TSLPatcher Documentation

TSLPatcher v1.2.8b1
ChangeEdit v1.0.4b8

Last changed 2006–08–09

Table of Contents

1. About TSLPatcher

TSLPatcher is a small utility designed to help in applying 3rd party Mods to the SWKotORII:**T**he **S**ith **L**ords game (though it will work with SWKotOR1 as well). Its primary intended use is as an installer-type of application to distribute with Mods to make them easier to install and make compatible with other Mods.

It does not claim to be the "Ultimate Mod Installer". While it can be useful for many types of MODs, it will not help in all situations. See the bullet list below to for some examples to see if you can gain anything from using it with your Mod.

1.1 What can TSLPatcher help with?

A few things:

■ It allows you to add new entries to the dialog.tlk file without distributing the whole 10 MB file.

■ It allows you to add new lines to 2DA files that may already exist within the override folder, allowing you to modify those files without making your mod incompatible with other mods that have modified the same file.

■ It allows you to change values in existing cells in a 2DA file located in the override folder, again ensuring compatibility with other mods that modify the same 2DA file. (As long as that mod does not modify that same cell, of course...)

■ It allows you to add new columns to existing 2DA files in override, again helping to ensure compatibility with other mods using the same files.

■ It allows you to modify the values of existing fields within GFF files located in the override folder. Aside from allowing compatibility with other Mods using the same file, it also allows you to update fields that refer to lines in a 2DA file to point to new lines the TSLPatcher has added dynamically.

■ It allows you to keep StrRefs of the new entries the TSLPatcher has added in memory, and insert those StrRefs into 2DA and GFF files as needed. For example, if you have added the name of a new force power to dialog.tlk, the TSLPatcher may memorize the StrRef the name string ended up as, and insert that value in the "name" column in spells.2da.

■ It allows you to keep cell values from 2DA files as well as the line number of newly added rows in memory, and insert those values into other 2DA files or GFF files. For example, if you add a new row to appearance.2da, the TSLPatcher may memorize the line number that row was added as, and insert that number as Appearance_type value in a Creature template (.UTC) file.

■ It allows you to insert new fields into GFF-format files (UT*, DLG, JRL etc...) present in the override folder and either assign the value directly or fetch it from a memorized *StrRef* or 2DA value. For example you could use it to insert new journal entries into a custom global.jrl file that another mod has already put in override.

■ If used in combination with the *nwnnsscomp.exe* utility (by *Edward T. Smith* and KotORified by tk102) it can replace tokens placed within script source files with the proper memorized StrRef or 2DA values before compiling those scripts and putting the resulting NCS files in the user's Override folder.

For example it could be used to make the parameter to a `ChangeObjectAppearance()` script function call point to the correct line number of a row TSLPatcher has added to *appearance.2da*. Or it could be used to make sure the parameter to `BarkString()` is the correct *StrRef* of a *dialog.tlk* entry that has just been added.

■ It allows you to put other files it does not need to modify in the necessary folders within the game folder. I.e. it could make sure that .MOD files go in the *Modules* folder, .WAVs go in the proper *StreamVoice/StreamSounds/StreamMusic* folder, other files get put in *Override* etc.

■ It allows you to modify GFF files that already exist in either the override folder or an ERF or RIM format archive file within the game folder or any of its sub-folders. You can also insert new GFF files into any such files.

■ It allows you to insert recompiled NCS files into an ERF or RIM format archive file within the game folder or any of its sub-folders, or overwrite any NCS file with the same name already existing at the destination.

1.2. What can TSLPatcher NOT help with?

Among other things:

■ It is not able to make two Mods that make changes to the same standard game script (NCS) file. Due to the complex format and vast possibility of potential changes of script files, such files will have to be merged manually. Please include the NSS source code files with your mod to allow people to do this if necessary. When creating new scripts for your mod, you should name them in a way (using a custom prefix or suffix in the resref filename) that makes you reasonable certain others would not name their scripts the same way, to avoid this problem.

■ It will currently not modify files that are present within sub-folders within the override folder. This is by design to avoid ambiguity with the possibility of several copies of the same file being present in different sub-folders (or the user mistakenly placing the installer and associated files in a subfolder within the override folder). It will only look for files to modify directly in the override folder.

■ It is currently unable to modify files inside BIF format files. Only files located in the override folder and ERF or RIM format archive files can be modified.

2. Setup Instructions

To use the TSLPatcher as a mod installer, these steps should work for most situations:

2.1. Create a new folder for your Mod, name it whatever you like.

2.2. Put the *TSLPatcher.exe* in this folder. You should rename it to something more intuitive so people understand they should run it to install the mod.

2.3. Create a folder inside your Mod folder named **exactly** *tslpatchdata*.

2.4. Copy all the files that belong to your MOD into the *tslpatchdata* folder.

2.5. Extract **UNALTERED** copies of any 2DA and standard game GFF format files you want the TSLPatcher to work with. These files will be copied to override and then modified if the file did not already exist in the user's override folder. Put these files in the *tslpatchdata* folder.

2.6. If your mod needs to use custom *dialog.tlk* entries, use *TalkEd.exe* to create a new TLK file. Add your new entries to this file and name it exactly *append.tlk*.

If you are using a non-English version of the game that has a *dialogf.tlk* file as well, create a new file with the feminine form of your strings as well and name it exactly *appendf.tlk*. (Note that *appendf.tlk* **must** have exactly the same number of entries as *append.tlk*. If a string has no specific feminine form, you **must** put the same text in both files.) Save your *append.tlk* (and, if applicable, *appendf.tlk*) in the *tslpatchdata* folder.

2.7. Use WordPad (that comes with Windows) to create an RTF format text file named **exactly** *info.rtf*. The contents of this file will be displayed in the text box in the main window when the TSLPatcher starts.

You may put whatever you like in this file, but it can be useful to put either special installation instructions, if necessary, or the Read Me file here. Please use only basic formatting and standard fonts in the info.rtf file. Embedded images will not work. Save info.rtf in your tslpatchdata folder.

2.8. If your mod needs to use the script compile functionality, copy *nwnnsscomp.exe* and *nwscript.nss* into the *tslpatchdata* folder. Also copy the **NSS** script files you intend to use with it, and all **include files** those scripts use, into the *tslpatchdata* folder.

2.9. Start *ChangeEdit.exe* to configure what the TSLPatcher should do with your files. See section 3 below for more information about *ChangeEdit*. Create a new configuration file in *ChangeEdit* and name it **exactly** *changes.ini*, and save it inside the *tslpatchdata* folder.

This file will contain all the work instructions for the TSLPatcher, telling it what to do with your files. Unless you really know what you are doing you should use *ChangeEdit* to create and modify this file, though it is a plain text file you can open and edit with Notepad as well if you wish.

2.10. Last but not Least: **Test that your installer works** as you intended **before** you release your mod! If you don't want to risk ruining your install, you can create a fake install location by creating a new folder and copying the *dialog.tlk* file into this folder, and create an *override* folder next to it. Copy any modified files you want to use for testing into this "*override*" folder.

The TSLPatcher only checks for the existence of *dialog.tlk* in the folder selected for installation to ensure that it is a "valid" install location. As such you should be able to install your mod into this new folder. Carefully examine the resulting files in this fake *override* folder to make sure everything has ended up as you intended. It is easy to make mistakes, and this way you save both yourself and the users of your MOD some headache later on. :)

Advanced: *TSLPatcher.exe* accepts two **optional** command line parameters. If present, the first parameter must be the name of an INI file with TSLPatcher settings that will be used instead of *changes.ini*. The second parameter must be the name of an RTF file that will be used instead of *info.rtf*.

This can be used to create *shortcuts* or *batch files* for several different install options for the same mod, where each shortcut defines a different INI file as parameter that will be used for installation. To allow users to pick between different install options within the TSLPatcher application it is recommended to use the *Setup List/Namespaces.ini* configuration instead.

3. ChangeEdit

ChangeEdit is a helper application made to make it a bit easier to configure what the TSLPatcher should do with your Mod files. While it will hardly win any awards for user friendliness, it should still make it a bit easier than creating a changes.ini file by hand, and certainly a little harder to make mistakes.

The purpose of *ChangeEdit* is to create a *changes.ini* file. This is the file that the TSLPatcher reads to figure out what it should do when run. As mentioned above this is a plain text file in the standard Windows INI format, and can be viewed and modified by Notepad if needed. Unless you really know what you are doing you should use *ChangeEdit* and only tweak things with Notepad if necessary.

To Create a new *changes.ini* file, select "New..." from the File menu. The tree view in the left panel should now update to show 6 different sections. Click on the section you want to modify to switch the right panel to that view.

Note :   Whenever there is a button with a blue circle with a white "i" inside, clicking that button will display (hopefully) useful information about that feature in ChangeEdit.

Important: Keep in mind as you read on that all files and modifiers are processed in the order they are listed. This usually doesn't matter, but if several modifiers depend on each other (i.e. a new row in portraits.2da refer to a new row in appearance.2da which refer to a new row in heads.2da), then the sequence things are added becomes important.

The different sections are:

3.1. Settings

This section allows you to configure how the TSLPatcher should behave in general terms. There are nine fields here you may edit to customize it to your liking:

3.1.1. Window caption
This is what will be displayed as the name of the main TSLPatcher window in the drag bar at the top.

3.1.2. Confirm message
 If you want to display a custom message in the Yes/No confirmation box that pops up when the user clicks the "Install Mod" button. If you leave this field blank, the default confirm message will be displayed. If you set this to "N/A" without the quotation marks, the confirm box will be disabled altogether and the user will be prompted to select the game folder as soon as they click the Install button.

3.1.3. Log level
This determines how detailed the feedback is that is displayed in the progress log as the TSLPatcher works. There are five feedback levels available:
0) No feedback at all. The text from "info.rtf" will continue to be displayed during installation.
1) Only general progress information will be displayed. Not recommended.
2) General progress information is displayed, along with any serious errors encountered.

3) General progress information, serious errors and warnings are displayed. This is recommended for the release version of your mod.
4) Full feedback. On top of what is displayed at level 3, it also shows verbose progress information that may be useful for a Modder to see what is happening. Intended for Debugging.

3.1.4. Run mode
This should be set to "Installer" if you want to use TSLPatcher to install mods. (If it is set to "Patcher" it will make the TSLPatcher function more like a modding utility where it asks for the location of each file to patch instead of asking for the override folder. The Install Files section is also skipped in this mode.)

3.1.5. Make Backups
This allows you to set if the TSLPatcher should make backup copies of any existing files it finds in the override folder before it modifies or overwrites them. If enabled, backup copies will be created in a folder named "backup" that is created in the same folder as the TSLPatcher.exe is run from.

3.1.6. Progress log style
This should usually be left at *Standard*. A handful of users have reported some strange incompatibility with the RichEdit DLLs they have installed in their system which makes the TSLPatcher terminate when installation begins. If this is set to "Compatibility", the progress log will be displayed in plain text instead. While harder to read, it should eliminate the incompatibility problem in those cases.

3.1.7. Required file name
This is an **optional** field that should be left blank in most cases. If this is set to the name of a file, that file must be present in the override folder of the user or the TSLPatcher will refuse to install the mod. This is mostly useful when making updates to existing mods to make sure that the user already has the mod that should be updated installed.

3.1.8. Required error msg
This **optional** field is used in combination with the above field. If the *Required file name* field is set, this message will be displayed to the user if they don't have the specified file in their override folder.

3.1.9. Ask for Game folder
This can be used to change how the *TSLPatcher* determines where the game folder the files should be installed into is located. If set to "Ask the user..." an *Open Folder* dialog box will open when installation starts, prompting the user to select their game folder. This is the standard behavior.

If you set this to "Look up folder path..." instead, *TSLPatcher* will attempt to find the game folder automatically by looking in the Windows Registry for it. If it fails to locate the path in the registry (which should only happen if the game is not properly installed) it will ask the user for the location with an *Open Folder* dialog box. If you use this setting you should also specify which version (KotOR1 or KotOR2:TSL) of the game your Mod is for to allow the *TSLPatcher* to look for the correct game folder. The default value here is KotOR2:TSL, which will be used if you don't change this setting.

Press the *Save changes* button to save any changes you have made in these fields to the *changes.ini* file.

3.2. TLK Entries

If you created an *append.tlk* file with custom *dialog.tlk* entries in step 2.5 above, this is where you configure the Patcher to apply them to the *dialog.tlk* file.

Press the *Open append.tlk file...* button on top of the right list in the window. This will list all your custom text entries in the list to the right. Select an entry you wish the TSLPatcher to add to *dialog.tlk* and press the *left arrow* icon. The entry will be added to the list to the left. Take note of the value in the left column which should look like StrRef0 for the first entry, with an incrementing number for each entry you add. This is the "Token" you will use in the 2DA and GFF sections to assign the resulting *StrRef* value for this entry to a 2DA cell or GFF field.

(StrRef is short for String Reference, and is an identifier number for an entry in the *dialog.tlk* file.)

3.3. 2DA Files

This section is where you list all the 2DA files the TSLPatcher should modify, and what changes it should do to them. To add a file to modify, Select the *2DA Files* section in the tree view and either choose *Add 2DA File* from the Modifiers menu, or right click on *2DA Files* in the tree view and select *Add 2da file* in the context menu.

You will then be prompted for the name of the file to modify. Check your spelling carefully, type the name in all lowercase letters and be sure to include the .2da extension. E.g. if you want to modify the *appearance.2da* file, type in `appearance.2da` in the box and click OK. Then select the newly added file in the tree view.

The right panel will now show an empty Modifier list (since you haven't added any yet). When the list contains entries, you can double-click one of them to edit that Modifier.

There are now two possible ways to proceed:

3.3.1. Way 1: Compare Files for differences and fabricate modifiers.
To do this, press the Compare button (the icon with the pointing hand) on top of the Modifier list. You will now be prompted to first select an **unaltered** copy of the 2DA file, suitably the one found in your *tslpatchdata* folder. Click Open and you will be prompted to select your modified version of that same 2DA file.

ChangeEdit will now compare the files for differences and fabricate Modifiers to reflect its findings.

Very Important: The Compare function will only look for differing values with no contextual knowledge of how those values are used. Thus, if there are any fields that should have special token values assigned, you will have to go through the fabricated modifiers and **add those tokens by hand**.
For example, if the Compare function added a modifier for a new line in *spells.2da*, you will have to manually assign the proper `StrRef#` tokens to the *name* and *desc* columns to point to your new custom *dialog.tlk* entries (see section 3.2 above), and you will have to assign the `high()` token to the *forcefriendly* or *forcehostile* column.

At any rate, you should look through the modifiers that have been created for you and check that everything appears to be in order. To view or edit a Modifier, double-click it in the list. See section 3.3.2 below for more information on how the 2DA Modifier editor windows work.

3.3.2. Way 2: Specify the changes to be made by hand.
Use the buttons below the Modifier list to either:

Add line to 2da
What it sounds like. A new line will be added from scratch at the bottom of the 2DA file.

Modify line in 2da
Change the value in one or more columns for a line that already exists in the 2DA file.

Copy line in 2da
This creates a new line at the bottom of the 2DA file by making a copy of an already existing row, letting you change the values in the relevant columns while keeping the other column values from the original row.

Add column to 2da
This lets you add a new column at the far right in the 2DA file. Note that not all 2DA files will react to having new columns added.

Note: In addition to creating new modifiers from scratch using the above mentioned buttons, you can select an existing modifier and copy it, and make the desired changes to it. This may be useful when you need to make many Modifiers that are very similar, with only a few values different. Select the Modifier you wish to copy in the list and click the *Compare* icon above the list to copy it.

3.3.2.1. Add 2da line
Clicking this button will prompt you to specify a Modifier label. A Modifier label is an identifier used by the TSLPatcher to keep track of this particular change, and is not something that will be written to any of the game files. You can name the label whatever you want, as long as it only contains alphanumerical characters, no spaces and is **<u>unique</u>** throughout this *changes.ini* file. It is suggested that you name it in a way that will help you remember what the modifier does.

When you have specified a Modifier, the *Add 2da line* window will open. Here you can enter the value to be set in each column for the new row.

The optional *Exclusive Column* box can be set to the name of a column in the 2da file. This will instruct TSLPatcher to only add the new row if no existing rows has the same **value** in this column as what your new line will have. If a matching line exists, **the existing line will be modified** instead with the values you assign to the columns and no new line added. This is useful for avoiding to add duplicate lines if running the installer more than once, and making updates to already installed Mods.

If you have a copy of the 2da file in question in the same folder as your *changes.ini* file ChangeEdit will load the column labels automatically for you into the *Column* dropdown list. If not, press the *Load...* button to the right of the *Column box* and select a 2da file in the open dialog that opens to load all the column labels. This is useful to save you the trouble (and potential source for errors) of having to type in the column names yourself. Next, select the name of the column to assign a value to in the drop down list.

If you have any custom `StrRef#` or `2DAMEMORY#` (more about those below) tokens set you wish to use, or need to use any of the built-in special tokens, all (thus far) assigned tokens will be listed in the *Value* drop down list. You can, of course, type in the value you wish the row to have in the selected column directly in the *Value box*. If you set the value to `high()`, the TSLPatcher will find the highest numerical value of all rows in this column, and set the value for this row to that value + 1.

When you are done, add this column to the list by pressing the *right arrow* icon. Add any other columns you wish to assign values to in the same manner. If you need to edit the value of a column you have already added, select it in the list to the right and press the *left arrow* icon.

Default value
Any columns you have not assigned a value to will have the default value **** set.

3.3.2.1.1. Special "columns"
There is one special value that can be set in the *Column* box that are not really a column name: `RowLabel` sets the value of the rowlabel for the line (the column on the far left in KotORTool's

2da editor). If this is left out, it will be set to the line number the row is added as.

If you wish to assign the line number that your new row will be added as to memory, type in
2DAMEMORY followed by a slot number in the Column input box rather than the name of a
column. The slot numbers start at 1 (i.e. 2DAMEMORY1) and go up to as many as you need. Then,
in the value box, type in RowIndex. This will save the line number of the row in the
2DAMEMORY token, which can then be assigned to cells in other 2DA files, or fields in GFF files.
In the same way, any column label can be assigned to a 2DAMEMORY token, causing the value in
that column for the new line to be stored.

3.3.2.2. Modify 2da line
Clicking this button will prompt you to specify a Modifier label. A Modifier label is an
identifier used by the TSLPatcher to keep track of this particular change, and is not something
that will be written to any of the game files. You can name the label whatever you want, as long
as it only contains alphanumerical characters, no spaces and is **<u>unique</u>** throughout this
changes.ini file. It is suggested that you name it in a way that will help you remember what the
modifier does.

When you have specified a Modifier, the *Modify 2DA line* window will open. First set which
row you wish to modify. This can either be done by specifying the RowIndex (i.e. line number)
of the row, or by specifying the RowLabel value of the row (far-left column in KotORTool's
2DA editor).

An extra, special case way of specifying the row to modify that only works for some 2DA files
is LabelIndex. This will look in the *label* column for a value. This obviously only works in
2DA files which have a label column, and only works reliably in files where the value in the
label column is unique for each row.

When you have specified a row to edit, set the columns that should be changed for this row,
and assign their new values:

Press the *Load...* button to the right of the *Column box* to load all the column labels from a 2da
file to save you the trouble (and potential source for errors) of having to type in the column
names yourself. Then select the name of the column to assign a value to in the drop down list.

If you have any custom StrRef# or 2DAMEMORY# tokens set you wish to use, or want to use
any of the built-in special tokens, all (thus far) assigned tokens will be listed in the *Value* drop
down list. You can, of course, type in the value you wish the row to have in the selected column
directly in the Value input box. If you set the value to high(), the TSLPatcher will find the
highest numerical value of all rows in this column, and set the value for this row to that value +
1.

When you are done, add this modified column to the list by pressing the *right arrow* icon. Add
any other columns you wish to change values for in the same manner. If you need to edit the
value of a column you have already added, select it in the list to the right and press the *left
arrow* icon.

Note: Only add columns you wish to change the values for. The values in all other columns
         will be kept untouched.

If you wish to assign the line number that your new row will be added as to memory, type in `2DAMEMORY` followed by a slot number in the *Column box* rather than the name of a column. The slot numbers start at 1 (i.e. `2DAMEMORY1`) and go up to as many as you need. Then, in the value box, type in `RowIndex`. This will save the line number of the row in the `2DAMEMORY` token, which can then be assigned to cells in other 2DA files, or fields in GFF files.

You can also assign the `RowLabel` or the name of any column to a `2DAMEMORY` token. The value in the 2DA file in that column is then saved in the `2DAMEMORY` token for later use.

3.3.2.3. Copy 2da line

Clicking this button will prompt you to specify a Modifier label. A Modifier label is an identifier used by the TSLPatcher to keep track of this particular change, and is not something that will be written to any of the game files. You can name the label whatever you want, as long as it only contains alphanumerical characters, no spaces and is **<u>unique</u>** throughout this changes.ini file. It is suggested that you name it in a way that will help you remember what the modifier does.

When you have specified a Modifier, the *Copy 2DA line* window will open. First select which row you wish to make a copy of for your new line. You can specify this blueprint row either by `RowIndex` (i.e. line number) of its `RowLabel` value (far-left column in KotORTool's 2DA editor).

The optional *Exclusive Column* box can be set to the name of a column in the 2da file. This will instruct TSLPatcher to only copy and add the new row if no existing rows has the same **value** in this column as what your new line will have. If a matching line exists, **the existing line will be modified** instead with the values you assign to the columns and no new line added. This is useful for avoiding to add duplicate lines if running the installer more than once, and making updates to already installed Mods.

Then you can specify the columns whose values you wish to change from the original:

If you have a copy of the 2da file in question in the same folder as your *changes.ini* file ChangeEdit will load the column labels automatically for you into the *Column* dropdown list. If not, press the *Load...* button to the right of the *Column box* and select a 2DA file in the open dialog that opens to load all the column labels. This is useful to save you the trouble (and potential source for errors) of having to type in the column names yourself. Next, select the name of the column to assign a new value to in the drop down list.

If you have any custom `StrRef#` or `2DAMEMORY#` tokens set you wish to use, or need to use any of the built-in special tokens, all (thus far) assigned tokens will be listed in the *Value* drop down list. You can, of course, type in the value you wish the row to have in the selected column directly in the Value input box. If you set the value to `high()`, the TSLPatcher will find the highest numerical value of all rows in this column, and set the new value to that value + 1.

When you are done, add this column to the list by pressing the *right arrow* icon. Add any other columns you wish to change values for in the same manner. If you need to edit the value of a column you have already added, select it in the list to the right and press the *left arrow* icon.

3.3.2.3.1. Special "columns"

There is a special value that can be set in the *Column box* that is not really a column name:

`NewRowLabel` allows you to set a new `RowLabel` value (the column on the far left in KotORTool's 2da editor) for your new row.

If you wish to assign the line number that your new row will be added as to memory, type in `2DAMEMORY` followed by a slot number in the *Column box* rather than the name of a column. The slot numbers start at 1 (i.e. `2DAMEMORY1`) and go up to as many as you need. Then, in the value box, type in `RowIndex`. This will save the line number of the row in the `2DAMEMORY` token, which can then be assigned to cells in other 2DA files, or fields in GFF files.

You can also assign the `RowLabel` or the name of any column to a `2DAMEMORY` token. The value for the new row in the 2DA file in that column is then saved in the `2DAMEMORY` token for later use.

3.3.2.4. Add 2da column
Clicking this button will prompt you to specify a Modifier label. A Modifier label is an identifier used by the TSLPatcher to keep track of this particular change, and is not something that will be written to any of the game files. You can name the label whatever you want, as long as it only contains alphanumerical characters, no spaces and is **<u>unique</u>** throughout this changes.ini file. It is suggested that you name it in a way that will help you remember what the modifier does.

Next, set the desired column label for your new column in the "New column label" input box. Column labels may be at most 16 characters, should be all lowercase and only contain alphanumerical characters. You may also set a custom *Default value* if you wish. This value will be assigned to all rows in the new column unless you assign a new value for a row specifically.

You may then assign values to rows for your new column. Select if you want to identify the row by the `RowIndex` (i.e. line number) or the `RowLabel` (leftmost column in KotORTool's 2DA Editor).

If you have any custom `StrRef#` or `2DAMEMORY#` tokens set you wish to use, or need to use any of the built-in special tokens, all (so far) assigned tokens will be listed in the *Value* drop down list. You can, of course, type in the value you wish the row to have in the selected column directly in the *Value box*.

When you are done, add this row modifier to the List by pressing *right arrow* icon. Add any other rows you wish to change values for in the same manner. If you need to edit the value of a row you have already added, select it in the row list to the right and press the *left arrow* icon.

If you for some reason want to store a value for a row in the new column, Select `Memory` in the row drop down box and type in a slot number following the `2DAMEMORY` that appears in the input box. The slot numbers start at 1 (i.e. `2DAMEMORY1`) and go up to as many as you need. Then, in the value box, type in "`I`" followed by the line number of the row, or "`L`" followed by the line number. This will save the value of that line in the new column in the `2DAMEMORY` token, which can then be assigned to cells in other 2DA files, or fields in GFF files.

3.4. GFF Files

GFF is a file format used for a variety of different file types in the game, such at UT* templates, DLG dialog files, the *global.jrl* Journal entry file, just to mention a few.

TSLPatcher can modify the value of the fields in GFF files found in the user's *override* folder (or place a copy of the file there if it doesn't already exist before modifying it).

To do this, select the GFF Files section in the tree view and either choose *Add GFF File...* from the Modifiers menu, or right-click the section and choose *Add GFF File...* from the context menu. Type in the name of the file to modify, be sure to include the file extension (i.e. `.dlg`, `.uti` etc...). Then select the newly added GFF file in the tree view to view its Modifier list.

The TSLPatcher normally modifies GFF format files in place, rather than overwrite them, if it finds that one of the files already exists within the user's *override* folder. If you want existing files to be overwritten by a fresh copy from *tslpatchdata* before they are modified, check the *Replace check box*.

The *Destination box* (new in v1.2.5) allows you to determine where the GFF file is located (if modifying existing files) or should be saved (if adding new files). There are two options here:

1) If the *Destination box* is empty or set to "*override*" (without the quotation marks), the files will be modified and/or placed in the override folder as before.

2) If the Destination box is set to the relative path (within the game folder) and name of an *ERF/MOD/RIM* archive file, the modified *GFF* file will be modified (if already existing) or inserted (if not already existing or the Replace setting is set) into this archive file instead of being placed in the *override* folder.

Important: If you set an *ERF/RIM* file to save your modified files in you must specify the relative path from the game folder to where the *ERF/RIM* file is located. If, for example, you want to modify a *GFF* file within the file **myarea.mod** located in the **Modules** folder, you would set the Destination to **Modules\myarea.mod**.

There are three main things you can do in the GFF panel to create instructions to modify GFF files, which are divided into two distinct sections:

3.4.1. Modifying existing GFF Fields
To change the value of a GFF field, type in the label of the field in the *GFF Field* box. If the field is not located at the root-level (top-level) of the GFF Field tree, you must specify a full label path to the field, where each label or list-index is separated by a backslash ("\") character. E.g. to change the subtype of an item property, you might enter `PropertiesList\0\Subtype`. Be careful when specifying the label. They are case sensitive, thus *Comments* and *comments* are not the same label. Use the GFF Editor to view the field tree in the GFF file you wish to modify.

Click the *Load* icon to the right of the *GFF Field box* to load all currently existing field labels from an existing GFF file. This may save you some typing and may help preventing typos in the field labels. If a file with the same name as the one you currently make modifiers for exists in the same folder as your *changes.ini* file (e.g. a *tslpatchdata* folder) it will be loaded directly. If the file could not be found you will be prompted to select one to load.

In the *Value box*, enter the new value you wish the field to have. Take care to only enter data of a type that the type of field can handle. Trying to assign a text string to an INT field is not a good

idea, for example. If you wish to assign a value stored in a `StrRef#` or `2DAMEMORY#` token to a field, all (so far) assigned tokens are listed in the drop down list, so you can select the relevant token there.

Press the *up arrow* icon to save the Modifier to the list. To edit a field value you have already entered, select it in the list and press the *down arrow* icon.

Note: A few complex field types requires a somewhat more arcane procedure to specify their value:

ExoLocString fields will require you to append a directive to what to edit at the end of the field label, since they are made up of one StrRef value and optionally one or several sub-strings. If your *ExoLocString* field has the label `Comments`, you would assign a new *StrRef* value to it by typing `Comments(strref)` in the *GFF Field box*. For the substrings you add `lang` followed by the *language+gender ID number* of the string, like: `Comments(lang0)` to modify the English localized string the `Comments` field.

Orientation fields will require you to specify each part value separated by a pipe ("|") character. For example `0.05|1.2|0.0|10.0`. Orientation fields are made up of 4 decimal values.

Position fields will require you to specify each coordinate value separated by a pipe ("|") character. For example `0.05|1.2|0.0`. Position fields are made up of 3 (x,y,z) decimal values.

STRUCT and *LIST* fields have no value, since they only contain other fields and carry no data themselves.

Note: In TSLPatcher v1.2.7b9 and onward it is possible to use the value stored in a `2DAMEMORY#` token as field path+name to modify a value. This can be useful in situations where you need to dynamically update fields that don't exist in the file already and you are having the patcher add dynamically (see section 3.4.2 below).

It can, for example, be used to insert new branches into the dialog tree in a DLG file by first adding the new fields and storing the path+name to the `RepliesList/EntriesList index` fields in a token storing the list-index of the new corresponding `ReplyList/EntryList` structs in another token, and then insert them into the Index fields after they have been created.

3.4.2. Adding new GFF Fields
To Insert new fields into a GFF file, press the *Manage new fields...* button (icon with blue arrow with a plus beneath) next to the red arrow icons in the GFF Modifications panel.

A new window will open where you can add New Field modifiers to the file. Press the *Add new field...* button (blue arrow and red plus icon) above the field list to add a new field. You will be prompted to specify a Modifier label. A Modifier label is an identifier used by the TSLPatcher to keep track of this particular change, and is not something that will be written to any of the game files. You can name the label whatever you want, as long as it only contains alphanumerical characters, no spaces and is **__unique__** throughout this *changes.ini* file. It is suggested that you name it in a way that will help you remember what the modifier does.

Back in the *Add GFF Field* window you can now set the specifics for your new field. Available input fields are:

3.4.2.1. *Field Type*
Use this **first** to set the data type of the field you wish to add. Different data types can store different kinds of values. Press the blue "`i`" button in this window to view more information about the supported GFF field data types. Depending on what you select here, the remaining input fields may be dimmed down or enabled.

3.4.2.2. *Label*
Type in the label of your new field here. A label is an identifier key used to retrieve the value from the GFF file. A label can be at most 16 characters long, may only contain alphanumerical characters and no spaces. Labels must be unique on each level of the GFF tree (i.e. within the same STRUCT), but may be named identically in separate parts of the field tree. All fields **must** have a Label, except *STRUCTs* added to a *LIST* field parent, which has no label.

3.4.2.3. *Path*
This specifies where in the GFF field tree you wish to add the new field. Leave this blank to add the field at the root (top) level of the GFF file. Separate each field label (or list index) in the hierarchy with a backslash ("`\`") character if you wish to add your fields deeper in the tree.

Note: You may add new fields below a *STRUCT* or *LIST* parent field only. *LIST* fields can only contain *STRUCT* fields, while *STRUCT* field can contain fields of any type.

3.4.2.4. *Value*
This is where you assign what value you wish your new field to have. Be careful to only enter values that will fit within the selected field type.

You may use a `2DAMEMORY#` or `StrRef#` token as value. *STRUCTs* and *LISTs* are container fields that only holds collections of other fields, and thus have no value. *ExoLocStrings* have no value since they are made up of several separate data fields. All other field types should have a value set.

3.4.2.5. *StrRef*
This field is only used for *ExoLocString* type fields. When adding an *ExoLocString*, you can set the *dialog.tlk StrRef* value here. This field accepts `StrRef#` and `2DAMEMORY#` token values. Set this field to `-1` if your *ExoLocString* doesn't use any value in *dialog.tlk*.

3.4.2.6. *Type Id*
This field is only used when adding *STRUCT* type fields. When adding a *STRUCT*, you may set its *Type ID* here. If unused, just set it to 0. The *Type ID* is used for different things depending on what your *STRUCT* is used for. Check with a GFF Editor if it appears to be used used for what you are trying to do. The *Type Id* must be a number, 0 or larger.

Note: If your struct is added to a *LIST*, putting `ListIndex` in this box will make TSLPatcher insert the index the *STRUCT* was added as in the parent *LIST* as *Type ID*. This is used in some places, such as in the `Categories` *LIST* in *global.jrl* where the *Type Id* matches the List index.

3.4.2.7. *Localized strings*
This list field (with its associated buttons) is only used for *ExoLocString* type fields. Here you may add localized substrings to your *ExoLocString*. Put the *Language+gender ID* number of the string in the first column, and type in the text in the second column. (Language ID 0 is for English text, which is usually what you need to use.) If you wish to add more than one localized substring, press the *Add* button to the left of the list.

3.4.2.8. *Index token*

Only used for *STRUCT* fields added to a *LIST* parent field. Put a `2DAMEMORY#` token in this box to store the List index number of the new struct. This field should be left blank if you don't need to store the list index for later use.

3.4.2.9. *Path token*
Set this to the name of a `2DAMEMORY#` token if you wish to store the Field path+name this particular field is added as. This can be used later to modify this field as described in section 3.4.1 above. This field should be left blank if you don't need to store the field path+name.

3.4.2.10. *Sub-fields*
Only used for *STRUCT* and *LIST* container fields. If you add a new *STRUCT* or *LIST* field, you most likely want to add new fields below/inside this new container as well (since an empty *LIST* or *STRUCT* is kind of useless). This button allows you to add other fields below your new *LIST/STRUCT* in the hierarchy, which will get their path dynamically from the parent.

First save your new field, then click the *Edit...* button to open a new Editor window, listing all sub-fields directly below (or contained inside, however you wish to visualize it) your *LIST/STRUCT*. This new window will work just like the main Editor window for the GFF file, with the exception that the Path field will always be dimmed down. **Never set a Path for sub-fields**, they will get their path from your parent field.

Press the *right-arrow* icon to add your new field to the list to the right. To view or modify a field listed here, either select the field and press the *left arrow* icon, or double-click in the list.

3.4.3. Compare two GFF files for differences to create modifiers
Click on the Icon with the pointing hand above the Modifier list to access the compare function. This function will ask you to select two copies of the same file, one unaltered and one that has been modified. It will then compare the two and manufacture GFF Modifiers from the things it finds that has been changed in the modified file.

When the comparison is done, *ChangeEdit* will show a status report message informing you of how many Modified fields (added to the Changed fields Modifier List) and how many New fields (added to the New fields window) it has found and created Modifiers for.

Keep in mind that the Compare function is intended to save you some time with tedious data entry, and is not meant as a one-click solution for all manner of GFF editing needs. Depending on the situation and type of files you work with it will be more or less useful. Some modifiers may need some manual tweaking afterwards to function as intended when they refer to other data, or need dynamic data inserted.

Note: *ChangeEdit* will only detect and make Modifiers for the type of operations that the *TSLPatcher* is capable of performing on the GFF files. E.g. if the modified file has deleted fields compared to the original *ChangeEdit* will not notice, since *TSLPatcher* currently is unable to delete fields from GFF files.

Important: Keep in mind that *ChangeEdit* will make modifiers directly for the new data it finds, in the order it finds them in the file, without any knowledge about how this data is used or interlinked with other fields. Always double-check the Modifiers it has created to make sure everything is in order, and insert any Tokens you wish to use for assigning dynamic values to fields.

3.5. Install Files

The Install Files section lets you configure the TSLPatcher to move files that it hasn't modified in any of the previous sections to their proper place. You can use it to help the user put files in their proper location (*override*, *Modules*, *StreamVoice* folders etc).

Important: Do **not** add any files that have been modified by any of the other sections to the InstallList, or the modified files might be overwritten! The other sections already modify files in the game folder. The only exception to this is ERF files which has had files added to them by those sections. They must still be added to the Install list to be put in their proper places.

To add files, you can type in the name of the folder the file should be moved to in the *Folder Name box*. All folders are assumed to be located directly in the main game folder. If you need to install files into folders inside any of those folders, you must specify the relative path to that folder, with each folder name separated by a backslash character.

E.g. to install a file in the *_HuttHap* folder inside the *AVO* folder inside the *StreamVoice* folder, type in `StreamVoice\AVO\_HuttHap` in the *Folder name box*. If the specified folder(s) do not already exist in the user's game folder, they will be created.

Note: If you want to put files directly into the game folder, set the folder name to "`.\`".

Either type in the name of the file directly in the *File name box*, or press the *Select...* button to the right of that box to select a filename with a standard Windows Open File dialog box.

If you wish to replace any existing files with the same name at the specified install location, check the *Replace Existing...* check box. **Only** do this if you can be reasonably certain it won't mess up the user's game or any other mods they have installed. If this box is not checked and the file already exists, a warning will be issued in the TSLPatcher progress log and the file will be skipped.

When everything is set, press the *up arrow* icon to add the file to the file list.

If you wish to add a lot of files to the same folder with the same Replace setting, the above procedure can be somewhat tedious. If so, press the *Mass Add Files* icon next to the arrow icons. A dialog box will open asking you to specify the folder the files should be installed in, and if existing files should be replaced. When you press the *Select* button in this dialog box, a standard Open File dialog box will open where you can select multiple files by holding down CTRL or SHIFT when you click on files in the list. All selected files will be added to the file list when you click the Open button.

To modify a file already added to the file list, either select it in the list and click the *down arrow* icon, or double-click the file in the list. It will then be loaded into the lower box to allow you to edit it.

3.6. Script Source

This section allows you to specify a number of *NSS script source files* that the TSLPatcher should process for any 2DAMEMORY# or StrRef# tokens, substitute the tokens with their stored value, compile the modified source code and put the resulting NCS file in the user's *override* folder.

Either enter the name of a *NSS* script file, or press the *Select...* button to the right of the *File name box*. Check the *Replace existing* box if you want the TSLPatcher to replace any NCS files with the same name already existing in the at the destination when installing. If this is unchecked and the NCS already exists in at the destination a warning will be posted in the TSLPatcher progress log and the file will be skipped. This might result in the installation failing if the file in question doesn't already have what your mod needs.

If you want the resulting *NCS* file to be installed in the user's *override* folder, leave the *Destination box* blank or set it to "*override*" (without the quotation marks). If you want the *NCS* file to be saved inside an *ERF/MOD/RIM* file instead, put the relative path (from the game folder) and name of that ERF format file in this box.

Important: You must specify the relative path from the main game folder to where the *ERF/RIM* file you want to save your recompile script in exists. If you for example want to save your script inside the **myarea.mod** file located in the **Modules** folder you would set the Destination box to **Modules\myarea.mod**.

Only *override* or the path\name of an *ERF/RIM* file is a valid value in the Destination box. This can currently not be used to put the file in another folder than the *override* within the game folder. If someone needs that functionality, please let me know.

Press the *up arrow* icon to add your new *NSS* file to the list. Double-click a file in the list or select it and press the *down arrow* icon to edit an entry in the list.

Important: If your script uses any **include** files, all those include files must be placed in the same folder as the script, but not added to the Script Source list **unless** you need the TSLPatcher to change anything in them.

If that is the case, those include files must be added to the Script list **before** the scripts they are used by, so they are processed before the using script is compiled. Include files are only processed for tokens, never compiled. Make sure your include files does not contain any main() or StartingConditional() functions, even ones commented out, or the TSLPatcher may try to compile them.

In the *NSS* scripts themselves, wherever you want a value to be inserted, type in the name of the token to insert, enclosed in hash ("#") characters into the source code.

E.g. to insert the value stored in the 2DAMEMORY1 token as a parameter in a call to the function ChangeObjectAppearance(), and the StrRef0 token of a newly added *dialog.tlk* entry into a script using the BarkString() function, your source code script line might look like:

```
ChangeObjectAppearance(OBJECT_SELF, #2DAMEMORY1#);
BarkString(OBJECT_INVALID, #StrRef0#, 25, 20);
```

Important: For this to work, the files *nwnnsscomp.exe* and *nwscript.nss* must be placed within the *tslpatchdata* folder, even if you use multiple Setup Groups within subfolders. As of version 1.2.7b4, a modified version of nwnnsscomp, made by *tk102*, has been specifically modified for use with TSLPatcher and should be

included in the TSLPatcher RAR archive. While other versions may work, the Patcher was made primarily to work with this version. Unless you have a very compelling reason to use another I suggest you use the provided compiler.

The *nwscript.nss* file can be found within the scripts.bif file in the game data and can be extracted with KotorTool. Be aware that non-US versions of the game ship with an error in this file, which must be corrected prior to use.

3.7. Soundset files

This section allows you to modify sound entries in **SSF** format Soundset files, allowing you to set the StRef of a SSF entry to a TLK entry you have added to *append.tlk* to be added to the user's *dialog.tlk* file.

Either select *Add SSF File...* from the Modifiers menu, or right click on the *Soundset files* section in the tree view and choose *Add SSF File...* from the context menu. You will be asked for the name of the SSF file to modify. Specify a name, including the SSF extension and click the OK button to proceed.

The right panel in the *ChangeEdit* window will now display a list of all sound entry modifiers you have added for this Soundset file. To add a modifier, select an Entry from the *SSF Entry drop down menu* below the grid, and type in either a `StRef#` token, a `2DAMEMORY#` token or a valid *StRef* number in the *StRef input box* below it. The drop down list of the *StRef box* will list any `StRef#` and `2DAMEMORY#` tokens that have been assigned in this *changes.ini* file which you can choose from to save some typing.

To save your modifier, click the red up-arrow icon and it will be added to the grid. To edit an existing modifier, either double click it in the grid, or select it in the grid and press the red down-arrow icon. To delete a modifier, select it in the grid and click the trashcan icon above the grid.

Like with 2DA and GFF files, the *TSLPatcher* will check if a SSF file already exists in the *override* folder of the user and modify the existing file there if found. If you want it to always install and modify a fresh copy of the SSF file from the *tslpatchdata* folder, overwriting any existing SSF file with the same name in *override*, check the *Replace file...* check box near the bottom of the SSF panel.

Note: The *Unknown(#)* sound entries are present in the SSF format, but I have been unable to determine what they are used for, or if they are just reserved and unused. If you figure out what any of those entries are for, please let me know.

3.8. Setup List

This section is not accessible from the Tree View since it is independent of a particular *changes.ini* file. This functionality is accessed from the *Files* menu. This functionality was added in TSLPatcher v1.2.7.

Quick overview

A *Setup List* is a way of providing more than one possible way of installing a Mod from the same Installer. It may for example be used to let the user choose to update a previously installed version of the Mod, or install it completely from scratch. Technically it will allow you to create multiple *changes.ini* and *info.rtf* files and present the user with a list of options where they pick which set to use when TSLPatcher is launched. This is configured in a separate INI format file that must always be named *namespaces.ini*, and be located directly in the *tslpatchdata* folder.

If the *namespaces.ini* file is present within the *tslpatchdata* folder a dialog box is shown when TSLPatcher (1.2.7 or later) is launched, presenting the user with a dropdown list of available installation options, and a short description of each. When the user has selected which Setup to use, the associated INI and RTF file are loaded and the main TSLPatcher window is shown like before. If the *namespaces.ini* file does not exist within the *tslpatchdata* folder, TSLPatcher will not show this dialog box and just go to the main window like it usually does.

Configuration

To configure a *namespaces.ini* file, launch *ChangeEdit.exe* and choose Setup Lists from the File menu. In this submenu you can choose to either create a new *namespaces.ini* file, or open an existing one for editing. If you create a new one, note that it must always be named exactly *namespaces.ini* for the TSLPatcher to read it. A new edit window will open.

The list to the left lists all the different sets of Setup INI/RTF files present in the current *namespaces.ini* file. To edit an existing Setup, click it in the list, and its data will be loaded into the box on the right.

To create a new Setup, click the *New...* icon above the list. You will be asked to specify an identifier label for your new Setup. Like with modifier labels in *changes.ini*, this identifier should be unique within the *namespaces.ini* file and only contain alphanumerical characters or underscores and no spaces. While it will never be displayed to the user, only used internally, you should pick something that helps you remember what the setup is.

To edit the values of a selected or newly created Setup, use the input boxes in the panel to the right. The following fields are available:

Config file name - This is the name of the INI file the TSLPatcher will look in for instructions on what to do. This is usually changes.ini, but you can specify another name here if you have more than one configuration file in the *tslpatchdata* folder.

Info file name – This is the name of the RTF format document that will be displayed in the text area in the main TSLPatcher window, usually containing the ReadMe file or special installation instructions. This is usually *info.rtf*, but you can pick another name for your setup if you have several in the *tslpatchdata* folder.

Data folder – This field is optional. If left blank the TSLPatcher will look for the two above named files, as well as any data files that are to be installed, within the *tslpatchdata* folder as usual. If this field is set to the name of a sub-folder created within the *tslpatchdata* folder, TSLPatcher expects the above named INI and RTF file, along with **all** data files that should be

installed, to be present within that folder instead of in the *tslpatchdata* folder.

Name – This is a descriptive name of this particular setup. This text will be displayed in the dropdown menu in TSLPatcher where the user can choose which Setup to use for installation.

Description – This is a short description text of this Setup, which should explain what it means. This text will be displayed in the information box in TSLPatcher when the user selects the Setup in the dropdown menu.

When you have typed in your desired values, press the *Save Changes* button to commit the values to the *namespaces.ini* file.

Important: For multiple setups that are placed in sub-folders and use *nwnnsscomp.exe* to recompile scripts using include files before installing them you must place *nwnnsscomp.exe* and *nwscript.nss* in the *tslpatchdata* folder, not in the individual sub-folders. You must use the custom version of *nwnnsscomp.exe*, modified by tk102 for this purpose, or the compilation will not work.

4. Troubleshooting

Here are some problems that I have gotten a bunch of questions about, that I can remember at the moment. If there are other problems that might be worth addressing here, please let me know.

4.1. *2DAMEMORY Trouble.*
You: Values saved to a 2DAMEMORY token don't appear to be written when I assign it to something!
Me: Make sure you have assigned to token a value BEFORE you try to use it. All modifiers are processed in the order they are listed. If you try to read a token before it is set, bad things will happen.

4.2. *RichEdit Error*
You: I get "RichEdit Line Insertion error" when trying to Install something! What's going on?
Me: I wish I knew. This is some obscure bug that apparently presents itself when the user has some old/custom/odd version of the RichEdit DLL files installed on their system. The RichEdit DLLs are a component that comes with Windows, though some applications (such as MS Office) overwrites them for some peculiar reason. I have no idea what the cause of this problem is, and I am unable to reproduce the problem. To work around it, go to the Settings section in your *changes.ini* file and change the Progress Log Style setting to "Compatibility". (Or set "PlaintextLog" to 1 in the [Settings] section if you do it the hard way with Notepad).

4.3. *Ambiguity problems*
You: Changes applied to 2DA/GFF files don't seem to take effect in-game!
Me: Make sure you don't have copies of those files within sub-folders of your override folder. The TSLPatcher only looks for files to modify in the override folder itself, not in sub-folders.

4.4. *Failed installation – reverting files*
You: Something went wrong during Mod installation. How do I undo the changes that have been made?
Me: Look for the folder named "backup" in the same folder as the TSLPatcher.exe was run from. Copy all files from this back to their proper place. There should be a copy of the TSLPatcher progress log in a file called installog.rtf in the same folder that lists all files that have been changed.

4.5. *Interface strangeness*
You: The user interface of the ChangeEdit and TSLPatcher applications look odd when I'm using the modern, "bloated" look introduced with Windows XP. What is wrong?
Me: This is "normal". Sorry, it's the best I can do in my spare time when having to use an 8 years old RADevelopment environment (purchasing newer versions of Delphi is just too expensive.) and it's too much work to switch to another environment/language at this point. :)

5. Partial Change History

This list is rather incomplete since I have a tendency of forgetting to write down what I'm doing. And since this has grown way beyond what I initially had planned for it, I didn't bother writing change notes when doing the earliest versions. :)

Change Log for Version 1.2.8b1 (WIP):

2006-08-09   Changed the *InstallList* functionality to allow installing files into ERF/RIM archives as well, and not just folders within the game folder. This destination is set the same as you set the folder, only specify a filename at the end of the relative path (from the game folder) as well. (E.g. `Modules\904mal.rim`)

Changed the *InstallList, GFFList* and *CompileList* to allow renaming files from the source to install using the keys `!SaveAs` (to change the name the file is installed as) and `!SourceFile` (to specify another name than is listed for the file to copy.) These keys are added to the sections for the file they should affect. This is not yet added to ChangeEdit but must be added by hand in the INI file for now.

Added a "configuration summary" button to the main TSLPatcher window, allowing the user to view a brief overview of which files the TSLPatcher is set up to modify, and other relevant info, before deciding whether to proceed with installation.

Various fixes to bugs I discovered that slipped through the testing with the rather substantial changed made in version 1.2.8b0.

Change Log for Version 1.2.8b0 (REL):

2006-08-06   Changed how the ERF/RIM insertion of GFF and NCS files work to make it a bit more useful. It'll now work directly with such files located in the game folder or any subfolders, and it will modify existing GFF files instead of overwriting unless the Replace setting is set.

Changed processing order for the InstallList to allow placing new ERF/MOD/RIM files in their proper location before the GFF/Compile sections run so they can save any necessary modified files into those archive files. The different install phases now run in this order: TLK Appending, Install List, 2DA changes, GFF Changes, Script compilation, SSF Editing.

Extended the ERF insertion functionality to support RIM format files as well.

Change Log for Version 1.2.7b9 (REL):

2006-07-23   Added new value keyword to the "Add GFF Field" sections. If "!FieldPath" is assigned to a 2DAMEMORY# token there, the full path+name of that field will be stored in the token.

Modified the "Modify GFF Field" section to allow using 2DAMEMORY# tokens as field keys. Unlike other sections, values cannot be assigned to a 2DAMEMORY token here. Assigning a value to a 2DAMEMORY# token here will not store the value in the token, but rather insert it into the field whose path+name is stored in the token. This and the above change should allow inserting new branches into DLG files.

Updated ChangeEdit's "New GFF Field" editor with a FieldPath token input box, and modified the "Modify GFF Field" panel list to show both Added and Modified fields and the ability to re-arrange them (since modifiers with tokens as keys must always come below the Add Field section in which they are set).

Change Log for Version 1.2.7b8 (Unreleased):

2006-07-20   The main TSLPatcher window now shows the installation path both when reading it from the registry and when then user has selected it.

Added a "!SourceFile" special key to all sections except TLK and 2DA, which allows specifying the name of the file that should be used as a blueprint if it needs to be copied to the installation destination. The file will be renamed to the section name when being installed, this is just to allow different install options to coexist in the same folder which requires different versions of the same file to work with.

Added Open dialog box buttons to allow selecting an INI and RTF file in the Setup List editor in ChangeEdit, instead of having to type the names in manually.

Change Log for Version 1.2.7b7 (REL):

2006-07-08   Some improvements to the user interface of ChangeEdit. The "ExclusiveColumn" key now has a text box of its own and is not added to the column list. Column labels are now loaded into the dropdown boxed in the Add/Copy 2da line editors automatically if the 2da file exists in the same folder as the *changes.ini* file (usually *tslpatchdata*). Fixed TSLPatcher to display the folder installation will be made to in the statusbar if the "Let user select" option is not set. Modified TSLPatcher to allow the "ExclusiveColumn" key to occur anywhere in a 2da modifier section, not just at the top, to eliminate a potential source for errors.

Change Log for Version 1.2.7b5 (REL):

2006-05-28   Added option for the Patcher to look up the game folder locations in the Registry instead of asking the user for the location. Two new keys under the **[Settings]** section determine if this is active. If `LookupGameFolder` is set to 1 the game paths will be looked up in the registry. If the `LookupGameNumber` key is set to 1 the path to *KotOR1* will be looked up, if set to 2 *KotOR2:TSL* will be used.

Change Log for Version 1.2.7b4 (REL):

2006-05-11   Fixed most problems related with multiple setups and processing and recompiling scripts. Made changes to make the TSLPatcher work better with the custom version of *nwnnsscomp* that tk102 kindly provided.

Change Log for Version 1.2.7b1 (REL):

2006-04-29   Added support for multiple configuration files and a somewhat user-friendly interface to choose which set to use. This allows for creating mod installers with multiple installation options.

Change Log for Version 1.2.6b3 (REL):

2006-03-09   Added a section for modifying SSF soundset files to assign dynamic StRef

values to its sound entries.

2006-03-19   Fixed bug which caused script compilation to fail if the TSLPatcher was run on a Windows 98 or Windows 2000 system.

2006-03-25   Updated *ChangeEdit* with a GFF Compare function and a button to copy existing 2DA modifiers. Lots of varying interface improvements (I hope) have also been made in *ChangeEdit* to make it less annoying to use.

Change Log for Version 1.2.5a (REL):

2006-02-03   Added primitive support for adding modified GFF files and recompiled NCS files to ERF archive files.

2006-02-05   Fixed bug preventing compilation of scripts using include files.

2006-02-05   Modified CompileList functionality to allow TSLPatcher to modify include files and not just compilable scripts.

Change Log for Version 1.2a (REL):

2006-01-09   Updated Patcher to use the new GFF Class for all GFF operations.

2006-01-10   Added functionality for the Patcher to add new fields to GFF files.

2006-01-12   Added functionality for the Patcher process NSS files for tokens and compile them with NWNNSSComp.exe.

2006-01-14   Added "ScriptCompilerFlags" key to "Settings" section of INI that allows setting of extra commandline parameters to nwnnsscomp.exe.

2006-01-18   If the TypeID of a STRUCT added to a LIST is set to "ListIndex", that will be substituted for the index in the LIST the STRUCT is about to be added as.

2006-01-24   Changed InstallList behavior to create the specified folder path if it does not already exist, instead of skipping that folder.

2006-01-26   Added an optional "!ReplaceFile" key for GFF Modifier lists which will make the file be overwritten rather than modified in place if it already exists in the override folder.

2006-01-27   Made Patcher show nwnnsscomp.exe output in the progress log if the LogLevel is set to 4 (Verbose) to help with debugging.

Change Log for Version 1.1.7 (REL):

2005-08-23   Added support for high() token when assigning a RowLabel to a new line, not just a copied line like before.

2005-08-23   Added a "Required" Key to the "Settings" section. When set to a filename that file must exist in the override folder in order for the installer to proceed. If it does not exist, it will log an error

message found in the "RequiredMsg" key.

Change Log for Version 1.1.6 (REL):
2005-07-31   Added fallback plaintext logging since it seems calling RTF a
                 "standard" would be a mistake... Version incompatibilities++ :/
                 Add a "PlaintextLog" key under "Settings" in the INI to use it.

Change Log for Version 1.1.2: (REL):
2005-06-07   Added "ExclusiveColumn" key to Add2daLine() and Copy2daLine()
                 which allows skipping adding lines if a line with the same
                 value in that specified column already exist in the 2DA file.

2005-06-07   Added LABEL as possible index for Change2daLine().

2005-06-07   Added Log summary of encountered Warnings and Errors.

2005-06-07   Added date to start of log, and Install/Patch aware message.

2005-06-07   Progress log now saved to installlog.rtf when completed.

2005-06-07   ".\" folder now reported as "Game" folder in log for InstallList.

2005-06-09   Added support for overwriting HACK and INSTALL files.

2005-06-09   Added "ReplaceFile" key to HACK file modifier list.

2005-06-09   Added "Replace#" key to InstallList file list.

2005-06-09   Fixed user-selected files in InstallMode, now copied to Override
                 before being modified.

2005-06-10   Changed behavior for ExclusiveColumn flagged Add/Copy row modifiers.
                 If row already exists, it now transforms the modifier into
                 a ChangeRow modifier instead, updating the found matching line.

2005-06-10   Put in a simple filter to not allow the File Installer to overwrite the
                 game EXE files or the dialog.tlk file directly.

6. Special Thanks

A very big thanks to **tk102** (of the starwarsknights.com forums) for providing a modified version of the nwnnsscomp.exe script compiler (originally created by Edward T. Smith (Torlack) for the *Neverwinter Nights* game) that has been adapted to better work with the TSLPatcher.

Special thanks to everyone on the StarWarsKnights.com forums at LucasForums who have offered ideas, feedback and suggestions about the TSLPatcher. An extra thanks to *Tupac Amaru* (the forum member) for helping me test the TSLPatcher and its helper applications to iron out the most serious bugs. Big thanks!

And finally thanks to YOU, for enduring my English and reading all the way down here. :P