggles.MMgr_QuEasing_Switch == _ME_MMgr_QuEasing_Switch_Off )
                {
                    // What can the township afford?
                    AvailableCash = Township.GetCashOnHand();

                    // NoOfMuSold cannot be more than what can be afforded.
                    if( NoOfMuSold > ( AvailableCash / UnitPrice ) )
                        NoOfMuSold = Round_As_Mu( AvailableCash / UnitPrice );
                }
                // NoOfMuSold cannot be more than the Frmr has on hand.
                AvailableMbMEu = FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu();
                if( NoOfMuSold > AvailableMbMEu ) NoOfMuSold = AvailableMbMEu;
                // Calculate the extended price of this amount of materiel.
                ExtendedPrice = Round_As_Cash( UnitPrice * NoOfMuSold );

                // Exchange the cash.
                Township.SetCashOnHand( Township.GetCashOnHand() - ExtendedPrice );  // Can be < 0
                FrmrPtr->SetCashOnHand( FrmrPtr->GetCashOnHand() + ExtendedPrice );

                // Waste is transferred to the RecycledMbMzPool.
                MbMuBillOfLading.FillRequisitionForm( MuType, NoOfMuSold );
                FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].RequisitionWaste( &MbMuBillOfLading );
                MbMzPtr->RecycleWaste( &MbMuBillOfLading );
            }
        }
    }
}
```

// Subsidiary to DoBuyRecycledMbMu.

long BModEcoSys::DoBuyRecycledMbMu(long CurrentFunction)

```
{
	for( FrmrNo = 0; FrmrNo < NoOfFrmrs; FrmrNo++ )
	{
		FrmrPtr = FrmrList.GetFrmrPtr( FrmrSlotNoList.GetRandomSlotNo() );
		// A Frmr for which the value of its raw materiel exceeds
		//   a planning threshold, a pre-determined fraction of its
		//   effective net worth, will not choose to purchase more raw
		//   materiel.  This prevents a Frmr from investing all
		//   of its net worth into a single type of supplies.
		// Additionally, it must have some CashOnHand.
		if( ( FrmrPtr->IsPlanningToBuyRecycled() ) &&
			( FrmrPtr->GetCashOnHand() > 0 ) &&
			( Township.GetNoOfMbMu( FrmrPtr->FrmrType ) > 0.1 ) )
		{
			bResult = BuyRecycledMbMu( FrmrPtr );
		}
	}
}
```

bool BModEcoSys::BuyRecycledMbMu(BFrmr* FrmrPtr)

```
{
    // In this function a Frmr buys appropriate raw
    //   mus from the township.

    MuType = FrmrPtr->FrmrType;
    if( Toggles.Global_Pm_Switch == _ME_Global_PriceMode_Generic )
    {
        MbMzPtr = &Township.MbMu_MuStore;
    }
    else
    {
        MbMzPtr = &Township.MbMzPool.MuPool[ MuType ];
    }
    TodaysDate = Township.Age.GetInSeconds();
    if( MbMzPtr->GetNoOfMu() <= 0 )
    {
        if( MbMzPtr->GetNoOfMu() < 0 ) MbMzPtr->SetNoOfMu( 0 );
        return FALSE;
    }

    MbMuUnitIValue = ComputeUnitPrice( MbMzPtr->GetMbMuIValue(), MbMzPtr->GetNoOfMu() );
```

```
// The base prices now determined.
// The Frmr will make an offer, and the township will
//   determine if it is acceptable.
FrmrPriceQuoteBuyRecycledMbMu = FrmrPtr->GetQuoteBuyRecycledMbMu( MbMuUnitIValue, MuType );
TownshipPriceQuoteSellRecycledMbMu = Township.GetQuoteSellRecycledMbMu( MbMuUnitIValue, MuType );
// If the township wants more than the Frmr is willing to pay, no deal.
if( TownshipPriceQuoteSellRecycledMbMu <= FrmrPriceQuoteBuyRecycledMbMu )
{
    if( Toggles.Global_Db_Switch == _ME_DataBank_Switch_On )
    {
        DataBankMe.SampleRecycledHiLo( TownshipPriceQuoteSellRecycledMbMu, FrmrPriceQuoteBuyRecycledMbMu );
    }
    // If you get to here, the Frmr has decided to make a purchase.
    // The purchase price is the average of the two quotes.  The seller gets more
    //   than he asked for, and the buyer pays less than it offered.
    UnitPrice = ( TownshipPriceQuoteSellRecycledMbMu + FrmrPriceQuoteBuyRecycledMbMu ) / 2.0;

    // Now we must determine how much materiel changes hands.
    // Each day the Frmr can buy enough materiel for
    //   four residences to live on for 10 ticks. (= 4x40)
    // TODO: adjust this when factories activated.
    // There is a maximum allowed per transaction.  It is like the
    //   40 Nrg in a colony in PSoup.  Enough for the workers.
    NoOfMuSold = _ME_MaxMbMEuSold * _ME_HiresMax_PostInd;
    // What can the Frmr afford?
    AvailableCash = FrmrPtr->GetCashOnHand();
    // NoOfMuSold cannot be more than what can be afforded.
    if( NoOfMuSold > ( AvailableCash / UnitPrice ) )
        NoOfMuSold = Round_As_Mu( AvailableCash / UnitPrice );
    // NoOfMuSold cannot be more than the township has available.
    if( NoOfMuSold > MbMzPtr->GetNoOfMu() )
        NoOfMuSold = MbMzPtr->GetNoOfMu();
    // Calculate the extended price of this amount of materiel.
    ExtendedPrice = Round_As_Cash( UnitPrice * NoOfMuSold );
    // Address rounding issue.
    while( Is_LT_WT( ( FrmrPtr->GetCashOnHand() - ExtendedPrice ), 0 ) )
    {
        ExtendedPrice -= ( 1 / _ME_Rounding_Factor_Cash );
    }

    // Draw the MbMEu from the correct pool of MbMEu.
    MbMuBillOfLading.FillRequisitionForm( MuType, NoOfMuSold );
    MbMzPtr->RequisitionRecycledMbMu( &MbMuBillOfLading );
    // Transfer product to the Frmr via a bill of lading.
    FrmrPtr->RecycledMbMzOnHand.MuPool[ MuType ].StoreRecycledMbMu( &MbMuBillOfLading );
```

```
        // Make a payment.
        Township.SetCashOnHand( Township.GetCashOnHand() + ExtendedPrice );
        FrmrPtr->SetCashOnHand( FrmrPtr->GetCashOnHand() - ExtendedPrice );
    }
}
```

// Subsidiary to DoAgentsRepro.

long BModEcoSys::DoAgentsRepro(long CurrentFunction)

```
{
        for( FrmrNo = 0; FrmrNo < NoOfFrmrs; FrmrNo++ )
        {
                FrmrPtr = FrmrList.GetFrmrPtr( FrmrSlotNoList.GetRandomSlotNo() );
                // A Frmr will reproduce if it is:
                //   - Experienced enough  (Age >= than its ART).
                //   - Financially sound enough (NetWorth >= its CRT).
                if( FrmrPtr->GetReproductiveStatus() == _ME_ReproStatus_MH )
                {
                        ReproduceFrmr( FrmrPtr );
                }
        }

        for( WrkrNo = 0; WrkrNo < NoOfWrkrs; WrkrNo++ )
        {
                WrkrPtr = WrkrList.GetWrkrPtr( WrkrSlotNoList.GetRandomSlotNo() );
                // A Wrkr will reproduce if it is:
                //   - Experienced enough  (Age > than its ART).
                //   - Financially sound enough (NetWorth >= its CRT).
                if( WrkrPtr->GetReproductiveStatus() == _ME_ReproStatus_MH )
                {
                        ReproduceWrkr( WrkrPtr );
                }
        }
}
```

bool BModEcoSys::ReproduceFrmr(BFrmr* FrmrPtr)

```
{
    ThisLotPtr = (BTwpLot*) FrmrPtr->LotPtrPair.ObjectVdPtr;
    // If there is no empty lot available within the commuting
    //   area, the second daughter cannot find a location
    //   and must die.  Find a vacant lot near this lot.
    D2HasALot = FindVacantLot( ThisLotPtr, &VacantLotPtrPair );
    if( D2HasALot == FALSE ) return FALSE;
```

```
// We assume that the Frmr is ready to reproduce.
D1Ptr = FrmrList.AddFrmr( IncNextAgentSerNo() );
if( D1Ptr == NULL )
{
    return FALSE;
}
D2Ptr = FrmrList.AddFrmr( IncNextAgentSerNo() );
if( D2Ptr == NULL )
{
    // Release the D1 slot in the WrkrList.
    FrmrList.DelFrmr( D1Ptr->SlotNo );
    return FALSE;
}

// AddFrmr assigns a few attrributes as follows:
// - SlotNo
// - FrmrSerNo
// - Dynasty
// - PrevPtr, PrevSlotNo, NextPtr, NextSlotNo

// THE REPRODUCTION IS DEFINITELY ON, START THE PROCESS.

FrmrPtr->D1SerNo = D1Ptr->FrmrSerNo;
FrmrPtr->D2SerNo = D2Ptr->FrmrSerNo;

// The parent Frmr must withdraw from society before it
//   splits to form two daughters; D1 and D2.
RemoveThisFrmrFromSupplierLists( FrmrPtr );
RemoveThisFrmrFromEmployerLists( FrmrPtr );
RemoveThisFrmrFromCustomerLists( FrmrPtr );

// The parent's genes must be removed from the average genes in the Materiel Manager.
ExcludeFromSums( FrmrPtr );

// LotPtrPair is a potential problem for a new business.
// Location, Location, Location!!!
// D1 gets parent's location.  Point the Frmr to the lot.
D1Ptr->LotPtrPair = FrmrPtr->LotPtrPair;
// Now point the lot to the Frmr.
D1LotPtr = (BTwpLot*) D1Ptr->LotPtrPair.ObjectVdPtr;
D1LotPtr->SetFrmrPtrPair( D1Ptr->SlotNo, (void*) D1Ptr, _ME_PtrPairActiveFlag_Yes );
// Set the color.
D1Ptr->TransmitRsPtr( RsPtr );
D1Ptr->ComputeFrmrsOwnColor();
//D1Ptr->DrawingColor = D1Ptr->FrmrsOwnColor;
```

```
{
    // Determine the lot delevopment type.
    D2Ptr->FrmrType = FrmrPtr->FrmrType;
    //D2Ptr->FrmrType = GenerateRandomFrmrType();
    // D2 gets vacant location.  Point the Frmr to the lot.
    D2Ptr->LotPtrPair = VacantLotPtrPair;
    // Now point the lot to the Frmr.
    D2LotPtr = (BTwpLot*) D2Ptr->LotPtrPair.ObjectVdPtr;
    D2LotPtr->SetFrmrPtrPair( D2Ptr->SlotNo, (void*) D2Ptr, _ME_PtrPairActiveFlag_Yes );
    // Set the color.
    D2Ptr->TransmitRsPtr( RsPtr );
    D2Ptr->ComputeFrmrsOwnColor();
    // Set the lot development type.
    D2LotPtr->SetLotDevType( FrmrPtr->FrmrType );
    D2LotPtr->Location.SetWireFrame();
}

// Go through the remaining attributes in order and fill them in.
D1Ptr->FrmrType = FrmrPtr->FrmrType;
D1Ptr->MaSerNo = FrmrPtr->FrmrSerNo;
FrmrPtr->D1SerNo = D1Ptr->FrmrSerNo;
D1Ptr->D1SerNo = -1;
D1Ptr->D2SerNo = -1;
D1Ptr->Generation = FrmrPtr->Generation + 1;
D1Ptr->ReproStatus = _ME_ReproStatus_IH;
D1Ptr->EconomicStatus = _ME_EconomicStatus_PostInd;
D1Ptr->NoOfHiresMax = _ME_HiresMax_PostInd;
D1Ptr->NoOfHires = 0;
D1Ptr->DateLastSupplied[ _ME_MU_MuType_Wood ] = Township.Age.GetInSeconds() - 1;
D1Ptr->DateLastSupplied[ _ME_MU_MuType_Food ] = Township.Age.GetInSeconds() - 1;
D1Ptr->DateLastSupplied[ _ME_MU_MuType_Crafts ] = Township.Age.GetInSeconds() - 1;
D1Ptr->DateLastSupplied[ _ME_MU_MuType_Goods ] = Township.Age.GetInSeconds() - 1;
D1Ptr->HasStarved_Flag = -1;
D1Ptr->Age.SetInSeconds( 0 );
D1Ptr->BirthDate = Township.Age;
D1Ptr->SetCashOnHand( FrmrPtr->GetCashOnHand() / 2.0 );

for( MuType = 0; MuType < _ME_MU_NoOfMuTypes; MuType++ )
{
    MbMuBillOfLading.ZeroAllValues();
    MbMuBillOfLading.SetMuType( MuType );
    MbMuBillOfLading.SetNoOfMu( FrmrPtr->RecycledMbMzOnHand.MuPool[ MuType ].GetNoOfMu() / 2.0 );
    FrmrPtr->RecycledMbMzOnHand.MuPool[ MuType ].RequisitionRecycledMbMu( &MbMuBillOfLading );
    D1Ptr->RecycledMbMzOnHand.MuPool[ MuType ].StoreRecycledMbMu( &MbMuBillOfLading );
```

```
    BillOfLading.ZeroAllValues();
    BillOfLading.SetMuType( MuType );
    BillOfLading.SetNoOfMu( FrmrPtr->InventoryOnHand.MuPool[ MuType ].GetNoOfMu() / 2.0 );
    FrmrPtr->InventoryOnHand.MuPool[ MuType ].RequisitionMateriel( &BillOfLading );
    D1Ptr->InventoryOnHand.MuPool[ MuType ].StoreMateriel( &BillOfLading );

    BillOfLading.ZeroAllValues();
    BillOfLading.SetMuType( MuType );
    BillOfLading.SetNoOfMu( FrmrPtr->SupplyMuOnHand.MuPool[ MuType ].GetNoOfMu() / 2.0 );
    FrmrPtr->SupplyMuOnHand.MuPool[ MuType ].RequisitionMateriel( &BillOfLading );
    D1Ptr->SupplyMuOnHand.MuPool[ MuType ].StoreMateriel( &BillOfLading );

    MbMuBillOfLading.ZeroAllValues();
    MbMuBillOfLading.SetMuType( MuType );
    MbMuBillOfLading.SetNoOfMu( FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu() / 2.0 );
    FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].RequisitionWaste( &MbMuBillOfLading );
    D1Ptr->WasteMbMzOnHand.MuPool[ MuType ].RecycleWaste( &MbMuBillOfLading );
}

MbEuBillOfLading.ZeroAllValues();
MbEuBillOfLading.SetMuType( _ME_MU_MuType_MbEu );
MbEuBillOfLading.SetNoOfMu( FrmrPtr->MbEu.GetNoOfMu() / 2.0 );
FrmrPtr->MbEu.RequisitionHours( &MbEuBillOfLading );
D1Ptr->MbEu.ZeroAllValues();
D1Ptr->MbEu.StoreHours( &MbEuBillOfLading );

D1Ptr->Genes = FrmrPtr->Genes;
D1Ptr->Genes.MutateGenes( &ERules, &Sounds );

IncludeInSums( D1Ptr );

D1Ptr->EmploymentHistory.InitBEmploymentHistory();
D1Ptr->TogglesPtr = FrmrPtr->TogglesPtr;

if( D2HasALot )
{
    D2Ptr->MaSerNo = FrmrPtr->FrmrSerNo;
    FrmrPtr->D2SerNo = D2Ptr->FrmrSerNo;
    D2Ptr->D1SerNo = -1;
    D2Ptr->D2SerNo = -1;
    D2Ptr->Generation = FrmrPtr->Generation + 1;
    D2Ptr->ReproStatus = _ME_ReproStatus_IH;
    D2Ptr->EconomicStatus = _ME_EconomicStatus_PostInd;
    D2Ptr->NoOfHiresMax = _ME_HiresMax_PostInd;
    D2Ptr->NoOfHires = 0;
    D2Ptr->DateLastSupplied[ _ME_MU_MuType_Wood ] = Township.Age.GetInSeconds() - 1;
```

```
D2Ptr->DateLastSupplied[ _ME_MU_MuType_Food ] = Township.Age.GetInSeconds() - 1;
D2Ptr->DateLastSupplied[ _ME_MU_MuType_Crafts ] = Township.Age.GetInSeconds() - 1;
D2Ptr->DateLastSupplied[ _ME_MU_MuType_Goods ] = Township.Age.GetInSeconds() - 1;
D2Ptr->HasStarved_Flag = -1;
D2Ptr->Age.SetInSeconds( 0 );
D2Ptr->BirthDate = Township.Age;
D2Ptr->SetCashOnHand( FrmrPtr->GetCashOnHand() - D1Ptr->GetCashOnHand() );

for( MuType = 0; MuType < _ME_MU_NoOfMuTypes; MuType++ )
{
    MbMuBillOfLading.ZeroAllValues();
    MbMuBillOfLading.SetMuType( MuType );
    MbMuBillOfLading.SetNoOfMu( FrmrPtr->RecycledMbMzOnHand.MuPool[ MuType ].GetNoOfMu() );
    FrmrPtr->RecycledMbMzOnHand.MuPool[ MuType ].RequisitionRecycledMbMu( &MbMuBillOfLading );
    D2Ptr->RecycledMbMzOnHand.MuPool[ MuType ].StoreRecycledMbMu( &MbMuBillOfLading );

    BillOfLading.ZeroAllValues();
    BillOfLading.SetMuType( MuType );
    BillOfLading.SetNoOfMu( FrmrPtr->InventoryOnHand.MuPool[ MuType ].GetNoOfMu() );
    FrmrPtr->InventoryOnHand.MuPool[ MuType ].RequisitionMateriel( &BillOfLading );
    D2Ptr->InventoryOnHand.MuPool[ MuType ].StoreMateriel( &BillOfLading );

    BillOfLading.ZeroAllValues();
    BillOfLading.SetMuType( MuType );
    BillOfLading.SetNoOfMu( FrmrPtr->SupplyMuOnHand.MuPool[ MuType ].GetNoOfMu() );
    FrmrPtr->SupplyMuOnHand.MuPool[ MuType ].RequisitionMateriel( &BillOfLading );
    D2Ptr->SupplyMuOnHand.MuPool[ MuType ].StoreMateriel( &BillOfLading );

    MbMuBillOfLading.ZeroAllValues();
    MbMuBillOfLading.SetMuType( MuType );
    MbMuBillOfLading.SetNoOfMu( FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu() );
    FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].RequisitionWaste( &MbMuBillOfLading );
    D2Ptr->WasteMbMzOnHand.MuPool[ MuType ].RecycleWaste( &MbMuBillOfLading );
}

MbEuBillOfLading.ZeroAllValues();
MbEuBillOfLading.SetMuType( _ME_MU_MuType_MbEu );
MbEuBillOfLading.SetNoOfMu( FrmrPtr->MbEu.GetNoOfMu() );
FrmrPtr->MbEu.RequisitionHours( &MbEuBillOfLading );
D2Ptr->MbEu.ZeroAllValues();
D2Ptr->MbEu.StoreHours( &MbEuBillOfLading );

D2Ptr->Genes = FrmrPtr->Genes;
D2Ptr->Genes.MutateGenes( &ERules, &Sounds );

IncludeInSums( D2Ptr );
```

```
      D2Ptr->EmploymentHistory.InitBEmploymentHistory();
      D2Ptr->TogglesPtr = FrmrPtr->TogglesPtr;
   }

   D1Ptr->UnionList.InitBContactList();
   D1Ptr->UnionList.PoolPtr = FrmrPtr->UnionList.PoolPtr;
   D1Ptr->CustomerList.InitBContactList();
   D1Ptr->CustomerList.PoolPtr = FrmrPtr->CustomerList.PoolPtr;
   D1Ptr->SupplierList.InitBContactList();
   D1Ptr->SupplierList.PoolPtr = FrmrPtr->SupplierList.PoolPtr;
   D2Ptr->UnionList.InitBContactList();
   D2Ptr->UnionList.PoolPtr = FrmrPtr->UnionList.PoolPtr;
   D2Ptr->CustomerList.InitBContactList();
   D2Ptr->CustomerList.PoolPtr = FrmrPtr->CustomerList.PoolPtr;
   D2Ptr->SupplierList.InitBContactList();
   D2Ptr->SupplierList.PoolPtr = FrmrPtr->SupplierList.PoolPtr;

   // At this point, FrmrPtr:
   //    - has been withdrawn from Employer, Supplier and Customer lists,
   //    - has been disconnected from the Lot,
   //    - has had assets drained, and
   //    - has had genes copied and mutated.
   // The daughters have all they need.
   // We can delete the old body of the Frmr from the active list.

   FrmrList.DelFrmr( FrmrPtr->SlotNo );

   AddThisFrmrToEmployerLists( D1Ptr );
   AddThisFrmrToSupplierLists( D1Ptr );
   AddThisFrmrToCustomerLists( D1Ptr );

   AddThisFrmrToEmployerLists( D2Ptr );
   AddThisFrmrToSupplierLists( D2Ptr );
   AddThisFrmrToCustomerLists( D2Ptr );
}
```

bool BModEcoSys::ReproduceWrkr(BWrkr* WrkrPtr)

```
{
   ThisLotPtr = (BTwpLot*) WrkrPtr->LotPtrPair.ObjectVdPtr;
   // If there is no empty lot available within the commuting
   //   area, the second daughter cannot find a location
   //   and must die.  Find a vacant lot near this lot.
   D2HasALot = FindVacantResidence( ThisLotPtr, &ResidencePtrPair );
   if( D2HasALot == FALSE ) return FALSE;
```

```
// We assume that the Wrkr is ready to reproduce.
D1Ptr = WrkrList.AddWrkr( IncNextAgentSerNo() );
if( D1Ptr == NULL )
{
    return FALSE;
}
D2Ptr = WrkrList.AddWrkr( IncNextAgentSerNo() );
if( D2Ptr == NULL )
{
    // Release the D1 slot in the WrkrList.
    WrkrList.DelWrkr( D1Ptr->SlotNo );
    return FALSE;
}

ThisLotPtr = (BTwpLot*) WrkrPtr->LotPtrPair.ObjectVdPtr;

// AddWrkr assigns a few attrributes as follows:
// - SlotNo
// - WrkrSerNo
// - Dynasty
// - PrevPtr, PrevSlotNo, NextPtr, NextSlotNo

// THE REPRODUCTION IS DEFINITELY ON, START THE PROCESS.

// Take a statistical reading.
WrkrPtr->D1SerNo = D1Ptr->WrkrSerNo;
WrkrPtr->D2SerNo = D2Ptr->WrkrSerNo;

// Collect death-related statistics.

// The parent Wrkr must withdraw from society before it
//   splits to form two daughters; D1 and D2.
RemoveThisWrkrFromCustomerLists( WrkrPtr );
RemoveThisWrkrFromUnionLists( WrkrPtr );

ExcludeFromSums( WrkrPtr );

// LotPtrPair is a potential problem for a new Wrkr.
D1Ptr->LotPtrPair = WrkrPtr->LotPtrPair;  // D1 gets parent's location.
// Now point the lot to the Wrkr.
D1LotPtr = (BTwpLot*) D1Ptr->LotPtrPair.ObjectVdPtr;
D1Ptr->ResUnitNo = WrkrPtr->ResUnitNo;       // D1 gets parent's apartment.
// Free up the parent's slot in Resident List.
D1LotPtr->ResidentList.DelResident( D1Ptr->ResUnitNo );
// We know at least one unit is free.
```

```
    // Color the resident.
    D1Ptr->TransmitRsPtr( RsPtr );
    D1Ptr->ComputeWrkrsOwnColor();
    D1Ptr->DrawingColor = D1Ptr->WrkrsOwnColor;
    // Take an apartment at this address.
    AddResident( D1LotPtr, D1Ptr );

    // Find a residence for D2.
    // If there is no empty rental unit available within the commuting
    //   area, and no available vacant lot, the second daughter cannot
    //   find a location and must die.

    // Look for a place to settle in the commuting area.
    {
        // Move D2 into the rental unit.
        D2LotPtr = (BTwpLot*) ResidencePtrPair.ObjectVdPtr;
        // Color the resident.
        D2Ptr->TransmitRsPtr( RsPtr );
        D2Ptr->ComputeWrkrsOwnColor();
        D2Ptr->DrawingColor = D2Ptr->WrkrsOwnColor;
        // Take an apartment at this address.
        AddResident( D2LotPtr, D2Ptr );
    }

    // Go through the remaining attributes in order and fill them in.
    D1Ptr->MaSerNo = WrkrPtr->WrkrSerNo;
    WrkrPtr->D1SerNo = D1Ptr->WrkrSerNo;
    D1Ptr->D1SerNo = -1;
    D1Ptr->D2SerNo = -1;
    D1Ptr->Generation = WrkrPtr->Generation + 1;
    D1Ptr->ReproStatus = _ME_ReproStatus_IH;
    D1Ptr->DateLastHired = Township.Age.GetInSeconds() - 1;
    D1Ptr->DateLastSupplied[ _ME_MU_MuType_Wood ] = Township.Age.GetInSeconds() - 1;
    D1Ptr->DateLastSupplied[ _ME_MU_MuType_Food ] = Township.Age.GetInSeconds() - 1;
    D1Ptr->DateLastSupplied[ _ME_MU_MuType_Crafts ] = Township.Age.GetInSeconds() - 1;
    D1Ptr->DateLastSupplied[ _ME_MU_MuType_Goods ] = Township.Age.GetInSeconds() - 1;
    D1Ptr->HasStarved_Flag = -1;
    D1Ptr->Age.SetInSeconds( 0 );
    D1Ptr->BirthDate = Township.Age;
    D1Ptr->SetCashOnHand( WrkrPtr->GetCashOnHand() / 2.0 );

    for( MuType = 0; MuType < _ME_MU_NoOfMuTypes; MuType++ )
    {
        BillOfLading.ZeroAllValues();
        BillOfLading.SetMuType( MuType );
        BillOfLading.SetNoOfMu( WrkrPtr->SupplyMuOnHand.MuPool[ MuType ].GetNoOfMu() / 2.0 );
```

```
        WrkrPtr->SupplyMuOnHand.MuPool[ MuType ].RequisitionMateriel( &BillOfLading );
        D1Ptr->SupplyMuOnHand.MuPool[ MuType ].StoreMateriel( &BillOfLading );

        MbMuBillOfLading.ZeroAllValues();
        MbMuBillOfLading.SetMuType( MuType );
        MbMuBillOfLading.SetNoOfMu( WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu() / 2.0 );
        WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].RequisitionWaste( &MbMuBillOfLading );
        D1Ptr->WasteMbMzOnHand.MuPool[ MuType ].RecycleWaste( &MbMuBillOfLading );
    }

    MbEuBillOfLading.ZeroAllValues();
    MbEuBillOfLading.SetMuType( _ME_MU_MuType_MbEu );
    MbEuBillOfLading.SetNoOfMu( WrkrPtr->MbEu.GetNoOfMu() / 2.0 );
    WrkrPtr->MbEu.RequisitionHours( &MbEuBillOfLading );
    D1Ptr->MbEu.ZeroAllValues();
    D1Ptr->MbEu.StoreHours( &MbEuBillOfLading );

    D1Ptr->Genes = WrkrPtr->Genes;
    D1Ptr->Genes.MutateGenes( &ERules, &Sounds );

    IncludeInSums( D1Ptr );

    D1Ptr->EmployerList.InitBContactList();
    D1Ptr->EmployerList.PoolPtr = WrkrPtr->EmployerList.PoolPtr;
    D1Ptr->SupplierList.InitBContactList();
    D1Ptr->SupplierList.PoolPtr = WrkrPtr->SupplierList.PoolPtr;
    D1Ptr->EmploymentHistory.InitBEmploymentHistory();
    D1Ptr->TogglesPtr = WrkrPtr->TogglesPtr;
    Township.DrawOneLot( D1LotPtr->LotSlotNo );

    if( D2HasALot )
    {
        D2Ptr->MaSerNo = WrkrPtr->WrkrSerNo;
        WrkrPtr->D2SerNo = D2Ptr->WrkrSerNo;
        D2Ptr->D1SerNo = -1;
        D2Ptr->D2SerNo = -1;
        D2Ptr->Generation = WrkrPtr->Generation + 1;
        D2Ptr->ReproStatus = _ME_ReproStatus_IH;
        D2Ptr->DateLastHired = Township.Age.GetInSeconds() - 1;
        D2Ptr->DateLastSupplied[ _ME_MU_MuType_Wood ] = Township.Age.GetInSeconds() - 1;
        D2Ptr->DateLastSupplied[ _ME_MU_MuType_Food ] = Township.Age.GetInSeconds() - 1;
        D2Ptr->DateLastSupplied[ _ME_MU_MuType_Crafts ] = Township.Age.GetInSeconds() - 1;
        D2Ptr->DateLastSupplied[ _ME_MU_MuType_Goods ] = Township.Age.GetInSeconds() - 1;
        D2Ptr->HasStarved_Flag = -1;
        D2Ptr->Age.SetInSeconds( 0 );
        D2Ptr->BirthDate = Township.Age;
```

```
    D2Ptr->SetCashOnHand( WrkrPtr->GetCashOnHand() - D1Ptr->GetCashOnHand() );

    for( MuType = 0; MuType < _ME_MU_NoOfMuTypes; MuType++ )
    {
        BillOfLading.ZeroAllValues();
        BillOfLading.SetMuType( MuType );
        BillOfLading.SetNoOfMu( WrkrPtr->SupplyMuOnHand.MuPool[ MuType ].GetNoOfMu() );
        WrkrPtr->SupplyMuOnHand.MuPool[ MuType ].RequisitionMateriel( &BillOfLading );
        D2Ptr->SupplyMuOnHand.MuPool[ MuType ].StoreMateriel( &BillOfLading );

        MbMuBillOfLading.ZeroAllValues();
        MbMuBillOfLading.SetMuType( MuType );
        MbMuBillOfLading.SetNoOfMu( WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu() );
        WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].RequisitionRecycledMbMu( &MbMuBillOfLading );
        D2Ptr->WasteMbMzOnHand.MuPool[ MuType ].StoreRecycledMbMu( &MbMuBillOfLading );
    }

    MbEuBillOfLading.ZeroAllValues();
    MbEuBillOfLading.SetMuType( _ME_MU_MuType_MbEu );
    MbEuBillOfLading.SetNoOfMu( WrkrPtr->MbEu.GetNoOfMu() );
    WrkrPtr->MbEu.RequisitionHours( &MbEuBillOfLading );
    D2Ptr->MbEu.ZeroAllValues();
    D2Ptr->MbEu.StoreHours( &MbEuBillOfLading );

    D2Ptr->Genes = WrkrPtr->Genes; // TODO: Mutate these.
    D2Ptr->Genes.MutateGenes( &ERules, &Sounds );

    IncludeInSums( D2Ptr );

    D2Ptr->EmployerList.InitBContactList();
    D2Ptr->EmployerList.PoolPtr = WrkrPtr->EmployerList.PoolPtr;
    D2Ptr->SupplierList.InitBContactList();
    D2Ptr->SupplierList.PoolPtr = WrkrPtr->SupplierList.PoolPtr;
    D2Ptr->EmploymentHistory.InitBEmploymentHistory();
    D2Ptr->TogglesPtr = WrkrPtr->TogglesPtr;
}

// At this point the mother (WrkrPtr):
//    - has been withdrawn from contact lists,
//    - has been disconnected from the Lot,
//    - has had its assets drained, and
//    - has had its genes copied and mutated.
// The daughters have all they need.
// We can delete the body of the Wrkr from the active list.

WrkrList.DelWrkr( WrkrPtr->SlotNo );
```

```
    AddThisWrkrToUnionLists( D1Ptr );
    AddThisWrkrToCustomerLists( D1Ptr );
    AddThisWrkrToUnionLists( D2Ptr );
    AddThisWrkrToCustomerLists( D2Ptr );
}
```

bool BModEcoSys::FindVacantLot(BTwpLot* TwpLotPtr, BPtrPair* LaPtr)

```
{
    CaPtr = &TwpLotPtr->CommuteArea;
    CaWidth = CaPtr->CaWidth;
    for( CaRow = 0; CaRow < CaWidth; CaRow++ )
    {
        for( CaCol = 0; CaCol < CaWidth; CaCol++ )
        {
            TrgtTwpLotPtr = (BTwpLot*) CaPtr->GetTwpLotPtr( CaCol, CaRow );
            if( TrgtTwpLotPtr->GetLotDevType() == _ME_LotDevType_UnSpec )
                VacantLotList.AddLotSlotNo( TrgtTwpLotPtr->LotSlotNo );
        }
    }

    if( VacantLotList.GetNoOfUnits() <= 0 )
    {
        // No vacant lots exist within the commuting area.
        LaPtr->InitBPtrPair();
        return FALSE;
    }

    NoOfVacantLots = VacantLotList.GetNoOfUnits();
    if( NoOfVacantLots > 0 )
    {
        // Randomly select one of the vacant rental lots in the list.
        RandomNumber = RsPtr->randDblExc();       // Real number (0,1)
        VacantLotSlotNo = (long) floor( RandomNumber * (double) NoOfVacantLots );
        // Figure out what lot this unit is on.
        LotSlotNo = VacantLotList.Units[ VacantLotSlotNo ];
        TrgtTwpLotPtr = Township.GetLotPtr( LotSlotNo );
        LaPtr->PtrPairActiveFlag = _ME_PtrPairActiveFlag_Yes;
        LaPtr->AddrSlotNo = -1;     // Only used with BResidentList.
        LaPtr->ObjectSlotNo = TrgtTwpLotPtr->LotSlotNo;
        LaPtr->ObjectVdPtr = (void*) TrgtTwpLotPtr;
    }
}
```

// Subsidiary to DoAgentsDeath.

long BModEcoSys::DoAgentsDeath(long CurrentFunction)

```
{
	for( FrmrNo = 0; FrmrNo < NoOfFrmrs; FrmrNo++ )
	{
		FrmrPtr = FrmrList.GetFrmrPtr( FrmrSlotNoList.GetRandomSlotNo() );

		// A Frmr will die if it is:
		//   - Old enough  (Age >= than its ADT).
		//   - Financially unsound enough (NetWorth < its CDT) or HasStarved.
		Age = FrmrPtr->Age.GetInSeconds();
		NetWorth = FrmrPtr->GetNetWorth();
		if( ( Age > FrmrPtr->Genes.ADT ) ||
			( NetWorth < FrmrPtr->Genes.CDT ) ||
			( FrmrPtr->HasStarved_Flag == _ME_HasStarved_Flag_Yes ) )
		{
			ExcludeFromSums( FrmrPtr );
			KillFrmr( FrmrPtr );
		}
	}

	for( WrkrNo = 0; WrkrNo < NoOfWrkrs; WrkrNo++ )
	{
		WrkrPtr = WrkrList.GetWrkrPtr( WrkrSlotNoList.GetRandomSlotNo() );

		// A Wrkr will die if it is:
		//   - Old enough  (Age >= than its ADT).
		//   - Financially unsound enough (NetWorth < its CDT) or HasStarved.
		Age = WrkrPtr->Age.GetInSeconds();
		NetWorth = WrkrPtr->GetNetWorth();
		if( ( Age > WrkrPtr->Genes.ADT ) ||
			( NetWorth < WrkrPtr->Genes.CDT ) ||
			( WrkrPtr->HasStarved_Flag == _ME_HasStarved_Flag_Yes ) )
		{
			ExcludeFromSums( WrkrPtr );
			KillWrkr( WrkrPtr );
		}
	}
}
```

bool BModEcoSys::KillFrmr(BFrmr* FrmrPtr)

```
{
    if( FrmrPtr->GetNetWorth() < FrmrPtr->Genes.CDT ) newDeathMode = _ME_Sp08Ad_DeathMode_CDT;
```

```
    if( FrmrPtr->Age.GetInSeconds() > FrmrPtr->Genes.ADT ) newDeathMode = _ME_Sp08Ad_DeathMode_ADT;
    if( FrmrPtr->HasStarved_Flag == _ME_HasStarved_Flag_Yes ) newDeathMode = _ME_Sp08Ad_DeathMode_MPT;

    TodaysDate = Township.Age.GetInSeconds();

    // Clear all contact information with Wrkrs.
    RemoveThisFrmrFromEmployerLists( FrmrPtr );
    ClearUnionList( FrmrPtr );
    RemoveThisFrmrFromSupplierLists( FrmrPtr );
    ClearCustomerList( FrmrPtr );
    ClearSupplierList( FrmrPtr );
    // Send inventory to the township.
    ClearFrmrInventory( FrmrPtr );
    // Vacate the lot.
    TwpLotPtr = (BTwpLot*) FrmrPtr->LotPtrPair.ObjectVdPtr;
    TwpLotPtr->FrmrPtrPair.InitBPtrPair();
    TwpLotPtr->SetLotDevType( _ME_LotDevType_UnSpec );
    Township.DrawOneLot( TwpLotPtr->LotSlotNo );
    // Release the slot in the FrmrList.
    FrmrList.DelFrmr( FrmrPtr->SlotNo );
}
```

bool BModEcoSys::ClearUnionList(BFrmr* FrmrPtr)

```
{
    NoOfContacts = FrmrPtr->UnionList.GetNoOfContacts();
    if( NoOfContacts <= 0 ) return TRUE;

    for( ContactNo = 0; ContactNo < NoOfContacts; ContactNo++ )
    {
        ContactPtr = FrmrPtr->UnionList.GetTopContactPtr();
        WrkrPtr = (BWrkr*) ContactPtr->AgentVdPtr;
        MtchPtr = ContactPtr->MtchPtr;
        FrmrPtr->UnionList.DelContact( ContactPtr );
        WrkrPtr->EmployerList.DelContact( MtchPtr );
    }
}
```

bool BModEcoSys::ClearCustomerList(BFrmr* ThisFrmrPtr)

```
{
    NoOfContacts = ThisFrmrPtr->CustomerList.GetNoOfContacts();
    if( NoOfContacts <= 0 ) return TRUE;

    ContactPtr = ThisFrmrPtr->CustomerList.GetTopContactPtr();
    for( ContactNo = 0; ContactNo < NoOfContacts; ContactNo++ )
    {
```

```
        if( ContactPtr->AgentType == _ME_CI_AgentType_Frmr )
        {
            ThatFrmrPtr = (BFrmr*) ContactPtr->AgentVdPtr;
            MtchPtr = ContactPtr->MtchPtr;
            ThisFrmrPtr->CustomerList.DelContact( ContactPtr );
            ThatFrmrPtr->SupplierList.DelContact( MtchPtr );
        }
        else
        {
            WrkrPtr = (BWrkr*) ContactPtr->AgentVdPtr;
            MtchPtr = ContactPtr->MtchPtr;
            ThisFrmrPtr->CustomerList.DelContact( ContactPtr );
            WrkrPtr->SupplierList.DelContact( MtchPtr );
        }
    }
}
```

bool BModEcoSys::ClearSupplierList(BFrmr* ThisFrmrPtr)

```
{
    NoOfContacts = ThisFrmrPtr->SupplierList.GetNoOfContacts();
    if( NoOfContacts <= 0 ) return TRUE;

    for( ContactNo = 0; ContactNo < NoOfContacts; ContactNo++ )
    {
        ContactPtr = ThisFrmrPtr->SupplierList.GetTopContactPtr();
        ThatFrmrPtr = (BFrmr*) ContactPtr->AgentVdPtr;
        MtchPtr = ContactPtr->MtchPtr;
        ThisFrmrPtr->SupplierList.DelContact( ContactPtr );
        ThatFrmrPtr->CustomerList.DelContact( MtchPtr );
    }
}
```

bool BModEcoSys::ClearFrmrInventory(BFrmr* FrmrPtr)

```
{
    // Send cash to the township.
    Township.SetCash_Grants( Township.GetCash_Grants() + FrmrPtr->GetCashOnHand() );
    FrmrPtr->SetCashOnHand( 0.0 );

    NoOfMuTypes = _ME_MU_NoOfMuTypes;
    for( MuType = 0; MuType < NoOfMuTypes; MuType++ )
    {
        MbMuBillOfLading.SetMuType( MuType );
        MbMuBillOfLading.SetNoOfMu( FrmrPtr->RecycledMbMzOnHand.MuPool[ MuType ].GetNoOfMu() );
        FrmrPtr->RecycledMbMzOnHand.MuPool[ MuType ].RequisitionRecycledMbMu( &MbMuBillOfLading );
        Township.ProcessEstateProperties( &MbMuBillOfLading );
```

```
        BillOfLading.SetMuType( MuType );
        BillOfLading.SetNoOfMu( FrmrPtr->InventoryOnHand.MuPool[ MuType ].GetNoOfMu() );
        FrmrPtr->InventoryOnHand.MuPool[ MuType ].RequisitionMateriel( &BillOfLading );
        Township.ProcessEstateProperties( &BillOfLading );

        BillOfLading.SetMuType( MuType );
        BillOfLading.SetNoOfMu( FrmrPtr->SupplyMuOnHand.MuPool[ MuType ].GetNoOfMu() );
        FrmrPtr->SupplyMuOnHand.MuPool[ MuType ].RequisitionMateriel( &BillOfLading );
        Township.ProcessEstateProperties( &BillOfLading );

        MbMuBillOfLading.SetMuType( MuType );
        MbMuBillOfLading.SetNoOfMu( FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu() );
        FrmrPtr->WasteMbMzOnHand.MuPool[ MuType ].RequisitionWaste( &MbMuBillOfLading );
        Township.ProcessEstateProperties( &MbMuBillOfLading );
    }

    MbEuBillOfLading.SetMuType( _ME_MU_MuType_MbEu );
    MbEuBillOfLading.SetNoOfMu( FrmrPtr->MbEu.GetNoOfMu() );
    FrmrPtr->MbEu.RequisitionHours( &MbEuBillOfLading );
    Township.MbEuG_MuPool.StoreHours( &MbEuBillOfLading );
}
```

bool BModEcoSys::KillWrkr(BWrkr* WrkrPtr)

```
{
    if( WrkrPtr->GetNetWorth() < WrkrPtr->Genes.CDT ) newDeathMode = _ME_Sp08Ad_DeathMode_CDT;
    if( WrkrPtr->Age.GetInSeconds() > WrkrPtr->Genes.ADT ) newDeathMode = _ME_Sp08Ad_DeathMode_ADT;
    if( WrkrPtr->HasStarved_Flag == _ME_HasStarved_Flag_Yes ) newDeathMode = _ME_Sp08Ad_DeathMode_MPT;

    // Take a statistical reading.
    TodaysDate = Township.Age.GetInSeconds();

    // Clear all contact information with Frmrs.
    RemoveThisWrkrFromUnionLists( WrkrPtr );
    ClearEmployerList( WrkrPtr );
    RemoveThisWrkrFromCustomerLists( WrkrPtr );
    ClearSupplierList( WrkrPtr );
    // Send inventory to the township.
    ClearWrkrInventory( WrkrPtr );
    // Vacate the lot.
    TwpLotPtr = (BTwpLot*) WrkrPtr->LotPtrPair.ObjectVdPtr;
    TwpLotPtr->ResidentList.DelResident( WrkrPtr->ResUnitNo );
    if( TwpLotPtr->ResidentList.GetNoOfResidents() == 0 )
    {
        TwpLotPtr->SetLotDevType( _ME_LotDevType_UnSpec );
```

```
    }
    // Release the slot in the WrkrList.
    WrkrList.DelWrkr( WrkrPtr->SlotNo );
}
```

bool BModEcoSys::ClearEmployerList(BWrkr* WrkrPtr)

```
{
    NoOfContacts = WrkrPtr->EmployerList.GetNoOfContacts();
    if( NoOfContacts <= 0 ) return TRUE;

    for( ContactNo = 0; ContactNo < NoOfContacts; ContactNo++ )
    {
        ContactPtr = WrkrPtr->EmployerList.GetTopContactPtr();
        FrmrPtr = (BFrmr*) ContactPtr->AgentVdPtr;
        MtchPtr = ContactPtr->MtchPtr;
        WrkrPtr->EmployerList.DelContact( ContactPtr );
        FrmrPtr->UnionList.DelContact( MtchPtr );
    }
}
```

bool BModEcoSys::ClearSupplierList(BWrkr* WrkrPtr)

```
{
    NoOfContacts = WrkrPtr->SupplierList.GetNoOfContacts();
    for( ContactNo = 0; ContactNo < NoOfContacts; ContactNo++ )
    {
        ContactPtr = WrkrPtr->SupplierList.GetTopContactPtr();
        FrmrPtr = (BFrmr*) ContactPtr->AgentVdPtr;
        MtchPtr = ContactPtr->MtchPtr;
        WrkrPtr->SupplierList.DelContact( ContactPtr );
        FrmrPtr->CustomerList.DelContact( MtchPtr );
    }
}
```

bool BModEcoSys::ClearWrkrInventory(BWrkr* WrkrPtr)

```
{
    // Send cash to the township.
    Township.SetCash_Grants( Township.GetCash_Grants() + WrkrPtr->GetCashOnHand() );
    WrkrPtr->SetCashOnHand( 0.0 );

    NoOfMuTypes = _ME_MU_NoOfMuTypes;
    for( MuType = 0; MuType < NoOfMuTypes; MuType++ )
    {
        BillOfLading.SetMuType( MuType );
        BillOfLading.SetNoOfMu( WrkrPtr->SupplyMuOnHand.MuPool[ MuType ].GetNoOfMu() );
```

```
        WrkrPtr->SupplyMuOnHand.MuPool[ MuType ].RequisitionMateriel( &BillOfLading );
        Township.ProcessEstateProperties( &BillOfLading );

        MbMuBillOfLading.SetMuType( MuType );
        MbMuBillOfLading.SetNoOfMu( WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].GetNoOfMu() );
        WrkrPtr->WasteMbMzOnHand.MuPool[ MuType ].RequisitionWaste( &MbMuBillOfLading );
        Township.ProcessEstateProperties( &MbMuBillOfLading );
    }

    MbEuBillOfLading.SetMuType( _ME_MU_MuType_MbEu );
    MbEuBillOfLading.SetNoOfMu( WrkrPtr->MbEu.GetNoOfMu() );
    WrkrPtr->MbEu.RequisitionHours( &MbEuBillOfLading );
    Township.MbEuG_MuPool.StoreHours( &MbEuBillOfLading );
}
```

// Subsidiary to DoCleanup.

long BModEcoSys::DoCleanup(long CurrentFunction)

```
{
        AgeAllAgents();
}
```

bool BModEcoSys::AgeAllAgents(void)

```
{
    FrmrPtr = FrmrList.GetTopFrmrPtr();
    NoOfFrmrs = FrmrList.GetNoOfFrmrs();
    for( FrmrNo = 0; FrmrNo < NoOfFrmrs; FrmrNo++ )
    {
        FrmrPtr->Age.AgeByOneSecond();
        // Advance to next record.
        FrmrPtr = FrmrPtr->NextPtr;
    }

    WrkrPtr = WrkrList.GetTopWrkrPtr();
    NoOfWrkrs = WrkrList.GetNoOfWrkrs();
    for( WrkrNo = 0; WrkrNo < NoOfWrkrs; WrkrNo++ )
    {
        WrkrPtr->Age.AgeByOneSecond();
        // Advance to next record.
        WrkrPtr = WrkrPtr->NextPtr;
    }
}
```