Step-by-Step Guide to the NWN2 Companion System

v 1.1

By Andy and Christopher, aka Andysks and ColorsFade.

Table of Contents

About This Guide

My name is Andy, and even though I document this, all work belongs to ColorsFade. I only wanted a companion system like in the OC, but being not so good in scripting, he helped me since he uses the same system. I chose to document this because knowledge is still fresh on the matter, and maybe other people will have the same wishes as me :).

-Andy

There is a lot of legacy code and unused scripts in the NWN2 engine. It took me hours to figure out which functions to call, when to call them, and which functions to ignore. My hope is, with this guide, others will save precious time.

-Christopher

The Companion Management System that is native to the OC is primarily contained in a script library called **ginc_companion**. There are several tasks that must be accomplished in order to correctly manage your companions through this system:

- Initializing Companions
- Setting Spawn Waypoints
- Setting Hangout Waypoints (where companions go when you dismiss them)
- Spawning Companions
- Dismissing Companions from the party
- Moving Hangouts
- Preserving Companion state while loading new Modules
- Using the Roster GUI

You will need to write new scripts to perform custom logic for your own companions. It is recommended that you have a basic working knowledge of scripting before attempting to implement this system.

Getting Started

I recommend creating your own script library for all the functions you're going to want to write to manage your companions. These are simple functions and I will provide examples in this guide. I recommend naming your script something like **d_ginc_party**. The "ginc" lets us know that it's a global include file, and the "d_" prefix ensures it won't overwrite other "ginc" scripts.

Spawn Waypoints

The first thing you need to do is lay down spawn waypoints for your companions. Each companion should have *one* and *only one* spawn waypoint in the entire campaign. Spawn waypoints are named "spawn_companiontag". The "spawn_" portion must be lowercase or these waypoints will not work.

Example: If you have a companion named Adoward, and that companion's tag is "adoward", then their spawn waypoint would be named "spawn_adoward".

Initializing Companions

Your companions must be **Initialized** the first time your module loads. This is a one-time event and needs to happen in the k_mod_load file. The easiest way to do this is to copy the k_mod_load file and rename it, because we're going to change it, and you never want to change the default game files.

I renamed my copy to **dk_mod_load**. That new dk_mod_load file now needs to be associated with all of your modules in the OnModuleLoad event.

A little more than halfway down the script there is a section where code runs one time. The beginning looks like this:

```
if ( GetGlobalInt(VAR_CAMPAIGN_SETUP_FLAG) == FALSE )
// Code goes here
```

What we want to do here is call a function to Initialize all of our companions. This is a simple, two-step process:

1. Write the function in our d_ginc_party library
2. Call the function in our dk_mod_load script

Remember to add a reference to ginc_companion in your d_ginc_party library:

```
#include "ginc_companion"
```

Now add the function to d_ginc_party. We're going to use the method InitializeCompanion(string sRosterName, string sTag, string sResRef) from ginc_companion on each companion in our module/campaign:

```
void InitializeCompanions()
{
	InitializeCompanion("wurdy","wurdy","wurdy");
	InitializeCompanion("kayne","kayne","kayne");
	InitializeCompanion("celeste","celeste","celeste");
	InitializeCompanion("doxie","doxie","doxie");
	// More lines for however many companions you have
}
```

The next thing we need to do is call this function, InitializeCompanions(), in the dk_mod_load script, at the proper point. Make sure to include d_ginc_party library in the dk_mod_load script:

```
// Only setup once for entire campaign
if ( GetGlobalInt(VAR_CAMPAIGN_SETUP_FLAG) == FALSE )
{
	SetGlobalInt(VAR_CAMPAIGN_SETUP_FLAG, TRUE );

	// Setup all Companions
	InitializeCompanions();
```

Calling our function here ensures that it only gets called once when the campaign/module first loads. At this point, our companions are initialized and we're ready to spawn them at their spawn waypoints.

Spawning Companions

According to documentation in the ginc_companion library, companions should never be placed in an area. They should only be spawned. More specifically, they should be spawned with the method:

SpawnNonPartyRosterMemberAtHangout("companiontag");

This is the ONLY method that should ever be called to spawn a companion.

Note: You may be thinking right now, "But I haven't placed any hangouts yet. How will the companion spawn?" That is part of what the InitializeCompanion() method does. It sets their current hangout waypoint to be the waypoint labeled "spawn_companiontag". Later on, the hangout will be changed to something other than the spawn waypoint.

Spawning companions properly means using an OnClientEnter script for the area where the companion should first be encountered. I like to name my scripts "client_enter_areatag". So, for example, I want to spawn my Cleric companion, Celeste, in the forest_ruins area. I will create a script called client_enter_forest_ruins.

Inside that script I will write the following code:

```
int bSpawned = GetLocalInt(OBJECT_SELF, "SPAWNED");
if(!bSpawned)
{
        // Spawn Celeste
        SpawnNonPartyRosterMemberAtHangout("celeste");
        SetLocalInt(OBJECT_SELF, “SPAWNED”, TRUE);
}
```

What this code does is check a variable on the area to determine if the spawn has happened (this prevents our companion from being spawned more than once). If the area has not been spawned, then we spawn our companion at their hangout. Since our companion has been initialized, that means they will spawn at a waypoint named "spawn_celeste".

Adding & Removing Companions

Now that our companion is spawned, we need to provide a way to add or remove the companion from our party. When the PC initially meets the companion, this should be done through a dialog. Later on, we'll be able to change our party via the Roster GUI. For now, we'll discuss how to add a companion to our party.

Adding a Companion

There are several function calls that need to be made in order to add a companion to your party. It is my suggestion that you wrap these calls up into a single method and put that method in your d_ginc_party library. Here's my function, called AddCompanionToParty(). This is the ONLY function I use to add companions to the party.

```
////////////////////////////////////////////////////////////////////////////////
// AddCompanionToParty()
// Our one and only one method for adding a companion to the party.
////////////////////////////////////////////////////////////////////////////////
void AddCompanionToParty(string sCompanionTag)
{
        ////////////////////////////////////////////////////////
        // Make the companion visible on the Roster GUI
        ////////////////////////////////////////////////////////
        SetIsRosterMemberCampaignNPC(sCompanionTag, 0);

        ////////////////////////////////////////////////////////
        // Make the companion selectable on the Roster GUI
        // FROM: ga_roster_selectable
        ////////////////////////////////////////////////////////
        SetIsRosterMemberSelectable(sCompanionTag, 1);

        ////////////////////////////////////////////////////////
        // Add the companion to the party
        //FROM: ga_roster_party_add
        ////////////////////////////////////////////////////////
        object oPC = GetFirstPC();
        AddRosterMemberToParty(sCompanionTag, oPC);

        ////////////////////////////////////////////////////////
        // Just in case we forgot to set this, when we add a
        // companion to our party, we've met them.
        ////////////////////////////////////////////////////////
        SetLocalInt(oPC, "met_" + sCompanionTag, TRUE);

        ////////////////////////////////////////////////////////
        // Set the companion's XP equal to that of the PC
        ////////////////////////////////////////////////////////
        object oCompanion = GetObjectByTag(sCompanionTag);
        int nXP = GetPCAverageXP();
        SetXP(oCompanion, nXP);
        ForceRest(oCompanion);
}
```

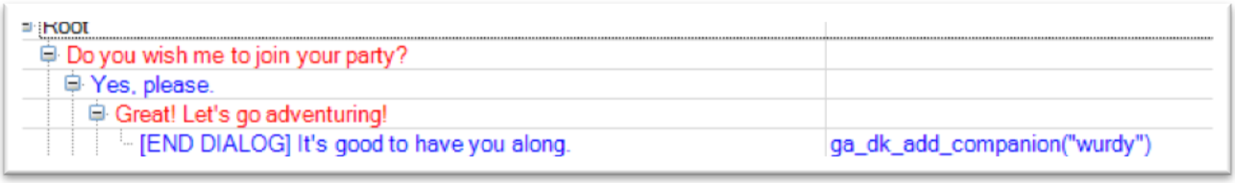
In order to use AddCompanionToParty() in a dialog node, we need to call it from a script. I named my script ga_dk_add_companion(). This is the entire contents of that script:

```
////////////////////////////////////////////////////////////////////
// Adds a companion to the party
////////////////////////////////////////////////////////////////////

#include "d_party"

void main(string sCompanionTag)
{
	AddCompanionToParty(sCompanionTag);
}
```

This allows us to call this function from a conversation node as an action:



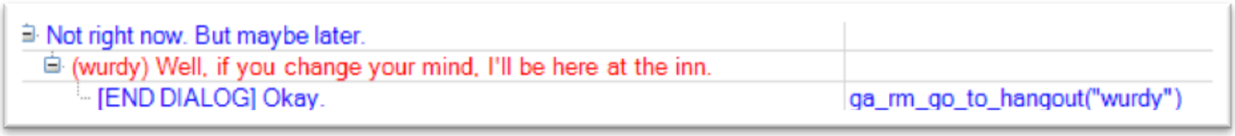
This will immediately add the companion to your party and grant them XP to match the player character. You may notice several function calls in the method listed above. I will detail those methods and their purpose later in this document.

Removing a Companion

There should only be one way to remove them from your party. This eliminates bugs and glitches that can occur if you attempt to use other methods. Via a conversation, this one method is called **ga_rm_go_to_hangout**.

It is important that this be the ONLY method you use to remove party members! This method will change the companion's hangout waypoint from *spawn_companionname* to *hangout_companionname*. This is crucial, because your hangout spot for the companion is probably not the same as their spawn location.

Used in a conversation, it looks like this:



Note: If you need to remove a companion from your party without using dialog, I recommend copying the contents of the ga_rm_go_to_hangout script into your d_ginc_party library and wrapping it in a function call. This will allow you to use it in a non-dialog script. I will not go into details about how to accomplish that task, however, as it is a basic scripting technique and beyond the scope of this tutorial.

Note: There is a caveat to removing a companion, which I discuss on **p21**.

Managing Hangouts

When you dismiss a companion from your party with ga_rm_go_to_hangout, one of two things will happen:

1) If you are in the same physical area as the **default** hangout waypoint, the companion will leave your party and walk to the hangout. They will remain at the hangout until you usher them back into your party or you unload the module. Don't worry! There is a safe way to unload the module and ensure all non-party companions are saved properly. We'll get to it later.
2) If you are NOT in the same physical area as the default hangout waypoint (most likely the case) then the companion will leave your party and de-spawn. The current state of the companion will be saved out of the game.

Which means your companion is *not really* at their hangout. We have to spawn them at their hangout when we get there.

So, for example, let's say we met our trusty cleric, Celeste, in the FOREST_RUINS area. That was where her spawn_celeste waypoint was located. However, her default hangout is in the INN, back in town, in another module. How do we get her there?

We have to re-spawn her.

Once again we have to make use of an OnClientEnter script and a couple of custom functions.

Put Companions in Place

The script functions I am copying here are essentially the same ones that the OC uses in the Sunken Flagon Inn. These methods should be added to your d_ginc_party library:

```
//////////////////////////////////////////////////////////////////////
// Puts non-party companions at their hangout spots.
// This should only be used in an area where we have set the hangouts,
// like an inn or stronghold. To be called in the OnClientEnter
// script slot.
//////////////////////////////////////////////////////////////////////
void PutCompanionsInPlace()
{
	PutCompanionInPlace("wurdy");
	PutCompanionInPlace("kayne");
	PutCompanionInPlace("celeste");
	PutCompanionInPlace("doxie");
	// More for however many companions you have
}

int GetHasMetCompanion(string sCompanionTag)
{
	object oPC = GetFirstPC();
	int bHasMet = GetLocalInt(oPC, "met_" + sCompanionTag);

	return bHasMet;
}
```

```
/////////////////////////////////////////////////////////////////////
// Puts a non-party companion at their hangout.
// Should only be called by PutCompanionsInPlace()
/////////////////////////////////////////////////////////////////////
void PutCompanionInPlace(string sCompanionTag)
{
	object oPC = GetFirstPC();

	int bMetCompanion = GetHasMetCompanion(sCompanionTag);

	// If we haven't met the companion before, exit.
	if(!bMetCompanion)
	{
		return;
	}

	// If the companion isn't already in our party, then place them at
	// their hangout waypoint. There should only be one of these per module!
	object oCompanion = GetObjectByTag(sCompanionTag);
	if(!GetIsObjectInParty(oCompanion))
	{
		SpawnNonPartyRosterMemberAtHangout(sCompanionTag);
	}
}
```

Now that you have these methods in d_ginc_party, you can use them in an OnClientEnter script. Suppose we have a client_enter_inn script. All we need to do is call this:

```
#include "d_party"

void main()
{
	/////////////////////////////////////////////////////////////
	// Companions
	// This is the first "hangout" in the game, so spawn any
	// companions we've encountered, that are not already in the
	// party, at their hangout waypoints.
	/////////////////////////////////////////////////////////////
	PutCompanionsInPlace();
```

Our function, PutCompanionInPlace, takes care of the logic for us. It checks to see if we've met the companion or not (we don't want to spawn any companions we haven't met yet). It also checks to see if the companion is in our party, and if they are not, it spawns them at their waypoint.

Handling Multiple Hangouts

During the course of your campaign, you may want to provide multiple hangouts for the companions. The OC does this, since you can find all your non-party companions at either the Sunken Flagon, or Crossroad Keep.

In order to accomplish this, you need to change the hangouts for all the companions when you enter a hangout area.

The simplest way to do this is via the OnClientEnter script for the hangout area, and by using a method provided by the ginc_companion library:

```
SetHangOutSpot(string sRMRosterName, string sHangOutWPTag)
```

With this method, you can easily change the hangout waypoints for each companion.

As an example, let's say I have two hangouts for my companions: the inn, and the stronghold that the player has acquired.

My hangout waypoints at the inn might be labeled as such:

- hangout_inn_wurdy
- hangout_inn_celeste

My hangout waypoints at the stronghold might be labeled as:

- hangout_stronghold_wurdy
- hangout_stronghold_celeste

In the client_enter_inn script, I would then need to add the following:

```
////////////////////////////////////////////////////////////////
// Companions
// This is the first "hangout" in the game, so spawn any
// companions we've encountered, that are not already in the
// party, at their hangout waypoints.
////////////////////////////////////////////////////////////////
SetHangOutSpot("wurdy","hangout_inn_wurdy");
SetHangOutSpot("celeste", "hangout_inn_celeste");
SetHangOutSpot("kayne", "hangout_inn_kayne");
SetHangOutSpot("doxie", "hangout_inn_doxie");

PutCompanionsInPlace();
```

You can see here, we've made sure to adjust the hangout spots before calling PutCompanionsInPlace().

As long as you explicitly set the hangout waypoint with SetHangOutSpot() prior to calling PutCompanionsInPlace(), your non-party companions will always spawn in any hangout you enter. This allows you to have multiple hangouts.

Traveling Across Modules

There is one caveat in all of this: cross-module travel.

If you have built modules before then you know that in order to load an area from another module you need to call LoadModule(). However, if you are using the ginc_companion system, you need to change that call.

Whenever you want to load a new module, you need to call **SaveRosterLoadModule**() instead.

What this method does is de-spawn ALL non-party roster members before calling LoadModule itself. This is essential, so that your companions get saved out of the game properly, and don’t get lost or have their state messed up.

Roster GUI

Now that we know how to add & remove companions to our party via the dialog operations, let's focus our attention on the Roster GUI:

The Roster GUI allows us a fast and graphical way to manage our party. However, there are some caveats to its use.

The first thing to understand is that the Roster GUI will de-spawn all non-party companions, and save them out of the game, as soon as the player chooses the "Accept Party" button. This is why the Roster GUI must be used at the moment when the player is leaving an area.

If you recall, it is used in the OC on the door of the Sunken Flagon, as the player is exiting the building, and on the World Map Transition outside Crossroad Keep.

Using the Roster GUI on a door, or other object, like a World Map Transition, means you must replace the default scripts on the door/map with custom scripts, so that you can present the Roster GUI to the user instead of performing the default action (such as a door transition).

The default action of the object (door/map transition) is then performed via a callback script when the Roster GUI closes. Here is an example:

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

	// The callback script for ShowPartySelect needs this data.
	SetLocalString(oPC, "DOOR_TAG", GetTag(OBJECT_SELF));

	// Kayne is the 2nd companion you meet, and his meeting is unavoidable.
	if(GetHasMetCompanion("kayne"))
	{
		ShowPartySelect(oPC, TRUE, "d_roster_door_exit", TRUE );
	}
	else
	{
		ClearAllActions(TRUE);
		AssignCommand(OBJECT_SELF, ActionCloseDoor(OBJECT_SELF));
		DelayCommand(0.1f, JumpParty());
	}
}
```

This script is used on a door. The first thing it does is get the TAG of the door and assign it to the DOOR_TAG variable on the PC. This is so that the callback script, d_roster_door_exit, will know which door transition to perform once the Roster GUI is closed.

The second thing this script does is check to see if a particular companion (Kayne) has been met yet. This is because it doesn't make a lot of sense to show the Roster GUI until the player has met more than one companion. If Kayne has been met, then the player has encountered more than one companion, and we should show the Roster GUI. This is achieved with the following line of code:

ShowPartySelect(oPC, TRUE, "d_roster_door_exit", TRUE);

This tells the game to show the Roster GUI, and to perform the callback script, "d_roster_door_exit", when the Roster GUI Closes.

In the d_roster_door_exit script we will check for the DOOR_TAG variable on the PC and transition to the appropriate waypoint.

The remainder of the script is what happens if we're NOT supposed to show the Roster GUI. In that case, we just want to perform the transition to the waypoint. I am not going to show the JumpParty() function because it is simply a party jump to a specific waypoint.

Roster GUI Functions

There are two additional functions that are important for the Roster GUI. You may have noticed these function names in our earlier method, AddCompanionToParty:

ISRosterMemberCampaign

IsRostermemberSelectable

SetIsRosterMemberCampaignNPC

This function allows a companion to be visible or hidden on the Roster GUI. When we call InitializeCompanion(), this flag gets set to TRUE, which means the companion will NOT be visible on the Roster GUI. Since all companions get initialized at the beginning of the game, that means none of them are initially visible on the Roster GUI. This is as it should be; we don't want the player having access to companions via the Roster GUI before the player has actually met them in the story.

However, once we meet the companion, we need to toggle this flag to FALSE, so that the companion becomes available on the Roster GUI. We automatically make this call when we do the AddCompanionToParty call, but what if we decided to NOT take the companion into our party when we initially meet them? (Grobnar, anyone?) Then we need to toggle the flag there as well. Fortunately, there is a simple script we can call during a conversation, called **ga_roster_campaignnpc**, to do the job for us.

This should be called with a FALSE value to ensure the companion shows up .

ga_roster_campaignnpc("grobnar", FALSE);

IsRosterMemberSelectable

This is another function that affects companions in the Roster GUI. This function determines if a companion is selectable in the Roster GUI.

By setting this value to TRUE, the companion may be selected in the Roster GUI and then added/removed from the party. If this value is set to FALSE, the companion's name will show up in the Roster GUI (if SetIsRosterMemberCampaignNPC has been set to FALSE), but they will be greyed out and unavailable for selection. This is what happens with Shandra in the OC. She shows up in the Roster GUI, but you cannot add or remove her from the party via the Roster GUI.

For conversations, the script **ga_roster_selectable** allows us to toggle this value.

Removing a Companion: Part 2

So, remember on page 10 when I showed you the ga_rm_go_to_hangout script, and said it is the one-and-only-one way you should remove companions?

Well, there is a caveat to that: the ga_rm_go_to_hangout script does NOT set the companion's IsCampaignNPC flag to FALSE, and it does not set their IsRosterMemberSelectable to TRUE. Why does this matter? Let me explain a specific scenario:

You meet a companion on the road for the first time and a dialog initiates. At the end of the dialog you elect NOT to take the companion into your party. Your dialog calls ga_rm_go_to_hangout and the companion leaves. Since they were never added to your party, their IsCampaignNPC flag never gets turned to FALSE and their IsRosterMemberSelectable flag never gets set to TRUE. This companion will not show up in the Roster GUI. You will still meet them at their hangout, and if you have the proper dialog support you can add them to your party through dialog, however, they will not appear in the Roster GUI.

There are two solutions to this problem.

The first solution involves the use of two additional ga_ scripts in your conversation. You need to call the following two scripts prior to calling ga_rm_go_to_hangout:

- ga_roster_campaignnpc("companiontag", FALSE)
- ga_roster_selectable("companiontag", TRUE)

This is a totally viable solution and will ensure that your companion is visible and selectable in the Roster GUI.

However, I prefer to keep things simple when using scripts in conversations, and the simplest thing to do is wrap all three scripts into one. I dub this script ga_go_to_hangout:

ga _go _to _hangout

```
/*********************************************************************
REPLACEMENT script for ga_rm_go_to_hangout
This allows us to use ONE ga_ script call in a conversation, instead
of three.

1)	Sends a companion to their hangout.
2)	Marks the companion as NOT being a campaign NPC so they show up in
	the Roster GUI
3)	Marks them as Roster Selectable so they can be manipulated in
	the Roster GUI
*********************************************************************/

#include "ginc_companion"

int IsHangoutSpawnOrBlank(string sRosterName);
void SetStandardHangout(string sRosterName);

void main(string sRosterName)
{
	SetIsRosterMemberCampaignNPC(sRosterName, FALSE);
	SetIsRosterMemberSelectable(sRosterName, TRUE);

	if (sRosterName == "")
	{
		object oSelf = OBJECT_SELF;
		string sRosterName = GetRosterNameFromObject(oSelf);
		if (sRosterName == "")
		{
			PrettyError("ga_rm_go_to_hangout failed - couldn't get Roster Name
for " + GetName(oSelf));
			return;
		}
	}

	// change to default hangout spot if using the standard spawn or if not set
	if (IsHangoutSpawnOrBlank(sRosterName))
		SetStandardHangout(sRosterName);

	GoToHangOutSpot(sRosterName);
}


int IsHangoutSpawnOrBlank(string sRosterName)
{
	int nRet = FALSE;

	string sHangoutSpot = GetHangOutSpot(sRosterName);
	string sStandardSpawnTag = "spawn_" + sRosterName;
```

```
	if ((sHangoutSpot == "") || (sHangoutSpot == sStandardSpawnTag))
		nRet = TRUE;

	return (nRet);
}

// the standard hangout spot is "hangout_<roster name>"
void SetStandardHangout(string sRosterName)
{
	string sStandardHangoutTag = "hangout_" + sRosterName;
	SetHangOutSpot(sRosterName, sStandardHangoutTag);
}
```

This script, as you can see, calls SetIsRosterMemberCampaignNPC() and SetIsRosterMemberSelectable() at the beginning, thus bypassing the need to set those up as separate Actions on a dialog node. The rest of the script is a direct copy of the ga_rm_go_to_hangout script.

This script should be your one-and-only-one way to remove a companion (or send them away in that initial conversation).

Limiting Companions to Specific Hangouts

This question was raised on the message boards, so I'll address is here.

This tutorial has given no consideration to limiting companions in a certain hangout. It assumes you want to spawn all your "met"/discovered companions in a hangout. However, you can easily limit the companions that spawn in a hangout.

In the Area's OnClientEnter script, instead of calling PutCompanionsInPlace(), simply call the individual PutCompanionInPlace("companiontag") methods.

Example:

Suppose we have companions named: Wurdy, Doxie, Kayne, and Celeste. Kayne being a paladin and Celeste being a cleric, we also want them to spawn at a nearby temple when we arrive. We don't want any of our other companions to spawn there.

In our client_enter_temple script, we would simply put this:

```
SetHangOutSpot("kayne", "hangout_temple_kayne");
SetHangOutSpot("doxie", "hangout_temple_celeste");

PutCompanionInPlace("kayne");
PutCompanionInPlace("celeste");
```

Notice we do not use the PutCompanionsInPlace method; we don't want all the companions to show up, just these two. And notice that we first adjust their hangout spots to this area, the temple, prior to calling this method.

Using this method, you can have any number of companions show up in whichever areas you desire.

Conclusion

The ginc_companion system may seem a little bit overwhelming at first, but it is actually quite elegant, and once you get used to using it, it will make your companion management much easier.

The one thing to understand is that there is basically only ***one way*** to spawn companions, ***one way*** to add them to your group, and ***one way*** remove them from your group. Stick to those singular methods and you'll avoid issues.

Also be aware of two other things:

1. When you begin a dialog with a companion for the *very first time*, you should call ga_local_int("met_companiontag", TRUE, $PC) to set the "met_" variable. This variable automatically gets set if you ADD the companion to your party, but otherwise it does not. If you decline to accept a companion into your party when you first meet them, they will fail to show up in their hangouts if you haven't set this variable. I make sure I set this variable on the very first line of the conversation when I first meet the companion.

2. Make sure you are calling SetIsRosterMemberCampaignNPC and IsRosterMemberSelectable at the appropriate times if you plan on using the Roster GUI, and especially if you plan on "forcing" a companion into the party. Forced companions should have their SetIsRosterMemberCampaignNPC = FALSE, so they show up in the Roster GUI, but they should have their IsRosterMemberSelectable=FALSE as well, so they cannot be removed from the party.

As always, if anyone has issues or needs help, I am happy to assist. You may contact me at: cb.holmes@gmail.com.