# NWN2 Toolset Guide

## Volume IV – Sample Scripts

### Edition 6.1

If you find any corrections or have suggestions, edits and additions that would improve this document, I would enjoy hearing from you. Please send me a friendly e-mail. Thank you for taking the time to look through this work, and I hope you find it of some use.

—Bob Hall

September 2, 2013

## Table of Contents


Breaking Bottles in a Container....2
Creating Statues from Creatures....3
Evil Cloud Trigger....4
Custom Walk Waypoint Set....7
Friendly Storekeeper....11
Guard Spawn....13
Include Files Index....15
Intelligent Weapon One-Liner Trigger....17
Invisible Enclosure....19
Lightning Bolt Strike....23
Lock Area Doors at Night....25
Marking Removed Items as Stolen....28
Open Sarcophagus....29
Orient Camera....31
Placeable Damage Bug Fix....32
Restocking a Store....33
Returning Weapon....40
Set Name from Variables....46
Sound Activattion Trigger....47
Spawn a Party Member....49
Swag Bag Treasure....50
Tiling an Effect....51
Toggle Streetlights....54
Turn to Face....57
Undergrowth Trigger....60
Using Best Appraise Skill....62

## Breaking Bottles in a Container

```
// breaking_bottles
/*
   This script can be placed in the 'On Container Damaged' property field
   for a container. The script computes the net damage to the container
   after deducting the hardness, then checks whether any items of base
   type potion suffer damage. The odds for a potion to break is 3% per
   point of damage that the container takes (beyond the hardness rating
   of the container). Note that this includes any damage that takes the
   container below zero hit points. A potion that breaks is replaced by
   the broken item blueprint.
*/
// RJH 11sep10

const string BROKEN_ITEM_TAG = "n2_it_brokenitem";

void main()
{
  // Compute the inflicted damage
  int nHardness = GetHardness(OBJECT_SELF);
  int nDamage = GetTotalDamageDealt();
  nDamage = ( nDamage > nHardness ) ? (nDamage - nHardness) : 0;

  // Check the inventory for potions
  object oItem = GetFirstItemInInventory();
  while ( GetIsObjectValid( oItem ) ) {
    int nType = GetBaseItemType( oItem );
    if ( nType == BASE_ITEM_POTIONS ) {
      // Check for random damage
      if ( d100() < ( 3 * nDamage ) ) {
        // Replace potion with broken item
        DestroyObject( oItem, 1.0f, FALSE );
        CreateItemOnObject( BROKEN_ITEM_TAG );
      }
    }
    oItem = GetNextItemInInventory();
  }
}
```

## Creating Statues from Creatures

```
// statue_from_creature
/*
   This script should be placed in the 'On Spawn In' script of a creature. When
   the creature is spawned into the game, animation is permanently turned off,
   a stoneskin visual effect is applied, and a death effect is used to make the
   creature non-living. Finally, a collision ball is placed in the location of
   the creature to prevent player movement through the object.
*/
// RJH 11sep10

const string COLLISION_BALL_TAG = "plc_collbox";

void main()
{
   // Stop the creature animating
   effect eFreeze = EffectVisualEffect( VFX_DUR_FREEZE_ANIMATION );
   ApplyEffectToObject( DURATION_TYPE_PERMANENT,
      SupernaturalEffect( eFreeze ), OBJECT_SELF );

   // Apply a stone skin visual effect
   effect eStone = EffectVisualEffect( VFX_DUR_PROT_STONESKIN );
   ApplyEffectToObject( DURATION_TYPE_PERMANENT, eStone, OBJECT_SELF );

   // Avoid fadeout on death and make it generally inert
   SetIsDestroyable( FALSE, FALSE, FALSE );
   SetBumpState( OBJECT_SELF, BUMPSTATE_UNBUMPABLE );
   SetCommandable( FALSE, OBJECT_SELF );
   SetOrientOnDialog( OBJECT_SELF, FALSE );

   // Apply the death effect
   ApplyEffectToObject( DURATION_TYPE_PERMANENT, EffectDeath(), OBJECT_SELF );

   // Flag the creature as a statue (for the Flesh to Stone spell)
   SetLocalInt( OBJECT_SELF, "NW_STATUE", 1 );

   // Create collision ball around the statue
   location loc = GetLocation( OBJECT_SELF );
   object oCollBall = CreateObject( OBJECT_TYPE_PLACEABLE, COLLISION_BALL_TAG, loc );

   // Scale the box to fit the statue size
   float fSS = 0.5f;
   SetScale( oCollBall, fSS, fSS, fSS );
}
```

## Evil Cloud Trigger

```
// evil_cloud_enter
/*
   This is the 'On Enter' script for a Trigger blueprint that simulates
   movement into an 'evil cloud'. When a creature enters, it receives a
   permanent, supernatural concealment effect so that it does not time
   out or go away while resting. This effect is given a unique spell ID
   that will be used to remove the effect upon exit.

   An Ipoint is created to run a script "evil_cloud_run" that will apply
   the other cloud damage effects to the target creature. The benefit of
   applying the damage from an Ipoint, rather than from the heartbeat
   script of the Trigger area, is that the feedback messages to the player
   will refer to the name of the Ipoint ("Evil cloud"), rather than a
   generic "Something". I.e. 'Something damages the PC'.

   To create the evil cloud, first create a trigger blueprint that has
   this script in the 'On Enter' script property and 'evil_cloud_exit'
   in the 'On Exit' script property. The visual effect for the evil
   cloud can be generated by the 'Black Cloud' placed effect blueprint.
   In an area, create a roughly circular trigger that is smaller than
   the maximum diameter of the placed effect, then move the placed
   effect to the center of this trigger.
*/
// RJH 13jul11

const int CONCEALMENT = 20; // 20% concealment
const string EC_TARGET = "ec_target"; // For run script
const string EC_SOURCE = "ec_source"; // For exit script
const int EC_SPELL_ID = 66600; // Unique ID
const string EC_RUN_SCRIPT = "evil_cloud_run"; // Name of Ipoint script
const string EC_NAME = "<c=#8e8e8e>Evil cloud</c>"; // Name to show player
const string IPOINT = "plc_ipoint "; // Note the extra space at the end

void main()
{
    object oTarget = GetEnteringObject();

   // Apply a concealment effect
   effect eConceal = EffectConcealment( CONCEALMENT );
   eConceal = SupernaturalEffect( eConceal );
   eConceal = SetEffectSpellId( eConceal, EC_SPELL_ID );
   ApplyEffectToObject( DURATION_TYPE_PERMANENT, eConceal, oTarget );
```

```
   // Create an Ipoint to run the evil cloud effects
   location locTarget = GetLocation( oTarget );
   object oIpoint = CreateObject( OBJECT_TYPE_PLACEABLE,
      IPOINT, locTarget );
   if ( GetIsObjectValid( oIpoint ) ) {
      // Set the Ipoint name and make unusable
      SetFirstName( oIpoint, EC_NAME );
      SetUseableFlag( oIpoint, FALSE );

      // Store each object as a local variable on the other
      SetLocalObject( oTarget, EC_SOURCE, oIpoint );
      SetLocalObject( oIpoint, EC_TARGET, oTarget );

      // Run an effects script on the Ipoint.
      DelayCommand( 1.0f,
         ExecuteScript( EC_RUN_SCRIPT, oIpoint ) );
   }
}
```

```
//  evil_cloud_exit
/*
   This is the On Exit script for a Trigger that simulates exiting an 'evil
   cloud'. It removes the concealment effect applied by the Trigger's 'On
   Enter' script and destroys the Ipoint being used to apply the cloud effects.
*/
// RJH 13jul11

const string EC_SOURCE = "ec_source";
const int EC_SPELL_ID = 66600; // Unique ID

void main()
{
    object oTarget = GetExitingObject();

   // Cycle through the target effects
   effect eEffect = GetFirstEffect( oTarget );
   while ( GetIsEffectValid( eEffect ) ) {
      // Compare the spell ID to EC_SPELL_ID
      int nEffectSpellId = GetEffectSpellId( eEffect );
      if ( nEffectSpellId == EC_SPELL_ID ) {
         // Remove the concealment effect and exit the loop
         RemoveEffect( oTarget, eEffect );
         break;
      }
      eEffect = GetNextEffect( oTarget );
   }
```

```
   // Eliminate the Ipoint, which will also stop 'evil_cloud_run'
   object oIpoint = GetLocalObject( oTarget, EC_SOURCE );
   if ( GetIsObjectValid( oIpoint ) )
      DestroyObject( oIpoint, 0.0f, FALSE );

   // Remove the local variable
   DeleteLocalObject( oTarget, EC_SOURCE );
}
```

```
// evil_cloud_run
/*
   This is an Ipoint script that inflicts damage to a target from the 'evil cloud'.
*/
// RJH 13jul11

const string EC_TARGET = "ec_target";

/* This is a recursive routine that is run every round.
   It applies a negative energy damage effect.
*/
void DoCloudEffects( object oTarget )
{
   // Make a Will save attempt at DC 15
   int nDC = 15;
   if ( WillSave( oTarget, nDC, SAVING_THROW_TYPE_DEATH ) == 0 ) {
      // Apply negative energy damage
      effect eDeath = EffectDamage( d6(1), DAMAGE_TYPE_NEGATIVE );
      effect eVisual = EffectVisualEffect( VFX_IMP_NEGATIVE_ENERGY );
      effect eLink = EffectLinkEffects( eDeath, eVisual );
      ApplyEffectToObject( DURATION_TYPE_INSTANT, eLink, oTarget );
   }

   // Resume the command in six seconds
   DelayCommand( 6.0f, DoCloudEffects( oTarget ) );
}

void main()
{
   // Get the target
   object oTarget = GetLocalObject( OBJECT_SELF, EC_TARGET );
   if ( GetIsObjectValid( oTarget ) ) {
      // Launch the recursive cloud effects routine
      DelayCommand( 0.1f, DoCloudEffects( oTarget ) );
   }
}
```

## Custom Walk Waypoint Set

```
// inc_walkway
/*
   The following routines allow a custom set of waypoints to be configured
   as the walk waypoints for a creature. The InitCustomWWPController routine
   should be called first, followed by AddCustomWaypoint for each waypoint.
   Use ClearCustomWaypoints to clear the custom walk waypoints and reload the
   default waypoints.
*/
// RJH 11sep10

#include "x0_i0_walkway"

// Prototypes
object InitCustomWWPController( object oCreature=OBJECT_SELF, string sScriptSuffix = "" );
int AddCustomWaypoint( object oCreature, string sCustomTag );
int SetCustomWaypoints( object oCreature=OBJECT_SELF, string sScriptSuffix = "", string
sWPTag1 = "", string sWPTag2 = "", string sWPTag3 = "", string sWPTag4 = "", string sWPTag5
= "", string sWPTag6 = "" );
void ClearCustomWaypoints( object oCreature=OBJECT_SELF, int bIsWalk = TRUE );
void StartWalking( object oCreature=OBJECT_SELF );

// Initialize the custom WWP Controller.
object InitCustomWWPController( object oCreature=OBJECT_SELF, string sScriptSuffix = "" )
{
    SetWalkCondition( NW_WALK_FLAG_INITIALIZED, TRUE, oCreature );
   string sWWPTag = GetWPTag( oCreature );

   // Find the Waypoint Controller Object
   string sWWPControllerTag = WAYPOINT_CONTROLLER_PREFIX + sWWPTag;
   object oWWPC = GetWaypointByTag( sWWPControllerTag );

   // Create it if it doesn't exist
   if ( !GetIsObjectValid( oWWPC ) ) {
      oWWPC = CreateObject( OBJECT_TYPE_WAYPOINT,
         RR_WAYPOINT_CONTROLLER, GetLocation( oCreature ), FALSE,
         sWWPControllerTag );
   }

   // Store the Waypoint Controller object (and associated tag) on the creature
   SetLocalObject( oCreature, VAR_WWP_CONTROLLER, oWWPC );
   SetLocalString( oCreature, VAR_WP_TAG, sWWPTag );

   // Initialize the total
```

```
   string sPrefix = GetWPPrefix();
   SetLocalInt( oWWPC, sPrefix + WW_NUM, 0 );

   // Store the script suffix
   SetLocalString( oCreature, VAR_WP_TAG, sScriptSuffix );

   return ( oWWPC );
}

/* Add a walk waypoint for the creature oCreature to the waypoint
   controller with the tag sCustomTag. This routine can be
   called repeatedly to add successive custom walk waypoints.
*/
int AddCustomWaypoint( object oCreature, string sCustomTag )
{
   // Check for a valid tag
   if ( sCustomTag == "" )
      return FALSE;

   // Check for a valid WWP controller
   object oWPPC = GetWWPController( oCreature );
   if ( !GetIsObjectValid( oWPPC ) )
      return FALSE;

   // Check for a valid waypoint
   object oWaypoint = GetObjectByTag( sCustomTag );
   if ( !GetIsObjectValid( oWaypoint ) ) {
      return FALSE;
   } else if ( GetObjectType( oWaypoint ) != OBJECT_TYPE_WAYPOINT ) {
      return FALSE;
   }

   // Store the next waypoint
   int nCount = GetNumWaypoints( oCreature );
    string sPrefix = GetWPPrefix();
   SetLocalObject( oWPPC, sPrefix + IntToString( nCount + 1 ), oWaypoint );
   SetLocalInt( oWPPC, sPrefix + WW_NUM, nCount + 1 );

   return TRUE;
}

/* Set up to six custom waypoints by making repeated calls
   to the AddCustomWaypoint routine.
*/
```

## Custom Walk Waypoint Set

```
int SetCustomWaypoints( object oCreature=OBJECT_SELF, string sScriptSuffix = "", string
sWPTag1 = "", string sWPTag2 = "", string sWPTag3 = "", string sWPTag4 = "", string sWPTag5
= "", string sWPTag6 = "" )
{
   // Initialize the controller
   object oWWPC = InitCustomWWPController( oCreature, sScriptSuffix );
   if ( !GetIsObjectValid( oWWPC ) )
      return FALSE;

   // Set the custom waypoints
   if ( ( sWPTag1 != "" ) && !AddCustomWaypoint( oCreature, sWPTag1 ) )
      return FALSE;
   if ( ( sWPTag2 != "" ) && !AddCustomWaypoint( oCreature, sWPTag2 ) )
      return FALSE;
   if ( ( sWPTag3 != "" ) && !AddCustomWaypoint( oCreature, sWPTag3 ) )
      return FALSE;
   if ( ( sWPTag4 != "" ) && !AddCustomWaypoint( oCreature, sWPTag4 ) )
      return FALSE;
   if ( ( sWPTag5 != "" ) && !AddCustomWaypoint( oCreature, sWPTag5 ) )
      return FALSE;
   if ( ( sWPTag6 != "" ) && !AddCustomWaypoint( oCreature, sWPTag6 ) )
      return FALSE;
   return TRUE;
}

/* This clears the custom walk waypoints by causing
   the walk waypoints to be reloaded from scratch.
*/
void ClearCustomWaypoints( object oCreature=OBJECT_SELF, int bIsWalk = TRUE )
{
   // Set the first waypoint on the WPPC to null
   object oWPPC = GetWWPController( oCreature );
   int nWP = GetNumWaypoints( oCreature );
   string sPrefix = GetWPPrefix();
   int i;
   for ( i = 1; i <= nWP; i++ ) {
      SetLocalObject( oWPPC, sPrefix + IntToString(i), OBJECT_INVALID );
   }

   // Clear the counters
   SetLocalInt( oWPPC, sPrefix + WW_NUM, -1 );
   SetLocalInt( oCreature, VAR_WP_NEXT, -1 );
   SetLocalInt( oCreature, VAR_WP_CURRENT, -1 );
   SetLocalInt( oCreature, VAR_WP_PREVIOUS, -1 );

   // Reload the walk waypoints
```

```
    LookUpWalkWayPoints( oWPPC );

    // Set the walk flag appropriately
    SetWalkCondition( NW_WALK_FLAG_CONSTANT, bIsWalk, oCreature );
}

// Start the creature walking the custom waypoints.
void StartWalking( object oCreature=OBJECT_SELF )
{
    // Use appropriate skills, only once
    if( GetSpawnInCondition( NW_FLAG_STEALTH ) )
        SetActionMode( OBJECT_SELF, ACTION_MODE_STEALTH, TRUE );

    // Add the new action mode stuff
    if( GetSpawnInCondition( NW_FLAG_SEARCH ) )
        SetActionMode( OBJECT_SELF, ACTION_MODE_DETECT, TRUE );

    // Make sure the walk flag is set and pause is turned off
    SetWalkCondition( NW_WALK_FLAG_CONSTANT, TRUE, oCreature );
    SetWalkCondition( NW_WALK_FLAG_PAUSED, FALSE, oCreature );

    // Set the next walk waypoint to the closest custom waypoint
    SetLocalInt( oCreature, VAR_WP_NEXT, GetNearestWalkWayPoint( oCreature ) );

    // Force resume
    SetLocalInt( oCreature, VAR_FORCE_RESUME, TRUE );
}
```

## Friendly Storekeeper

```
// wp_friendly_storekeeper
/*
   This is a walk waypoint script for a storekeeper. First, configure a
   two-waypoint walk path for the storekeeper and place the first where
   you want the creature to stay and face. Next, copy this script to the
   waypoint script for the shopkeeper. Once per day the storekeeper will
   wave a greeting when the PC comes within a given distance and is the
   nearest PC. If the shopkeeper is displaced away from the first waypoint,
   he will attempt to return.
*/
// RJH 11sep10

#include "ginc_wp"
#include "ginc_behavior" // for IsBusy() call

/* This routine will cause the owner to turn and face the nearest PC then
   wave a greeting. The PC must be within fMaxDistance of the owner, and
   must not have been greeted by the same creature during the current day.
   A TRUE is returned only if the greeting is made.
*/
int greet_PC_today( float fMaxDistance = 5.0f )
{
   string selfTag = GetTag( OBJECT_SELF );
   string greetTag = "Greet_" + selfTag;
   int greetState = GetLocalInt( oTarget, greetTag );
   int currentDay = GetCalendarDay();

   // Check if the nearest creature is a PC
   object oTarget = GetNearestObject( OBJECT_TYPE_CREATURE, OBJECT_SELF, 1 );
   if ( !GetIsObjectValid( oTarget ) || !GetIsPC( oTarget ) )
      return FALSE;

   // Return if the PC is beyond the distance threshold
   float fDistance = GetDistanceToObject( oTarget );
   if ( fDistance > fMaxDistance )
      return FALSE;

   // Check if we've met today
   if ( greetState != currentDay ) {
      // Store the state on the PC
      SetLocalInt( oTarget, greetTag, currentDay );

      // Halt any current animation
```

```
      ClearAllActions();

      // Face the PC
      SetFacingPoint( GetPosition( oTarget ) );

      // Wave a greeting
      int iAnimation = ANIMATION_FIREFORGET_GREETING;
      float fDurationSeconds = 2.0f;
      ActionPlayAnimation( iAnimation, 1.0f, fDurationSeconds );
      if ( AnimationNeedsWait( iAnimation ) )
          ActionWait( fDurationSeconds );

      // return TRUE to indicate a greeting was made
      return TRUE;
   }

   return FALSE; // Did not perform a greeting
}

void main()
{
   // Check if busy
   if ( IsBusy() )
      return;

   // Always stay at the first waypoint
   SetNextWaypoint( 1 );

   /* If the nearest creature is a PC, do a greeting animation.
      Otherwise, turn to face in the direction of the waypoint.
   */
   if ( ! greet_PC_today() )
      FaceAndPause( 1, 1.0f );
}
```

## Guard Spawn

```
// guard_spawn
/*
   Set the "SpawnScript" variable on a guard and it will execute this script
   when it is spawned into the game. This allows you to set user defined events
   to trigger as well as setting various behaviors. You can give the guard a
   'On User Defined Event' script to process special events.
*/
// 15aug13 RJH

#include "nw_i0_generic"

void main()
{
   // Make creature enter search mode after spawning
   SetSpawnInCondition( NW_FLAG_SEARCH );

   // Enable immobile ambient animations
   SetSpawnInCondition( NW_FLAG_IMMOBILE_AMBIENT_ANIMATIONS );

   // Fire User Defined Event 1001 -- on heartbeat
   SetSpawnInCondition( NW_FLAG_HEARTBEAT_EVENT );

   // Fire User Defined Event 1002 -- on perceive
   SetSpawnInCondition( NW_FLAG_PERCIEVE_EVENT );

   // Defensive attacker will use defensive combat feats and parry
   SetCombatCondition( X0_COMBAT_FLAG_DEFENSIVE );
}
```

```
// userenv_guard
/*
   Sample 'On User Defined Event' script for a guard.
*/
// 01sep13

// Turn observer toward the observed target
void TurnToFace( object oObserver, object oTarget, float fDelay = 0.0f )
{
   vector vTarget = GetPosition( oTarget );
   vector vObserver = GetPosition( oObserver );
   vector vFace = vTarget - vObserver;
   float fAngle = VectorToAngle( vFace );
```

```
   // Orient observer toward the target
   AssignCommand( oObserver, ClearAllActions() );
   AssignCommand( oObserver, DelayCommand( fDelay + 0.1f, SetFacing( fAngle ) ) );
   AssignCommand( oObserver, DelayCommand( fDelay + 0.5f, SetFacing( fAngle ) ) );
   AssignCommand( oObserver, DelayCommand( fDelay + 1.0f,
      ActionPlayAnimation( ANIMATION_FIREFORGET_SEARCH ) ) );
}

// This gives a nChangePct% chance of changing direction.
void ChangeFacing( int nChangePct, int nRandomFace, float fDelay = 0.0f )
{
   if ( Random( 100 ) < nChangePct ) {
      // Pick a random facing
      float fFacing = 1.0f * Random( 360 );
      DelayCommand( 0.01f + fDelay, SetFacing( fFacing ) );
   }
}

void main()
{
   // Process the event
   int nEvent = GetUserDefinedEventNumber();
   switch( nEvent ) {
      case EVENT_HEARTBEAT:
         // Randomly change direction
         ChangeFacing( 5, 25, 0.1f * d10() );
         break;
      case EVENT_PERCEIVE:
         // Turn toward the creature spotted
         TurnToFace( OBJECT_SELF, GetLastPerceived(), 0.1f * d4() );
         break;
   }
}
```

## Include Files Index

Creating scripts that have multiple include files will cause the associated functions and constants to appear in the Script Assist tool. You can then select the name, right click, and go to the definition. However, adding too many include files seems to crash the toolbox. For this reason I split up the includes over multiple files, as shown below. The double underscore in the file names below puts them at the top of the script list.

```
// __include_ginc_index
/*
   This script will display the ginc* file functions
   and constants in the Script Assist tool.
*/
// RJH 01sep13

#include "ginc_2da"
#include "ginc_ai"
#include "ginc_actions"
#include "ginc_alignment"
#include "ginc_autosave"
#include "ginc_baddie_stream"
#include "ginc_behavior"
#include "ginc_combat"
#include "ginc_companion"
#include "ginc_cutscene"
#include "ginc_debug"
#include "ginc_journal"
#include "ginc_misc"
#include "ginc_object"
#include "ginc_transition"
#include "ginc_vars"

void main()
{
}
```

```
// __include_x0_index
/*
   This script will display the x0* file functions
   and constants in the Script Assist tool.
*/
// RJH 01sep13

#include "x0_i0_anims"
#include "x0_i0_assoc"
#include "x0_i0_behavior"
```

Include Files Index

```
#include "x0_i0_caltrops"
#include "x0_i0_campaign"
#include "x0_i0_combat"
#include "x0_i0_common"
#include "x0_i0_corpses"
#include "x0_i0_db"
#include "x0_i0_debug"
#include "x0_i0_deckmany"
#include "x0_i0_destroy"
#include "x0_i0_enemy"
#include "x0_i0_equip"
#include "x0_i0_henchman"
#include "x0_i0_highlight"
#include "x0_i0_infdesert"
#include "x0_i0_infinite"
#include "x0_i0_match"
#include "x0_i0_modes"
#include "x0_i0_npckilled"
#include "x0_i0_partywide"
#include "x0_i0_petrify"
#include "x0_i0_plotgiver"
#include "x0_i0_position"
#include "x0_i0_projtrap"
#include "x0_i0_secret"
#include "x0_i0_spells"
#include "x0_i0_voice"

#include "x2_i0_spells"

#include "x2_inc_switches"
#include "x2_inc_itemprop"

void main()
{
}
```

## Intelligent Weapon One-Liner Trigger

```
// gtr_iw_one_liner
/*
   This is the 'On Enter' script for a trigger that will cause an
   intelligent weapon to play a one-liner message. The entering object
   must be wielding the weapon for the message to appear. The message
   identifier is set by the 'OneLinerID' variable on the trigger. If
   the 'OnceOnly' variable is set to '1', then the message will only
   be played once.

   The intelligent weapon's Conversation must have a one-line conversation
   entry that has a Condition consisting of two 'gc_local_int' tests:

      1. Is 'X2_L_INTWEAPON_CONV_TYPE' equal to '5'?
      2. Is 'X2_L_INTWEAPON_CONV_NUMBER' equal to the ID?

   Both of these must be satisfied. The Conversation can have multiple
   such one-liners, with each line matching a different 'OneLinerID'.
   The 'sTarget' fields of 'gc_local_int' should be set to $OWNER.
*/
// RJH 23jul11

#include "x2_inc_intweapon"

const string ONCE_ONLY = "OnceOnly";
const string ONE_LINER_ID = "OneLinerID";
const string ALREADY_PLAYED = "already_played";

/* The player is wielding the intelligent weapon, so play the one-liner
   trigger message that has the matching OnceLinerID. One-liner messages
   with OnceOnly set to '1' will only be played once.
*/
void DoOneLiner( object oPlayer, object oWeapon )
{
   // Determine if this is a use-once message
   if ( GetLocalInt( OBJECT_SELF, ONCE_ONLY ) )
   {
      // Check if the message has been spoken before
      int bIsPlayed = GetLocalInt( OBJECT_SELF, ALREADY_PLAYED );
      if ( GetLocalInt( OBJECT_SELF, ALREADY_PLAYED ) )
         return;
   }

   // Flag as played
```

```
    SetLocalInt( OBJECT_SELF, ALREADY_PLAYED, TRUE );

    // Retrieve the conversation ID
    int nID = GetLocalInt( OBJECT_SELF, ONE_LINER_ID );

    // Trigger the quote
    IWPlayTriggerQuote( oPlayer, oWeapon, nID );
}

void main()
{
     object oEnter = GetEnteringObject();
    if ( GetIsObjectValid( oEnter ) )
    {
        // Is entering object equipped with the intelligent weapon?
        object oWeapon = IWGetIntelligentWeaponEquipped( oEnter );
        if ( GetIsObjectValid( oWeapon ) )
        {
            // Generate the appropriate one-liner
            DoOneLiner( oEnter, oWeapon );
        }
    }
}
```

## Invisible Enclosure

```
// invisible_enclosure
/*
   This is the On Enter script for a Trigger blueprint that causes a non-controlled
   creature of the specified type to move toward a waypoint upon entry. It can be
   used, for example, to cover an opening to a fenced-in area where the livestock
   have been given the 'X2_L_SPAWN_USE_AMBIENT = 1' variable setting.
*/
// RJH 07aug13

#include "inc_enclosure"

const string RACIAL_TYPE = "racial_type"; // Variable listing racial type to block
const string ENCLOSE_TAG = "enclose_tag"; // Name of tag to direct entering creatures

void main()
{
    object oTarget = GetEnteringObject();
   if ( ! GetIsObjectValid( oTarget ) )
      return; // Invalid object

   // Only block if the creature is of specified racial type (8 = animal; 28 = all)
   int nRacialType = GetLocalInt( OBJECT_SELF, RACIAL_TYPE );
   if ( GetRacialType( oTarget ) != nRacialType )
      return;

   // Only block if creature is not player-controlled
   object oControl = GetControlledCharacter( oTarget );
   if ( oControl != OBJECT_INVALID )
      return;

   // Check for an ignore enclosure flag
   if ( GetLocalInt( oTarget, IGNORE_ENCLOSURE ) )
      return;

   // Look for the enclosure waypoint
   string sEnclosureWP = GetLocalString( OBJECT_SELF, ENCLOSE_TAG );
   object oWP = GetWaypointByTag( sEnclosureWP );
   if ( ! GetIsObjectValid( oWP ) )
      return; // Invalid waypoint

   // Check if the trigger and waypoint are in the same area
   if ( GetArea( oWP ) != GetArea( OBJECT_SELF ) )
      return;
```

```
   // Get current creature location
   location locCreature = GetLocation( oTarget );

   // Stop the current movement
   AssignCommand( oTarget, ClearAllActions() );

   // Return creature to the waypoint
   int nIsRunning = GetLocalInt( oTarget, RUNNING_RTW );
   if ( !nIsRunning )
      ReturnToWaypoint( oTarget, oWP );
}
```

```
// ww_inc_enclosure
/*
   This is the include file for invisible enclosure calls.
*/
// RJH 07aug13

const string IGNORE_ENCLOSURE = "ignore_enclosure"; // Set on creature to ignore enclosure
const string STORE_IGNORE = "store_ignore";          // Temporary storage for running_rtw
const string HAS_STORE_IGNORE = "has_store_ignore";  // Set to TRUE when STORE_RTW is in use
const string RUNNING_RTW = "running_rtw";            // Set when ReturnToWaypoint is running

// Prototypes
void RestoreIgnoreFlag( object oCreature );
void TemporaryIgnoreFlag( object oCreature );
void ReturnToWaypoint( object oCreature, object oWaypoint );

// Restore the original ignore invisible enclosure flag
void RestoreIgnoreFlag( object oCreature )
{
   if ( !GetIsObjectValid( oCreature ) )
      return;

   if ( GetLocalInt( oCreature, HAS_STORE_IGNORE ) ) {
      // Restore the original ignore enclosure value
      int nIgnore = GetLocalInt( oCreature, STORE_IGNORE );
      SetLocalInt( oCreature, IGNORE_ENCLOSURE, nIgnore );
      SetLocalInt( oCreature, STORE_IGNORE, FALSE );
      SetLocalInt( oCreature, HAS_STORE_IGNORE, FALSE );
   }
}

// Temporarily set the ignore invisible enclosure flag
void TemporaryIgnoreFlag( object oCreature )
```

```
{
   if ( !GetIsObjectValid( oCreature ) )
      return;

   // Check if already in use
   if ( GetLocalInt( oCreature, HAS_STORE_IGNORE ) )
      RestoreIgnoreFlag( oCreature );

   // Store the current value, then clear it
   int nIgnore = GetLocalInt( oCreature, IGNORE_ENCLOSURE );
   SetLocalInt( oCreature, IGNORE_ENCLOSURE, TRUE );
   SetLocalInt( oCreature, STORE_IGNORE, nIgnore );
   SetLocalInt( oCreature, HAS_STORE_IGNORE, TRUE );
}

/* Recursive routine to cause creature to return to waypoint. It
   sets a local integer named "running_rtw" on the creature to TRUE
   while it is running. This is done to prevent multiple executions
   of this routine on the same creature at the same time.
*/
void ReturnToWaypoint( object oCreature, object oWaypoint )
{
   // Compare position of creature to the waypoint
   location locCreature = GetLocation( oCreature );
   location locWaypoint = GetLocation( oWaypoint );
   if ( GetDistanceBetweenLocations( locCreature, locWaypoint ) > 1.0f ) {
      // Flag as active
      SetLocalInt( oCreature, RUNNING_RTW, TRUE );

      // Only change actions if creature is not talking or in combat
      if ( !GetIsInCombat( oCreature ) && !IsInConversation( oCreature ) ) {
         // Check the current action
         int nAction = GetCurrentAction( oCreature );
         if ( nAction == 43 ) {
            // Stop random wandering and move to waypoint
            AssignCommand( oCreature, ClearAllActions() );
            AssignCommand( oCreature, ActionMoveToLocation( locWaypoint ) );
         } else if ( nAction == ACTION_INVALID ) {
            // Creature is doing nothing, so move to waypoint
            AssignCommand( oCreature, ActionMoveToLocation( locWaypoint ) );
         }
      }

      // Call itself in four seconds
      DelayCommand( 4.0f, ReturnToWaypoint( oCreature, oWaypoint ) );
   } else {
```

```
        // Flag as inactive
        SetLocalInt( oCreature, RUNNING_RTW, FALSE );
    }
}
```

## Lightning Bolt Strike

```
// lightning_bolt_strike
/*
   This will create a lightning bolt strike. It should be added as a
   heartbeat script for a Ipoint with a localized name of "Lightning
   bolt". If any creatures are within 5.0f of the Ipoint, it will
   strike the nearest creature. Otherwise it will strike the Ipoint.
   (For best visual effect, raise the Ipoint to about waist height.)

   If the "fx_frequency" local variable is set to a number of seconds
   greater than 5, use it to compute the odds of a strike this time
   through. If the "d6_damage" variable is set to an integer greater
   than 0, it will determine the number of d6 damage. Otherwise damage
   defaults to 3d6. The target is allowed a Reflex save against DC 11 +
   dice/2, or can evade.
*/
// RJH 11sep10

#include "ginc_debug"

const string FX_FREQUENCY = "fx_frequency";
const string DICE_DAMAGE = "d6_damage";

// Apply lightning damage to target
void ApplyDamage( object oTarget, float fDelay = 0.0 )
{
   // Get the dice of damage
   int nDice = GetLocalInt( OBJECT_SELF, DICE_DAMAGE );
   if ( nDice == 0 ) nDice = 3;
   int nDamage = d6( nDice );

   // Reduce on a save
   int nDC = 10 + (nDice / 2) + 1;
   int nDamage = GetReflexAdjustedDamage( nDamage, oTarget, nDC,
      SAVING_THROW_TYPE_ELECTRICITY );

   // Apply damage effect
   effect eDamage = EffectDamage( nDamage,
      DAMAGE_TYPE_ELECTRICAL, DAMAGE_POWER_ENERGY );
   DelayCommand( fDelay,
      ApplyEffectToObject( DURATION_TYPE_INSTANT, eDamage, oTarget ) );
}

void main()
```

## Lightning Bolt Strike

```
{
   // Check for an average frequency variable
   int nFrequency = GetLocalInt( OBJECT_SELF, FX_FREQUENCY );
   if ( Random( nFrequency ) > 5 ) return;

   // Generate the visual effects
   effect eEffect = EffectVisualEffect( VFX_SPELL_HIT_CALL_LIGHTNING );
   effect eLight = EffectVisualEffect( VFX_DUR_LIGHT_BLUE_20 );

   // Use one of these
   if ( Random( 10 ) > 2 ) {
      effect eImpact = EffectVisualEffect( VFX_HIT_SPELL_LIGHTNING );
   } else {
      effect eImpact = EffectVisualEffect( VFX_HIT_AOE_LIGHTNING );
   }

   // Compute a 0.0-5.9 second delay
   float fDelay = IntToFloat( Random( 60 ) ) / 10.0f;

   // Find the nearest creature
   object oNearest = GetNearestCreature( CREATURE_TYPE_IS_ALIVE,
      CREATURE_ALIVE_BOTH, OBJECT_SELF );
   float fDistance = GetDistanceToObject( oNearest );

   // Check if creature is within 5.0
   object oTarget = OBJECT_SELF;
   if ( fDistance < 5.0 ) {
      // Set to the creature
      oTarget = oNearest;

      // Apply damage to creature
      ApplyDamage( oNearest, fDelay );
   }

   // Create visual display
   DelayCommand( fDelay,
      ApplyEffectToObject( DURATION_TYPE_INSTANT, eEffect, oTarget ) );
   DelayCommand( fDelay,
      ApplyEffectToObject( DURATION_TYPE_INSTANT, eImpact, oTarget ) );
   DelayCommand( fDelay,
      ApplyEffectToObject( DURATION_TYPE_TEMPORARY, eLight, oTarget, 0.3f ) );
}
```

## Lock Area Doors at Night

```
// inc_doorlock
/*
   The following routines provide functionality for handling door locking and
   unlocking based on the hour of the day. The time range is defined by storing
   local integers on each door: "hour_locked"/"hour_unlocked", with the value
   between 1 and 24 inclusive. The door must be set lockable. The CheckDoorLocks
   routine should then be called from the area's heartbeat script.
*/
// RJH 11sep10

// These local integer variables should be set on the door placeable
const string HOUR_DOOR_LOCKED = "hour_locked";
const string HOUR_DOOR_UNLOCKED = "hour_unlocked";

// Constants used for door lock processing
const string RR_DLCK_CONTROLLER = "nw_waypoint001"; // Res Ref
const string LAST_HOUR_DOOR_CHECK = "DLCK_hour_check";
const string DLCKC_PREFIX = "DLCK";
const string DLCKC_TAG = "DLCK_TAG";
const string DOOR_PREFIX = "DOOR";
const string DOOR_NUM = "DOOR_NUM";

// Prototypes
void SetDoorTimeLock( object oDoor, int nHour );
object CreateDLKController( object oArea, string sDLCKControllerTag );
void CheckDoorLocks( object oArea );

/* Lock the door only if the hour nHour is between the
   hour to lock and the hour to unlock, as determined by
   the local door variables HOUR_DOOR_LOCKED/_UNLOCKED.
*/
void SetDoorTimeLock( object oDoor, int nHour )
{
   if( !GetIsObjectValid( oDoor ) )
      return;

   // Check for a valid lock range
   int nHourLocked = GetLocalInt( oDoor, HOUR_DOOR_LOCKED );
   if ( ( nHourLocked < 1 ) || ( nHourLocked > 24 ) )
      return;

   // Check for a valid unlock range
   int nHourUnlocked = GetLocalInt( oDoor, HOUR_DOOR_UNLOCKED );
```

```
   if ( ( nHourUnlocked < 1 ) || ( nHourUnlocked > 24 ) )
      return;

   // Determine whether door should be locked
   int nLock = FALSE;
   if ( nHourLocked < nHourUnlocked ) {
      // Not locked at midnight
      if ( ( nHour >= nHourLocked ) && ( nHour < nHourUnlocked ) )
         nLock = TRUE;
   } else {
      // Locked at midnight
      if ( ( nHour >= nHourLocked ) || ( nHour < nHourUnlocked ) )
         nLock = TRUE;
   }

   // Set the door lock state
   SetLocked( oDoor, nLock );
   if ( nLock ) {
      // If locked, close the door
      DelayCommand ( 0.1f, AssignCommand( oDoor,
         ActionCloseDoor( oDoor ) ) );
   }
}

// Create a door lock controller
object CreateDLKController( object oArea, string sDLCKControllerTag )
{
   // Create the door lock controller
   object oTarget = GetFirstObjectInArea( oArea );
   object oDLKController = CreateObject( OBJECT_TYPE_WAYPOINT,
      RR_DLCK_CONTROLLER, GetLocation( oTarget ), FALSE,
      sDLCKControllerTag );
   if ( !GetIsObjectValid( oDLKController ) )
      return oDLKController;

   // Look up the doors in the area
   int nNth = 1;
   while( GetIsObjectValid( oTarget ) ) {
      // Check for a door
      int nType = GetObjectType( oTarget );
      if ( nType == OBJECT_TYPE_DOOR ) {
         // Look for the lock/unlock variables
         int nHourLocked = GetLocalInt( oTarget, HOUR_DOOR_LOCKED );
         int nHourUnlocked = GetLocalInt( oTarget, HOUR_DOOR_UNLOCKED );

         // Check if a time range has been assigned
```

```
         if ( ( nHourLocked > 0 ) && ( nHourUnlocked > 0 ) ) {
              SetLocalObject( oDLKController,
                DOOR_PREFIX + IntToString( nNth ),
                oTarget );
            nNth++;
         }
      }
      oTarget = GetNextObjectInArea( OBJECT_SELF );
   }
   nNth--;
   SetLocalInt( oDLKController, DOOR_NUM, nNth );

   return oDLKController;
}

// Set the lock state of the doors in the area.
void CheckDoorLocks( object oArea )
{
   // Get the hour (range = [1, 24])
   int nHour = GetTimeHour() + 1;

   // Exit if the hour is unchanged
   int nLastHour = GetLocalInt( oArea, LAST_HOUR_DOOR_CHECK );
   if ( nLastHour == nHour )
      return;
   SetLocalInt( oArea, LAST_HOUR_DOOR_CHECK, nHour );

   // Look for a door lock controller
   string sAreaTag = GetTag( oArea );
   string sDLKControllerTag = DLCKC_PREFIX + "_" + sAreaTag;
   object oDLKController = GetWaypointByTag( sDLKControllerTag );

   // if controlled doesn't exist then create it
   if ( !GetIsObjectValid( oDLKController ) )
      oDLKController = CreateDLKController( oArea, sDLKControllerTag );

   // Cycle through the doors and set lock state appropriately
   int i;
   int nDoors = GetLocalInt( oDLKController, DOOR_NUM );
   for ( i = 1; i <= nDoors; i++ ) {
      // Set the door lock state
      object oDoor = GetLocalObject( oDLKController,
            DOOR_PREFIX + IntToString( i ) );
      SetDoorTimeLock( oDoor, nHour );
   }
}
```

## Marking Removed Items as Stolen

```
// mark_as_stolen
/*
   This script can be placed in the 'On Inventory Disturbed' property field
   of a container. It is used to flag any original contents as stolen if they
   are removed. It can be used, for example, on a locked container of a friendly
   NPC. When a character adds an item to the container, it is flagged as a stored
   item by setting a local variable on the object. If an item is removed from the
   container that does not have this variable set, then the stolen flag on the
   object is set to true.
*/
// RJH 11sep10

const string STORE_ITEM = "is_stored_item";

// Return true if item oItem is in the inventory of the calling object.
int GetHasObject( object oItem )
{
   object oInv = GetFirstItemInInventory();
   while ( GetIsObjectValid( oInv ) ) {
      // Return true on a match
      if ( oItem == oInv ) return TRUE;
      oInv = GetNextItemInInventory();
   }
   return FALSE;
}

void main()
{
   object oItem = GetInventoryDisturbItem();

   // Check if item is in inventory
   if ( GetHasObject( oItem ) ) {
      // Add flag to mark it as a stored item
      SetLocalInt( oItem, STORE_ITEM, 1 );
   } else if ( GetLocalInt( oItem, STORE_ITEM ) == 1 ) {
      // Removed item was flagged as stored
      SetLocalInt( oItem, sStoreItem, 0 );
   } else {
      // The item must be stolen
      SetStolenFlag( oItem, TRUE );
   }
}
```

## Open Sarcophagus

```
// sarcophagus_open
/*
   Place this script in the "On Open Script" field of a useable sarcophagus.
   The first time the sarcophagus is opened, a puff of dirt is released over
   the opening. Thereafter the effect is disabled.
*/
// 14jan12 RJH

const string HasOpened = "hasOpenedSarc";
const string DirtPuff = "fx_dirt_puff";

void DoDirtPuff()
{
   // Gather data
   object oArea = GetArea( OBJECT_SELF );
   location locSelf = GetLocation( OBJECT_SELF );
   string sResRef = GetResRef( OBJECT_SELF );
   float fFacing = GetFacing( OBJECT_SELF );

   // Set positional parameters based on the blueprint
   float fHeight = 0.5f;
   if ( ( StringCompare( sResRef, "plc_mc_sarcoph1" ) == 0 ) ||
        ( StringCompare( sResRef, "plc_mc_sarcoph2" ) == 0 ) )
   {
     fFacing += 255.0f;
   } else if (
       ( StringCompare( sResRef, "plc_mc_sarcoph3" ) == 0 ) ||
       ( StringCompare( sResRef, "plc_mc_sarcoph5" ) == 0 ) )
   {
     fFacing += 300.0f;
     fHeight = 0.6f;
   }

   // Compute the effect placement
   float fSin = 1.0f * sin( fFacing );
   float fCos = 1.0f * cos( fFacing );
   vector vSelf = GetPositionFromLocation( locSelf );
   vector vOffset = Vector( fCos, fSin, fHeight );
   location locEffect = Location( oArea, vSelf + vOffset, fFacing );

   // Apply the puff of smoke at the location
   effect eCloud = EffectNWN2SpecialEffectFile(  DirtPuff );
   ApplyEffectAtLocation( DURATION_TYPE_INSTANT, eCloud, locEffect );
```

```
}

void main()
{
   int nHasOpened = GetLocalInt( OBJECT_SELF, HasOpened );
   if ( nHasOpened == FALSE ) {
      // Trigger a puff of smoke over the opening
      DoDirtPuff();
      SetLocalInt( OBJECT_SELF, HasOpened, TRUE );
   }
}
```

## Orient Camera

```
// orient_camera
/*
   This is a "On Click Script" for use with a door that has a transition to a
   waypoint in the same area. Normally, such a transition will leave the camera
   facing in the original direction, which may be an awkward view for the player.
   This script will check whether the transition target for the open door is a
   waypoint, then orient the camera in the direction the waypoint is facing. It
   will then transition the party to the target.
*/
// RJH 27jun11

int StartingConditional()
{
   object oPC = GetFirstPC();
   if ( GetIsOpen( OBJECT_SELF ) )
   {
      object oTarget = GetTransitionTarget( OBJECT_SELF );
      if ( GetIsObjectValid( oTarget ) )
      {
         // Check for a waypoint
         int nType = GetObjectType( oTarget );
         if ( nType == OBJECT_TYPE_WAYPOINT )
         {
            // Orient the PC camera to face the waypoint direction
            float fTarget = GetFacing( oTarget );
            AssignCommand( oPC, SetCameraFacing( fTarget ) );
         }

         // Perform the transition
         JumpPartyToArea( oPC, oTarget );
      }
   }

   return ( FALSE );
}
```

## Placeable Damage Bug Fix

```
// on_placeable_damaged
/*
   This script should be added to the 'On Damaged Script' property of a placeable or
   door. Due to a bug in the game, the object hardness rating is not applied to reduce
   the amount of damage dealt. This script corrects the problem by healing up to the
   hardness rating on the first attack during a round.
*/
// RJH 11sep10

void main()
{
   // Heal hit points up to the hardness rating.
   int nHardness = GetHardness( OBJECT_SELF );
   int nDamageDealt = GetTotalDamageDealt();
   int nHeal = ( nDamageDealt > nHardness ) ? nHardness : nDamageDealt;
   ApplyEffectToObject( DURATION_TYPE_INSTANT, EffectHeal( nHeal ), OBJECT_SELF );
}
```

```
// on_placeable_attacked
/*
   This script should be added to the 'On Melee Attacked Script' property
   of a placeable or door. The first time through, add a hit point padding
   equal to the hardness rating. This corrects for the case where the
   death blow does not allow the object to get an opportunity to heal.
*/
// RJH 11sep10
const string HAS_PAD_HP = "hasPadHitpoints";

void main() {
   int bHasPadHitpoints = GetLocalInt( OBJECT_SELF, HAS_PAD_HP );
   if ( ! bHasPadHitpoints ) {
      // Apply bonus hit points then heal that amount
      int nHardness = GetHardness( OBJECT_SELF );
      effect nBHP = EffectBonusHitpoints( nHardness );
      ApplyEffectToObject( DURATION_TYPE_PERMANENT,
         SupernaturalEffect( nBHP ), OBJECT_SELF );
      ApplyEffectToObject( DURATION_TYPE_INSTANT,
         EffectHeal( nHardness ), OBJECT_SELF );

      // Flag as completed
      SetLocalInt( OBJECT_SELF, HAS_PAD_HP, TRUE );
   }
}
```

## Restocking a Store

```
// winc_restock_store
/*
   The following function calls can be used to perform restocking of a
   store. They should be called from the 'On Open Store' script in a
   store blueprint. The Add*StockRate routines perform the initialization
   of the restocking rate information. Each visit thereafter, RestockStore
   should be called to update the store inventory.

   Note that restock scripts are best used for items that are frequently
   expended by the party or those items that are sought by multiple party
   members and have a non-trivial price tag, such as longswords or bows.
   Low cost items such as clubs, daggers, torches and clothing, are best
   handled with the 'Infinite' setting in the store dialog.
*/
// RJH 07jul11

// Restock date tracking variables
const string RS_PRIOR_VISIT = "rs_prior_visit";
const string RS_TWODAY_DATE = "rs_twoday_date";
const string RS_WEEK_DATE = "rs_weekday_date";

// Inventory tracking variables
const string RS_NO_RESTOCK = "rs_no_restock_items";
const string RS_INVENTORY_TYPES = "rs_inventory_types";

// Prefixes used for creating unique local variable names
const string RS_ID_PREFIX = "rs_id_";
const string RS_NO_PREFIX = "rs_no_";
const string RS_PER_PREFIX = "rs_per_";
const string RS_TAG_PREFIX = "rs_tag_";
const string RS_CNT_PREFIX = "rs_cnt_";
const string RS_MAX_PREFIX = "rs_max_";

// Prototypes

// Initialization routines
void AddStockRate( int nPeriod, string sTag, int nCopies, int nMax );
void AddTwoDayStockRate( string sTag, int nCopies, int nMax );
void AddOneWeekStockRate( string sTag, int nCopies, int nMax );

// Calls that perform the inventory tracking and restocking
void UpdateInventory();
void RestockItem( string sTag, int nCopies, int nMax );
```

```
void RestockItems( int nPeriod, int nCount );
void RestockStore();

/* Add the item and its restocking parameters to the list of
   items to be restocked. The nPeriod determines the restocking
   interval, sTag is the tag of the item to be restocked,
   nCopies is the number of copies to restock after nPeriod,
   and nMax is the maximum inventory.
*/
void AddStockRate( int nPeriod, string sTag, int nCopies, int nMax )
{
   string sTagLC = GetStringLowerCase( sTag );
   int nNumRestock = GetLocalInt( OBJECT_SELF, RS_NO_RESTOCK ) + 1;
   string sCount = IntToString( nNumRestock );
   SetLocalInt( OBJECT_SELF, RS_PER_PREFIX + sCount, nPeriod );
   SetLocalString( OBJECT_SELF, RS_TAG_PREFIX + sCount, sTagLC );
   SetLocalInt( OBJECT_SELF, RS_CNT_PREFIX + sCount, nCopies );
   SetLocalInt( OBJECT_SELF, RS_MAX_PREFIX + sCount, nMax );
   SetLocalInt( OBJECT_SELF, RS_NO_RESTOCK, nNumRestock );
}

// Convenience call for AddStockRate with a two-day period
void AddTwoDayStockRate( string sTag, int nCopies, int nMax )
{
   AddStockRate( 2, sTag, nCopies, nMax );
}

// Convenience call for AddStockRate with a one-week period
void AddOneWeekStockRate( string sTag, int nCopies, int nMax )
{
   AddStockRate( 7, sTag, nCopies, nMax );
}

/* Clear the old inventory records, then cycle through the
   inventory, saving information about each unique item type
   as a pair of local variables on the store:

   1) Stock Count variable (string):
      Name = RS_NO_PREFIX + item tag
      Content = (int) item stock count
   2) Stock Index variable (int):
      Name = RS_ID_PREFIX + incremental index (1, 2, ...)
      Content =  item tag

   Variable 1 is used for quick lookup of an item count when
   the item tag is known. Variable 2 is used for fast cleanup
```

```
   of Variable 1 when the inventory is being re-initialized.
*/
void UpdateInventory()
{
   /* Incrementally clear the previous inventory by using
      the Stock Index variable to lookup the name of the
      Stock Count variable, then initialize both.
   */
   int i;
   int nItemTypes = GetLocalInt( OBJECT_SELF, RS_INVENTORY_TYPES );
   for ( i = 1; i <= nItemTypes; i++ ) {
      string sStockID = RS_ID_PREFIX + IntToString( i );
      string sTag = GetLocalString( OBJECT_SELF, sStockID );
      SetLocalString( OBJECT_SELF, sStockID, "" );
      string sStockCount = RS_NO_PREFIX + sTag;
      SetLocalInt( OBJECT_SELF, sStockCount, 0 );
   }

   /* Cycle through the store inventory. If an item does not
      have a stock count, set the Stock Index variable to the
      item tag and set the Stock Count variable to 1. Otherwise,
      increment the Stock Count variable by 1.
   */
   nItemTypes = 0;
   object oInventory = GetFirstItemInInventory( OBJECT_SELF );
   while( oInventory != OBJECT_INVALID ) {
      string sTag = GetStringLowerCase( GetTag( oInventory ) );
      string sStockCount = RS_NO_PREFIX + sTag;
      int nStockCount = GetLocalInt( OBJECT_SELF, sStockCount );
      if ( nStockCount > 0 ) {
         // Increment the count
         SetLocalInt( OBJECT_SELF, sStockCount, nStockCount + 1 );
      } else {
         // Check if the infinite flag is set
         if ( GetInfiniteFlag( oInventory ) ) {
            // Set to a large number
            SetLocalInt( OBJECT_SELF, sStockCount, 10000 );
         } else {
            // Initialize the count
            SetLocalInt( OBJECT_SELF, sStockCount, 1 );
         }

         // Store the tag
         nItemTypes++;
         string sStockID = RS_ID_PREFIX + IntToString( nItemTypes );
         SetLocalString( OBJECT_SELF, sStockID, sTag );
```

```
      }
      oInventory = GetNextItemInInventory( OBJECT_SELF );
   }

   // Save a count of the inventory
   SetLocalInt( OBJECT_SELF, RS_INVENTORY_TYPES, nItemTypes );
}

/* This function will add up to nCopies copies of the item with
   the tag sTag to the current inventory, for a maximum limit of
   nMax copies.
*/
void RestockItem( string sTag, int nCopies, int nMax )
{
   string sStockCount = RS_NO_PREFIX + sTag;
   int nStockCount = GetLocalInt( OBJECT_SELF, sStockCount );

   // Check if the inventory is at the maximum
   if ( nStockCount >= nMax ) {
      return;
   }

   // Determine the amount to add
   int nAdd = nCopies;
   if (  nCopies > nMax - nStockCount ) {
      nAdd = nMax - nStockCount;
   }

   // Add the copies
   int i;
   for ( i = 0; i < nAdd; i++ ) {
      // Create a matching item in the inventory
      object oItem = CreateItemOnObject( sTag, OBJECT_SELF );

      // Increase the stack size of munitions
      int nType = GetBaseItemType( oItem );
      switch ( nType )
      {
         case BASE_ITEM_ARROW:
         case BASE_ITEM_BOLT:
         case BASE_ITEM_BULLET:
         case BASE_ITEM_DART:
         case BASE_ITEM_THROWINGAXE:
            SetItemStackSize( oItem, 99, FALSE );
            break;
      }
```

```
   }
}

/* The list of items to be restocked is searched for items that
   are restocked on a time interval of nPeriod days. Each time a
   matching item is found, that item is restocked with a call to
   RestockItem. The nCount value determines the number of restock
   intervals that have passed.
*/
void RestockItems( int nPeriod, int nCount )
{
   int i;

   // Run through the list of items to restock
   int nNumRestock = GetLocalInt( OBJECT_SELF, RS_NO_RESTOCK );
   for ( i = 1; i <= nNumRestock; i++ ) {
      string sCount = IntToString( i );
      int nRate = GetLocalInt( OBJECT_SELF, RS_PER_PREFIX + sCount );
      if ( nRate == nPeriod ) {
         /* The restock rate matches the selected period,
            so obtain the stocking information.
         */
         string sTag = GetLocalString( OBJECT_SELF, RS_TAG_PREFIX + sCount );
         int nCopies = GetLocalInt( OBJECT_SELF, RS_CNT_PREFIX + sCount );
         int nMax = GetLocalInt( OBJECT_SELF, RS_MAX_PREFIX + sCount );

         // Restock the item
         RestockItem( sTag, nCount * nCopies, nMax );
      }
   }
}

/* This is the main routine for performing restocking of a store.
   The first time this is called, it stores the starting dates
   for the restocking. Thereafter, the current date is compared
   to the date of the last restocking. If two days have passed,
   then the two-day restocking is performed. Likewise for the
   one-week restocking intervals. This can be expanded with other
   restocking cycles to suite your module's needs.

   To expedite the process, on the first restocking, an inventory
   is taken that puts a lookup list of the stock information in
   local variables on the store object.
*/
void RestockStore()
{
```

```
   // Check for a previous visit
   int bPriorVisit = GetLocalInt( OBJECT_SELF, RS_PRIOR_VISIT );
   if ( !bPriorVisit ) {
      /* Initialize the restock trackers in such a manner
         that four restock cycles will be performed. This
         will ensure that the store is adequately stocked.
      */
      SetLocalInt( OBJECT_SELF, RS_TWODAY_DATE, nDate - (4*2) );
      SetLocalInt( OBJECT_SELF, RS_ONEWEEK_DATE, nDate - (4*7) );
      SetLocalInt( OBJECT_SELF, RS_PRIOR_VISIT, TRUE );
   }

   // Check for a two day restock
   int nTwoDay = GetLocalInt( OBJECT_SELF, RS_TWODAY_DATE );
   if ( ( nDate - nTwoDay ) >= 2 ) {
      if ( bHaveInventory == FALSE ) {
         // Build an updated inventory
         UpdateInventory();
         bHaveInventory = TRUE;
      }

      // 2+ days have passed since the last 2-day restock
      RestockItems( 2, ( nDate - nTwoDay ) / 2 );
      SetLocalInt( OBJECT_SELF, RS_TWODAY_DATE, nDate );
   }

   // Check for a one week restock
   int nOneWeek = GetLocalInt( OBJECT_SELF, RS_ONEWEEK_DATE );
   if ( ( nDate - nOneWeek ) >= 7 ) {
      if ( bHaveInventory == FALSE ) {
         // Build an updated inventory
         UpdateInventory();
         bHaveInventory = TRUE;
      }

      // 1+ week has passed since the last 1-week restock
      RestockItems( 7, ( nDate - nOneWeek ) / 7 );
      SetLocalInt( OBJECT_SELF, RS_WEEK_DATE, nDate );
   }
}
```

```
// restock_shop
/*
   Restock a shop.
*/
```

## Restocking a Store

```
// RJH 07jul11

#include "winc_restock_store"

const string INIT_RESTOCK = "init_store_restock";

void main()
{
   int bInitRestock = GetLocalInt( OBJECT_SELF, INIT_RESTOCK );
   if ( !bInitRestock ) {
      /* The following calls will establish the items that will be
         restocked for this store. Restocking occurs in two-day and
         one-week intervals. The parameters passed to each call are
         the tag of the item to be restocked, the number of the item
         to replace following each interval, and the maximum number
         of said items to allow. Hence:

           AddTwoDayStockRate( "NW_WAXHN001", 2, 5 );

         will set the restock to allow 2 copies of "NW_WAXHN001" to be
         added the store inventory every two days, up to a maximum of 5
         copies in the inventory.
      */
      AddTwoDayStockRate( "NW_WAXHN001", 1, 2 );       // Handaxe
      AddTwoDayStockRate( "NW_IT_MPOTION005", 1, 4 );  // Potion of Barkskin
      AddOneWeekStockRate( "NW_WAMMAR004", 1, 2 );     // Piercing arrow
      AddOneWeekStockRate( "x2_it_iwoodclub", 1, 1 );  // Ironwood club
      AddOneWeekStockRate( "x2_it_iwoodstaff", 1, 1 ); // Ironwood staff

      // Only need to initialize once
      SetLocalInt( OBJECT_SELF, INIT_RESTOCK, TRUE );
   }

   // Restock depleted inventory
   RestockStore();
}
```

## Returning Weapon

```
// i_returning_weapon_aq
/*
   This is the acquired item script for a returning weapon.
   It needs to be saved as i_<WEAPON_TAG>_aq.
*/
// RJH 11sep10

#include "winc_tag_based_scripts"

void main()
{
    object oCreature = GetModuleItemAcquiredBy();
    object oWeapon = GetModuleItemAcquired();

   AcquireReturnWeapon( oCreature, oWeapon );
}
```

```
// i_returning_weapon_eq
/*
   This is the equipped item script for a returning weapon.
   It needs to be saved as i_<WEAPON_TAG>_eq.
*/
// RJH 11sep10

#include "winc_tag_based_scripts"

void main()
{
   object oCreature = GetPCItemLastEquippedBy();
   object oWeapon = GetPCItemLastEquipped();

   EquipReturnWeapon( oCreature, oWeapon );
}
```

```
// i_returning_weapon_ue
/*
   This is the unequip item script for a returning weapon.
   It needs to be saved as i_<WEAPON_TAG>_ue.
*/
// RJH 11sep10

#include "winc_tag_based_scripts"
```

```
void main()
{
    object oCreature = GetPCItemLastUnequippedBy();
    object oWeapon = GetPCItemLastUnequipped();

   UnequipReturnWeapon( oCreature, oWeapon );
}
```

```
// inc_tag_based_scripts
/*
   The following routines are for use with tag-based scripting.
*/
// RJH 11sep10

const string CURR_OPPONENT = "RW_opponent";
const string CURR_LOCATION = "RW_location";
const string NEXT_OPPONENT = "RW_next_opponent";

// Prototypes
void AcquireReturnWeapon( object oCreature, object oWeapon );
void ReadyReturnWeapon( object oCreature, object oWeapon, int nAttemptsLeft );
void EquipReturnWeapon( object oCreature, object oWeapon );
void UnequipReturnWeapon( object oCreature, object oWeapon );

/* A returning weapon has been unequiped. If the owner is in
   combat, create a new copy in the owner's inventory. Store
   the opponent object and the owners location on the copy.
   The weapon owner is switched to passive melee in order
   to prevent a switch to melee.
*/
void UnequipReturnWeapon( object oCreature, object oWeapon )
{
   // Clean up variables
   DeleteLocalObject( oWeapon, CURR_OPPONENT );
   DeleteLocalObject( oWeapon, NEXT_OPPONENT );
   DeleteLocalLocation( oWeapon, CURR_LOCATION );

   // Check for validity
   if ( !GetIsObjectValid( oCreature ) || !GetIsObjectValid( oWeapon ) )
      return;

   // Is the creature in combat?
   if ( !GetIsInCombat( oCreature ) )
      return;
```

```
    // Check the stack size
    if ( GetItemStackSize( oWeapon ) != 1 )
        return;

    // Get the current opponent
    object oOpponent = GetAttackTarget( oCreature );

    // Clear actions to prevent switching to melee
    int nAction = GetCurrentAction( oCreature );
    if ( nAction == ACTION_ATTACKOBJECT ) {
        // Clear current actions
        AssignCommand( oCreature, ClearAllActions( TRUE ) );

        // Reset to passive attack
        if ( GetIsObjectValid( oOpponent ) ) {
            AssignCommand( oCreature, ActionAttack( oOpponent, TRUE ) );
        }
    }

    // Create a new copy of the item
    string sResref = GetResRef( oWeapon );
    object oCopyWeapon = CreateItemOnObject( sResref, oCreature, 1, "", FALSE );

    // Check if valid
    if ( GetIsObjectValid( oCopyWeapon ) ) {
        // Remove the old weapon, if it exists
        DestroyObject( oWeapon, 0.1f, FALSE );

        // Must be identified to equip
        SetIdentified( oCopyWeapon, TRUE );

        // Store the opponent object on the item
        SetLocalObject( oCopyWeapon, CURR_OPPONENT, oOpponent );

        // Store the owner's location on the item
        location locCreature = GetLocation( oCreature );
        SetLocalLocation( oCopyWeapon, CURR_LOCATION, locCreature );
    }
}

/*  If the creature's weapon slot is available and the creature is
    currently idle, move the weapon into the slot. Otherwise, wait
    a short interval and try again. Repeat until the weapon is
    equipped or all attempts have been used up.
*/
void ReadyReturnWeapon( object oCreature, object oWeapon, int nAttemptsLeft )
```

```
{
   // Check whether to give up
   if ( nAttemptsLeft < 1 ) {
      return;
   }

   // Check if slot is already occupied
   object oInSlot = GetItemInSlot( INVENTORY_SLOT_RIGHTHAND, oCreature );
   if ( GetIsObjectValid( oInSlot ) ) {
      return;
   }

   // Check if idle
   int nAction = GetCurrentAction( oCreature );
   if ( nAction == ACTION_INVALID ) {
      AssignCommand( oCreature, ActionEquipItem( oWeapon,
         INVENTORY_SLOT_RIGHTHAND ) );
   }

   // Repeat loop
   DelayCommand( 0.5f, ReadyReturnWeapon( oCreature, oWeapon, nAttemptsLeft - 1 ) );
}

/* A returning weapon has entered the inventory. If the item
   has an opponent object stored as a variable, treat the
   weapon as though it has returned to the owner. Call the
   EquipReturnWeapon to equip the weapon.
*/
void AcquireReturnWeapon( object oCreature, object oWeapon )
{
   if ( !GetIsObjectValid( oWeapon ) ) {
      return;
   }

   // Get the current opponent from an item variable
   object oOpponent = GetLocalObject( oWeapon, CURR_OPPONENT );

   // Clean up local variables
   DeleteLocalObject( oWeapon, CURR_OPPONENT );
   DeleteLocalObject( oWeapon, NEXT_OPPONENT );

   // Make sure that stack remains 1 item tall
   SetItemStackSize( oWeapon, 1, FALSE );

   // Check for an opponent object
   if ( !GetIsObjectValid( oOpponent ) ) {
```

```
      return;
   }

   // Check if slot is occupied
   object oInSlot = GetItemInSlot( INVENTORY_SLOT_RIGHTHAND, oCreature );
   if ( GetIsObjectValid( oInSlot ) ) {
      return;
   }

   // If doing something, stop action
   object oNewOpponent = GetAttackTarget( oCreature );
   int nAction = GetCurrentAction( oCreature );
   if ( nAction != ACTION_INVALID ) {
      AssignCommand( oCreature, ClearAllActions( TRUE ) );
   }

   // Equip the weapon
   DelayCommand( 0.1f, ReadyReturnWeapon( oCreature, oWeapon, 10 ) );

   // Check if previously on the attack
   if ( nAction != ACTION_ATTACKOBJECT ) {
      return;
   }

   // Switch to new opponent if valid
   if ( GetIsObjectValid( oNewOpponent ) ) {
      oOpponent = oNewOpponent;
   }

   // Store target on item
   SetLocalObject( oWeapon, NEXT_OPPONENT, oOpponent );
}

/* A returning weapon has been placed in the right hand
   slot. If the owner has not moved significantly and an
   opponent object is stored on the weapon, re-engage the
   for in combat.
*/
void EquipReturnWeapon( object oCreature, object oWeapon )
{
   location locLast = GetLocalLocation( oWeapon, CURR_LOCATION );
   object oOpponent = GetLocalObject( oWeapon, NEXT_OPPONENT );

   // Clean up variables
   DeleteLocalObject( oWeapon, NEXT_OPPONENT );
   DeleteLocalLocation( oWeapon, CURR_LOCATION );
```

```
    // If the NEXT_OPPONENT is set, attack that target
    if ( !GetIsObjectValid( oOpponent ) ) {
        return;
    }

    // Check whether the creature has moved significantly
    location locCurr = GetLocation( oCreature );
    float fDistance = GetDistanceBetweenLocations( locLast, locCurr );
    if ( fDistance < 0.5f ) {
        // Restart attack
        AssignCommand( oCreature, ActionAttack( oOpponent ) );
    }
}
```

## Set Name from Variables

```
// ga_set_name_from_variables
/*
   This action script will cause the conversation owner to set its name from
   local string variables. This causes the creature name to be set upon
   clicking on the creature. As this script may be run by multiple creatures,
   make sure the conversation is set to run 'Always'.
*/
#include "ginc_param_const"

const string FIRST_NAME = "FIRST_NAME";
const string LAST_NAME = "LAST_NAME";
const string HAS_SET_NAME = "ww_has_set_name";

void main()
{
   // Check if name was already set
   int nHasSet = GetLocalInt( OBJECT_SELF, HAS_SET_NAME );
   if ( nHasSet != 0 )
      return;

   // Set first name, if any
   string sName = GetLocalString( OBJECT_SELF, FIRST_NAME );
   if ( GetStringLength( sName ) > 0 )
      SetFirstName( OBJECT_SELF, sName );

   // Set last name, if any
   sName = GetLocalString( OBJECT_SELF, LAST_NAME );
   if ( GetStringLength( sName ) > 0 )
      SetLastName( OBJECT_SELF, sName );

   // Flag as set
   SetLocalInt( OBJECT_SELF, HAS_SET_NAME, 1 );
}
```

## Sound Activattion Trigger

```
// activate_sound
/*
   This is the On Enter script for the "Activate sound" trigger. When a
   creature enters the region, check for a tag match against the "NPCTag".
   On a success, turn on the sound with the "SoundTag" tag.
*/

const string NPC_TAG = "NPCTag";
const string SOUND_TAG = "SoundTag";

void main()
{
   // Check if the entering object is valid
   object oTarget = GetEnteringObject();
   if ( ! GetIsObjectValid( oTarget ) )
      return; // Invalid object

   // Check the entering object tag
   string sTargetTag = GetTag( oTarget );
   string sNPCTag = GetLocalString( OBJECT_SELF, NPC_TAG );
   if ( StringCompare( sNPCTag, sTargetTag ) != 0 )
      return; // Found a match

   // Activate the sound
   string sSoundTag = GetLocalString( OBJECT_SELF, SOUND_TAG );
   object oSound = GetNearestObjectByTag( sSoundTag, oTarget, 0 );
   if ( GetIsObjectValid( oSound ) )
      SoundObjectPlay( oSound );
}
```

```
// som_deactivate_sound
/*
   This is the On Exit script for the "Activate sound" trigger. When a
   creature exits the region, check for a tag match against the "NPCTag".
   On a success, turn off the sound with the "SoundTag" tag.
*/

const string NPC_TAG = "NPCTag";
const string SOUND_TAG = "SoundTag";

void main()
{
   // Check if the exiting object is valid
```

```
    object oTarget = GetExitingObject();
    if ( ! GetIsObjectValid( oTarget ) )
        return; // Invalid object

    // Check the exiting object tag
    string sTargetTag = GetTag( oTarget );
    string sNPCTag = GetLocalString( OBJECT_SELF, NPC_TAG );
    if ( StringCompare( sNPCTag, sTargetTag ) != 0 )
        return; // Found a match

    // Deactivate the sound
    string sSoundTag = GetLocalString( OBJECT_SELF, SOUND_TAG );
    object oSound = GetNearestObjectByTag( sSoundTag, oTarget, 0 );
    if ( GetIsObjectValid( oSound ) )
        SoundObjectStop( oSound );
}
```

## Spawn a Party Member

```
// spawn_party_member
/*
   This example shows how a creature can be added to the party. The
   hangout spot for this NPC will have the tag "spawn_rita_rogueblood".
   See the notes in "ginc_companion" (and "kinc_companion" in the OC.)
*/
// RJH 11sep10

#include "ginc_companion"

const string RES_RITA = "rr_rita_rogueblood"; // Resource name
const string TAG_RITA = "rita_rogueblood"; // Tag
const string ROS_RITA = "rita_rogueblood"; // Roster name

void main() {
   object oPC = GetFirstPC();

   // Add the creature blueprint with the resource name
   // "rr_rita_rogueblood" to the roster of available
   // party members.
   InitializeCompanion( ROS_RITA, TAG_RITA, RES_RITA );

   // Check if the creature is available
   if ( GetIsRosterMemberAvailable( ROS_RITA ) ) {
      // Spawn the roster member next to the PC
      location locPC = GetLocation( oPC );
      object oRR = SpawnRosterMember( ROS_RITA, locPC );

      // Add the roster member to party
      AddRosterMemberToParty( ROS_RITA, oPC );

      // Change the script set to use the companion scripts.
      // This may be unnecessary if the blueprint was already
      // configured to use the b_Companionscripts set, or
      // if the character's scripts have been customized.
      SetCreatureScriptsToSet( oRR, SCRIPTSET_PLAYER_DEFAULT );
   }
}
```

## Swag Bag Treasure

```
// i_swag_bag_ac
/*
   This illustrates the tag-based scripting technique. The "ac"
   at the end of the script name causes the module to run this
   script when the item with the tag "swag_bag" is acquired.
   The script will add a few sundry items to the Container item,
   then flag it to prevent future execution.
*/
// RJH 11sep10

#include "x2_inc_treasure"

const string STOCK_BAG = "stock_bag";
const int AMMO_PCT = 25;
const int DISP_PCT = 10;
const int ITEM_PCT = 60;
const int MAGIC_PCT = 5;

void main()
{
   object oPC = GetModuleItemAcquiredBy();
   object oItem = GetModuleItemAcquired();

   // This fires when the item is acquired by a PC
   if ( GetLocalInt( oItem, STOCK_BAG ) != 1 ) {
      // Generate random treasure
      int nTreasureType = X2_DTS_TYPE_MUNDANE;
      nTreasureType |= ( Random( 100 ) < AMMO_PCT ) ? X2_DTS_TYPE_AMMO : 0;
      nTreasureType |= ( Random( 100 ) < DISP_PCT ) ? X2_DTS_TYPE_DISP : 0;
      nTreasureType |= ( Random( 100 ) < ITEM_PCT ) ? X2_DTS_TYPE_ITEM : 0;
      nTreasureType |= ( Random( 100 ) < MAGIC_PCT ) ? X2_DTS_TYPE_MAGIC : 0;
      DTSGenerateTreasureOnContainer( oItem, oPC,
         X2_DTS_CLASS_LOW, nTreasureType );

      // Only do this once
      SetLocalInt( oItem, STOCK_BAG, 1 );
   }
}
```

## Tiling an Effect

```
// tile_area_effect
/*
   This script is used as an Area 'On Client Enter' script. It will
   tile the object with tag PLACEABLE_FX_TAG in a hex pattern with
   spacing PLACEABLE_FX_SIZE. The suggested placeable is a modified
   copy of 'Ipoint' (under MISC PROP) with the Appearance special
   effect set to a suitable effect file. Some examples:

   Special Effect       Size   Description
   ------------------  ------  ----------------------
   fx_rain_light       22.0f   Light sporadic rain
   fx_rain_light       11.0f   Light continuous rain
   fx_rain_heavy       22.0f   Heavy sporadic rain
   fx_rain_heavy       11.0f   Torrential rain
   fx_area_fog_*       17.0f   Dynamic colored fog
   fx_blazing_fire_lg  17.0f   Plane of fire[1]
   fx_dark_vision      39.0f   --Demo-- (full coverage = 33.8f)
   fx_frost_fog        22.0f   Intermittent snow
   fx_frost_fog        11.0f   Steady snow[1]
   fx_rockslide_after   7.0f   Plane of smoke[1]
   fx_thaymount_dust   17.0f   Wispy windblown dust

   [1] Seemed sluggish on a 32 x 32 outdoor area.
*/
// RJH 11sep10

#include "ginc_math"

const string PLACEABLE_FX_TAG = "fx_point";
const float PLACEABLE_FX_SIZE = 22.0f; // 87% of effect diameter
const int RANDOM_ORIENTATION = TRUE;
const string AREA_HAS_FX_VAR = "bAreaHasFX";

void CreateEnvHexGrid( object oArea, string sTemplate, float fHexSize, int bRandOrient =
FALSE )
{
   // Get the dimensions of the area where the effects will appear
   int nHeight = GetAreaSize( AREA_HEIGHT, oArea ) - 8;
   int nWidth = GetAreaSize( AREA_WIDTH, oArea ) - 8;

   // Compute the number of loops required
   float fVertGap = fHexSize * 0.866; // * sqrt(3)/2
   int jLoops = FloatToInt( 10.0f * ( IntToFloat( nHeight ) + 0.5f ) / fVertGap ) + 1;
```

```
   int iLoops = FloatToInt( 10.0f * ( IntToFloat( nWidth ) + 0.5f ) / fHexSize ) + 1;

   // Cycle over the loops
   int i, j;
   for ( j = 0; j < jLoops; j++ ) {
      // Offset an odd row by half a diameter
      float fDelta = ( j % 2 ) ? (fHexSize / 2.0f) : 0.0f;

      // Create a row
      for ( i = 0; i < iLoops; i++ ) {
         // Compute the coordinates
         float x = ( fHexSize * IntToFloat( i ) ) + 40.0f + fDelta;
         float y = ( fVertGap * IntToFloat( j ) ) + 40.0f;
         float fAngle = 0.0f;

         if ( bRandOrient ) {
            // Generate a random 1d360 degree orientation
            fAngle = IntToFloat( Random( 360 ) );
         }

         // Generate a hex grid location
         vector vPosition = Vector( x, y, 0.0f );
         location locPoint = Location( oArea, vPosition, fAngle );

         // Create the effect placeable
         object newObject = CreateObject(
               OBJECT_TYPE_PLACEABLE,
               sTemplate, locPoint );
      }
   }
}

int StartingConditional()
{
   object oFirstPC = GetFirstEnteringPC();
   string sTemplate = PLACEABLE_FX_TAG;
   float fDiameter = PLACEABLE_FX_SIZE;
   string sFlag = AREA_HAS_FX_VAR;

   // Get the current area
   object oArea = GetArea( oFirstPC );
   if ( !GetIsObjectValid( oArea ) ) {
      return FALSE;
   }

   // Check if the flag variable is already set
```

```
    int bFlag = GetLocalInt( oArea, sFlag );
    if ( bFlag ) {
        return FALSE;
    }

    // Create the environment object grid
    CreateEnvHexGrid( oArea, sTemplate, fDiameter, RANDOM_ORIENTATION );

    // Flag this area as completed
    SetLocalInt( oArea, sFlag, 1 );

    return FALSE;
}
```

## Toggle Streetlights

```
// inc_streetlight
/*
   The following routines provide functionality for managing streetlight
   activation and deactivation based upon the time of the day. To use them,
   call InitStreetLights() from an area's 'On Client Enter Script', then call
   UpdateLights() from the same area's 'On Heartbeat Script'.
*/
// RJH 03jul11

const string LI_STATE = "light_state";
const string LI_LIGHT_COUNT = "light_count";
const string LI_LIGHT_OBJ = "light_obj_";
const string LI_LIGHT_STATE = "state";

// Prototypes
void InitStreetLights( string sLightTag, int bState );
void SetStreetLights( int bLightState, int nMaxDelay );
void UpdateLights( int nRandomDelay );

/* Initialize the streetlight state. This checks for a local variable on
   the area with the name LI_LIGHT_COUNT. If this is zero, then search for
   light objects that have the tag sLightTag. Store these as object variables
   on the area and store the total in the LI_LIGHT_COUNT variable. Finally,
   call SetStreetLights() to set the initial light state.
*/
void InitStreetLights( string sLightTag, int bState )
{
   // Retrieve the total count of lights
   object oArea = GetArea( OBJECT_SELF );
   int nCount = GetLocalInt( oArea, LI_LIGHT_COUNT );

   if ( nCount == 0 ) {
      string sObjectName;

      // Store light objects on the area
      object oLight = GetObjectByTag( sLightTag, nCount++ );
      while ( GetIsObjectValid( oLight ) ) {
         // Store light object on the area
         sObjectName = LI_LIGHT_OBJ + IntToString( nCount );
         SetLocalObject( oArea, sObjectName, oLight );

         // Set light to a known state
         SetLightActive( oLight, bState );
```

```
            SetLocalInt( oLight, LI_LIGHT_STATE, bState );

            // Retrieve the next light
            oLight = GetObjectByTag( sLightTag, nCount++ );
         }

         // Store the total
         SetLocalInt( oArea, LI_LIGHT_COUNT, nCount );
      }

      // Set the lights
      SetLocalInt( oArea, LI_STATE, bState );
      SetStreetLights( bState, 0 );
}

/*  Set the status of the streetlights. Once a street light is found that
    is not at the required state, it is changed to the required state and
    a false is returned. If the nMaxDelay is a non-zero value, the light
    state is changed after a random delay of up to nMaxDelay seconds.
*/
void SetStreetLights( int bLightState, int nMaxDelay )
{
    object oArea = GetArea( OBJECT_SELF );
    int nCount = GetLocalInt( oArea, LI_LIGHT_COUNT );
    int nTenths = 10 * nMaxDelay; // Tenths of seconds
    int n;

    for ( n = 0; n < nCount; n++ ) {
        // Retrieve the light object from the area
        string sObjectName = LI_LIGHT_OBJ + IntToString( n );
        object oLight = GetLocalObject( oArea, sObjectName );

        // Check if valid
        if ( GetIsObjectValid( oLight ) ) {
            int bCurrState = GetLocalInt( oLight, LI_LIGHT_STATE );
            if ( bCurrState == bLightState ) {
                // Light state is already set
                continue;
            }

            // Set the state of this light
            if ( nMaxDelay > 0 ) {
                // After a random delay, apply the state
                float fDelay = 0.1f * IntToFloat( Random( nTenths ) );
                DelayCommand( fDelay, SetLightActive(
                    oLight, bLightState ) );
```

```
         } else {
            // Immediately apply the state
            SetLightActive( oLight, bLightState );
         }
         SetLocalInt( oLight, LI_LIGHT_STATE, bLightState );
      }
   }
}

/* This function is called from an area heartbeat script. At 6am, the
   street lights are turned on. At 4pm, the street lights are turned off.
*/
void UpdateLights( int nRandomDelay )
{
   object oArea = GetArea( OBJECT_SELF );
   int nHour = GetTimeHour();

   // Retrieve the current light state
   if ( nHour == 6 ) {
      int bLightsOn = GetLocalInt( oArea, LI_STATE );
      if ( bLightsOn == TRUE ) {
         // Turn off the lights
         SetStreetLights( FALSE, nRandomDelay );
         SetLocalInt( oArea, LI_STATE, FALSE );
      }
   } else if ( nHour == 16 ) {
      int bLightsOn = GetLocalInt( oArea, LI_STATE );
      if ( bLightsOn == FALSE ) {
         // Turn on the lights
         SetStreetLights( TRUE, nRandomDelay );
         SetLocalInt( oArea, LI_STATE, TRUE );
      }
   }
}
```

## Turn to Face

```
// ww_turn_to_face
/*
   This is the On Enter script for a Trigger blueprint that causes an NPC
   to turn and face the entering creature. If the 'NPC_Tag' is not set to
   the tag of a valid creature, it exits. If the 'OnlyFacePC' variable is
   set to 1, it ignores non-PCs who enter the trigger. If 'BlurtOnce' is
   non-null, the NPC will speak the string the first time the creature
   enters the trigger. If 'AnimationID' is not zero, the creature will
   perform the animation with this ID when it speaks the string.
*/
// RJH 31may12

const string NPCtag = "NPC_Tag";
const string OnlyFacePC = "OnlyFacePC";
const string BlurtOnce = "BlurtOnce";
const string AnimationID = "AnimationID";
const string BlurtCheck = "BlurtCheck";
const string VoiceChat = "VoiceChat";

void main()
{
    object oTarget = GetEnteringObject();
   if ( ! GetIsObjectValid( oTarget ) )
      return; // Invalid object

   // Get the NPC tag
   string sNPC = GetLocalString( OBJECT_SELF, NPCtag );
   if ( GetStringLength( sNPC ) == 0 )
      return; // Variable was not set

   // Find the NPC
   object oNPC = GetObjectByTag( sNPC );
   if ( ! GetIsObjectValid( oNPC ) )
      return; // Invalid object

   // Get the Face PC setting
   int bFacePC = GetLocalInt( OBJECT_SELF, OnlyFacePC );
   if ( bFacePC && ! GetIsPC( oTarget ) )
      return; // Entering object is not a PC

   // Check for a clear line of sight
   if ( !LineOfSightObject( oTarget, oNPC ) )
      return;
```

## Turn to Face

```
// Get entering creature's stealth mode
int nStealthMode = GetStealthMode( oTarget );
if ( nStealthMode == STEALTH_MODE_ACTIVATED )
   return; // Don't break the illusion of stealthiness

// Get the tag of the entering object
string sEnterTag = GetTag( oTarget );
if ( StringCompare( sNPC, sEnterTag ) == 0 )
   return; // Ignore self

// Check for a blurt string
int bDoBlurt = FALSE;
string sBlurt = GetLocalString( OBJECT_SELF, BlurtOnce );
if ( GetStringLength( sBlurt ) > 0 ) {
   string SBlurtCheck = BlurtOnce + sNPC;
   int bHasBlurted = GetLocalInt( oTarget, SBlurtCheck );
   if ( bHasBlurted == 0 ) {
      bDoBlurt = TRUE;

      // Clear actions so blurt string is not delayed
      AssignCommand( oNPC, ClearAllActions() );
   }
   SetLocalInt( oTarget, SBlurtCheck, 1 );
}

// Turn to face target
vector vTarget = GetPosition( oTarget );
AssignCommand( oNPC, SetFacingPoint( vTarget ) );

float fDelay = 0.1f;
if ( bDoBlurt ) {
   // Speak the blurt string
   DelayCommand( fDelay, AssignCommand( oNPC, ActionSpeakString( sBlurt ) ) );
   fDelay += 0.1f;

   int nAnimation = GetLocalInt( OBJECT_SELF, AnimationID );
   if ( nAnimation != 0 ) {
      // Perform the animation
      DelayCommand( fDelay, AssignCommand( oNPC, ActionPlayAnimation( nAnimation ) ) );
      fDelay += 0.1f;
   }
}

/* Check for a voice chat
```

```
      Bad idea  = 47 | Bored   = 39 | Can do     = 29 | Can't do  = 30
      Cheer     = 44 | Cuss    = 43 | Encumbered = 32 | Flee      =  7
      Follow me = 22 | Goodbye = 40 | Good idea  = 46 | Group     = 24
      Guard me  =  9 | Heal me =  4 | Hello      = 34 | Help      =  5
      Hide      = 28 | Hold    = 10 | Laugh      = 42 | Look here = 23
      Move over = 25 | No      = 36 | Rest       = 38 | Search    = 27
      Selected  = 33 | Stop    = 37 | Talk to me = 45 | Task done = 31
       Taunt     =  8 | Thanks  = 41 | Threaten   = 48 | Yes       = 35
      Pain   = 14-16
   */
   int nVoiceChat = GetLocalInt( OBJECT_SELF, VoiceChat );
   if ( nVoiceChat != 0 ) {
      string sVoiceCheck = VoiceChat + sNPC;
      int bHasVoiced = GetLocalInt( oTarget, sVoiceCheck );
      if ( bHasVoiced == 0 ) {
         DelayCommand( fDelay, PlayVoiceChat( nVoiceChat, oNPC ) );
         fDelay += 0.1f;
         SetLocalInt( oTarget, sVoiceCheck, 1 );
      }
   }
}
```

## Undergrowth Trigger

```
// underbrush_enter
/*
   This is the On Enter script for a Trigger blueprint that
   simulates movement through an area of underbrush. It is
   based upon the DMG rules for Forest Terrain, and has the
   following effects on a creature entering the region:

     * 20% concealment bonus
     * -2 Move Silently
     * -2 Tumble
     * 50% speed reduction effect*

   If a creature is incorporeal or has the Woodland Stride
   feat, then the speed reduction effect is not applied.
   This script does not cover the case where the creature
   becomes incorporeal after entry into the brush.
*/
// RJH 04jul11

#include "x2_inc_switches"

const int UNDERBRUSH_ID = 13400; // Spell ID of the effect
const int SLOW_RATE = 50; // Half movement rate
const int CONCEALMENT = 20; // 20% concealment
const int DC_CHANGE = 2; // +2 to DC for Tumble and Move Silent

void main()
{
    object oTarget = GetEnteringObject();

   // Create common effects
   effect eTumble = EffectSkillDecrease( SKILL_TUMBLE, DC_CHANGE );
   effect eMoveSilent = EffectSkillDecrease( SKILL_MOVE_SILENTLY, DC_CHANGE );
   effect eConceal = EffectConcealment( CONCEALMENT );
   effect eLink = EffectLinkEffects( eTumble, eMoveSilent );
   eLink = EffectLinkEffects( eLink, eConceal );

   // Check if creature is incorporeal or has the Woodland Stride feat
   if( !GetCreatureFlag( oTarget, CREATURE_VAR_IS_INCORPOREAL ) &&
      !GetHasFeat( FEAT_WOODLAND_STRIDE, oTarget ) ) {
      effect eSlow = EffectMovementSpeedDecrease( SLOW_RATE );
      eLink = EffectLinkEffects( eLink, eSlow );
   }
```

```
   // Apply the effect as supernatural to avoid negation
   eLink = SupernaturalEffect( eLink );
   eLink = SetEffectSpellId( eLink, UNDERBRUSH_ID );
   ApplyEffectToObject( DURATION_TYPE_PERMANENT, eLink, oTarget );
}
```

```
// underbrush_exit
/*
   This is the On Exit script for a Trigger that simulates movement through an area of
   underbrush. It removes the effect applied by the On Enter script.
*/
// RJH 04jul11

const int UNDERBRUSH_ID = 13400; // Spell ID of the effect

void main()
{
   object oTarget = GetExitingObject();

   // Search for the underbrish effect
   effect eEffect = GetFirstEffect( oTarget );
   while ( GetIsEffectValid( eEffect ) ) {
      int nID = GetEffectSpellId( eEffect );
      if ( nID == UNDERBRUSH_ID ) {
         // Remove the effect
         RemoveEffect( oTarget, eEffect );
         return;
      }
      eEffect = GetNextEffect( oTarget );
   }
}
```

## Using Best Appraise Skill

```
// ga_open_store_best_appraise
/*
   This action script can be used to open a store using the highest
   appraise skill in the party. (The ga_open_store script only uses
   the appraise skill of the PC.) After saving this script in your
   module, you can choose it as an action script in a conversation.

   sTag      Tag of a Store object.
   nMarkUp   Percentage increase of items sold by store.
   nMarkDown Percentage decrease of items bought by store.
*/
#include "ginc_param_const"
#include "ginc_item"

void main( string sTag, int nMarkUp, int nMarkDown )
{
   object oPC = GetPCSpeaker();
   object oBest = oPC;
   int nBestRank = 0;

   object oFM = GetFirstFactionMember( oPC );
   while ( GetIsObjectValid( oFM ) ) {
      int nRank = GetSkillRank( SKILL_APPRAISE,  oFM );
      if ( nRank > nBestRank ) {
         // Update the current best
         oBest = oFM;
         nBestRank = nRank;
      }
      object oFM = GetNextFactionMember(oPC);
   }
   N2_AppraiseOpenStore( GetTarget(sTag), oBest, nMarkUp, nMarkDown);
}
```