

RTMUI localization reference v1.2

1. INTRODUCTION	2
2. CREATING LOCALIZATION PACK.....	2
3. LOCALIZING UI.....	2
3.1. CHOOSING UI LOCALIZATION METHOD	2
3.2. LOCALIZING UI VIA RESOURCE LIBRARY	3
3.3. LOCALIZING UI VIA TRANSLATION DATABASE.....	3
4. TRANSLATION GUIDELINES.....	4
5. LOCALIZING EXTERNAL HELP FILES.....	5
6. UPDATING LOCALIZATION PACK.....	5

1. Introduction

Multilanguage system in all RivaTuner technology based products ([RivaTunerStatisticsServer](#), [D3DOverride](#), [EVGA Precision](#) or [MSI Afterburner](#)) is powered by RTMUI - original RivaTuner's user extendable localization engine, allowing third parties to create their own localization packs translating the application to other languages without modifying the application. If you can create localization pack for your native language and wish to share your work with other users, please follow this guide. Localization engine allows you to translate the following components of the application:

- Non-skinned user Interface (i.e. all advanced properties windows, messages, tooltips and menus displayed by application).
- External context sensitive help files (i.e. floating tooltip windows displayed by application's help system when you hover mouse cursor over both skinned and non-skinned controls).

However, it is completely up to you to choose a set of translated components to be included in your localization pack. Localization engine is flexible and it will not fail if some of these components are missing. For example, if you will not include localized context sensitive help system in your localization pack, localization engine will not fail and will simply use default (English) language for it.

2. Creating localization pack

Each localization pack is represented by a separate subfolder in `.\Localization` subfolder in the main application folder. You are free to choose any unique name for your localization pack subfolder and localization engine doesn't put any restrictions on it, however it is recommended to use standard names uniquely identifying your localization pack's language in order to minimize the risk of interfering with other third party localization packs.

Once you've chosen a name for your localization pack's subfolder and created it, you have to create localization pack description file. The file is called **Description** and it must be located in the root of your localization pack folder. This file contains full name of the language represented by your localization pack, which will be displayed in the list of available languages in application settings. The file may also optionally contain your full name and your contact info (email address and ICQ UIN), if you wish to make it public. If you specify this information in the description, application may display it in the information window when user selects a language represented by your localization pack. This way the users can contact you easily and report about typos, if they find any.

Description file format is based upon generic `.ini` file containing single **[Info]** section and it may contain the following fields:

Language field specifies language name.

Icon field is optional, it may specify filename of the icon image to be displayed next to the language name.

Creator field is optional, and it may specify full name of language pack creator.

CreatorEmail field is optional, and it may specify language pack creator's email address.

CreatorICQ field is optional, and it may specify language pack creator's ICQ UIN.

For example, localization description file may contain the following fields:

[Info]

Language = Russian

Icon = RUS.ico

Creator = Alexey Nicolaychuk aka Unwinder

CreatorEmail = AlexUnwinder@mail.ru

CreatorICQ = 64116381

Once you've created localization pack description file by specifying non-empty language name, you may restart application and you will see that your language has been added to `<User interface>` tab in advanced application properties.

3. Localizing UI

3.1. Choosing UI localization method

The first thing you've to do before starting UI localization is to choose your preferred localization method. Localization engine gives you maximum freedom and allows you to choose one of the following localization methods:

- Traditional resource library based localization, which suits best for nontraditional UNICODE only languages. If custom Resource.dll library is included in your localization pack, application can load all the resources from it instead of the executable file. Choose this way if your language cannot be represented by ANSI encoding or if you're already familiar with resource library based localization approach. You'll need additional resource library editing tools (for example Microsoft Visual Studio), if you choose this method.
- Translation database based localization, which suits best for traditional ANSI languages. Besides traditional resource library based localization, many multilanguage products also use so called language lookups approach, allowing multilanguage software to retrieve localized string from language lookup file by specified string or numeric identifier. This approach requires software developer to support each localizable string in the code, which greatly increases the risk of missing some string during localization,

makes language lookup depending on identifiers used by software developer. This makes language lookups hard for editing and understanding without assistance of original software developer. Opposing to this approach, RivaTuner's localization engine introduces unique and completely original runtime translation concepts, which lacks all these drawbacks. The engine offers new concepts, based on idea of automated contents based translation of any string displayed in any of application windows. Translation is performed via so called translation databases, which can be included in localization pack and which are telling the application how to translate each required phrase to a target language. It is strongly recommended to choose this method if your language fits in ANSI encoding, because translation database based language pack is drastically smaller than a pack containing localized resource library. Furthermore, you don't need any additional tools besides a file manager and plain text editor (e.g. Far) to create translation database based localization.

3.2. Localizing UI via resource library

Note: skip this chapter if you've decided to use translation database based UI localization.

Once localization pack folder is created, you may add resource library to it. You may use **/MR** command line switch in order to create resource library containing copy of the application's resources. For example, if you are creating Russian localization pack for MSI Afterburner and your localization pack is stored into **.\Localization\RUS** subfolder, then you may create the resource library by executing the following command in the application root folder:

```
MSIAfterburner.exe /MR Localization\RUS\Resource.dll
```

After creating new resource library, you may simply open it in any resource editor (e.g. Microsoft Visual Studio) and translate all the dialogs, strings and menus. Once you've translated at least one resource in this library, you may save your changes and restart application to verify that you're on the right way. If your localization pack is selected in the application properties, you'll see that application uses localized resources now.

3.3. Localizing UI via translation database

Note: skip this chapter if you've decided to use resource library based UI localization.

Translation database tells the application how to translate each phrase displayed in any of application windows to localization pack's language. The database is stored in **.\Translation** subfolder of your localization pack's folder. Localization engine creates the database by merging contents of all files found in all subfolders of your language folder's **.\Translation** subfolder. It means, that you may store a database as a single file located at root of the **.\Translation** subfolder, as well as in many independent files stored in its subfolders. Such approach makes the database more structured as well as allows appending it by adding new files instead of overwriting whole database. Database files are plain text files, containing the list of lines starting from **#src**, **#dst**, **#end**, **#hst** or **#dlu** tokens. If a line starts from any other character sequence, it is ignored by database loader. The meaning of database tokens is explained below:

#src <phrase> token is used to define a line describing source text of translated phrase. For example, **#src Cancel** line tells localization engine that we're going to translate **Cancel** to other language.

#dst <phrase> token is used to define a line describing destination text of translated phrase. In other words, it tells localization engine how to translate a source text, which we've previously specified with **#src** token.

#end token must be specified to finalize each translation database entry. This token tells localization entry that we've finished specifying translation database entry parameters and it is ok to add it to database.

These are the basic tokens, allowing you to perform 99% of translation. However, in some cases you may need additional tokens:

#dlu <width> [<height>] token is used to force localization engine to resize a host control associated with translated phrase, and it specifies new width and height of the host in dialog units. For example, you may use it to resize a button if translated text doesn't fit in it. You may specify only one parameter to change host control height only, or set the first parameter to zero to change host height only. You may also specify both positive and negative values to resize a control using fixed left/top (positive) or right/bottom (negative) host coordinates.

#hst <class> [<ID>] token is used to make a translation specific only to a specified class of host controls (e.g. only to buttons) or even only to one control specified by ID. For example, **#hst Button** line tells localization engine that the current translation must be applied only to Buttons, and **#hst Button 1200** line tells localization engine that the current translation must be applied only to the button with ID 1200. If you don't specify the token, the translation is allowed to be applied to all hosts of all classes. If you specify multiple translation of the same phrase for different host classes and ID, localization engine uses the following rule to resolve a translation:

1. Try to use translation specified exactly for current host class and current host ID
2. Try to use translation specified for current host class
3. Use common translation

For example, translation database may contain the following entries:

```
#src On-Screen Display  
#dst ОЭД  
#end ; tell localization engine to translate "On-Screen Display" as "ОЭД"  
...  
#src More
```

```
#dst Дополнительно
#dlu -100
#end
```

; tell localization engine to translate "More" as "Дополнительно" and to resize the button containing this text to 100 dialog units and to align it by the right edge

Once you've added at least one entry to the database, you may save your changes and restart the application to verify that you're on the right way. If your localization pack is selected in the application properties, you'll see that application uses localized resources now.

4. Translation guidelines

Please follow these guidelines when translating the user interface:

- Don't be afraid to combine resource library based and translation database based localization methods if necessary. The application first reads localized resource from resource library then tries to translate it via the database. Knowing this rule, you may always use resource library as your localization pack base and translation database for translating the phrases, which cannot be covered by resource library (for example, the names of all hardware monitoring module graphs can be translated via the database only) and retranslating some phrases if necessary.
- If you're using translation database based method, perform the translation by small portions, for example window by window and check translated UI in action after finishing each portion. Don't forget that the length of phrases differs for different languages, so don't forget to resize the controls with **#dlu** token if necessary.
- If you're using translation database based method, you may use built-in localization logging system for debugging by setting bit 1 of **LocalizationDebugFlags** configuration file entry to 1. When logging system is enabled, localization engine writes translation database initialization log and all requests to translation database and all responses from it into **.\Localization\Localization.log** file. Examining it you may see which translations are coming from each translation file, easily track the warnings from the localization engine (e.g. if you specify different transition for the same phrase in multiple files in the same translation database etc), see classes and IDs of all controls requesting translation, see which phrases lack translation and get other useful information. **LocalizationDebugFlags** is a bitmask so besides simply enabling the logging you can also use the following debug flags:

0x00000001 - Enable logging system

0x00000002 - Don't log correct entries

If this bit is set, logging system will not log correct entries when composing the database from the file system during application startup. Such logged information allows you to identify the location of each translation into each translation database file.

0x00000004 - Don't log skipped entries

If this bit is set, logging system will not log skipped entries when composing the database from the file system during application startup. The entries can be skipped if you try to specify more than one version of translation for the same phrase. Such logged information allows you to identify such situations and to see if specify corrosslinking translations.

0x00000008 - Don't log success requests

If this bit is set, logging system will not log all successful requests to localization engine. Such logged information allows you to see class and ID of each host control requesting and successfully receiving the translation.

0x00000010 - Don't log success request source file info

If this bit is set, logging system will not log source file info for all successful requests to localization engine. Such logged information allows you to location (i.e. source file name and line number) of each successfully received translation.

0x00000020 - Don't log failure requests

If this bit is set, logging system will not log all failed requests to localization engine. Such logged information allows you to see which translations are missing.

0x00000040 - Highlight resized hosts

If this bit is set, localization system will highlight all controls resized with #dlu token via the translation database.

0x00000080 - Show host info

If this bit is set, localization system will add info about each host control to floating tooltip (control class, ID and size in dialog units).

- Some translated phrases are not displayed explicitly and contain so called format specifiers (e.g. **%s**, **%d**, **%x** etc), which are used to format different strings using predefined list of variables of different types (e.g. **Logging to %s (%164u bytes)**). Please remember that the order of format specifiers is critical and you are not allowed to change it! Otherwise it may cause unpredictable results up to software crashing. For example, you cannot replace this string with **%164u bytes have been logged to %s file**. The only thing you are allowed to do with format specifiers is removing last (and only last) specifier(s) from the string. So it can be translated as **Logging to %s**, but cannot be translated as **Logging to a file (%164u bytes)**.
- Some phrases and words cannot be translated by design of application. For example, you won't be able to translate units of values displayed by [EVGA Precision](#) or [MSI Afterburner](#) hardware monitoring module in the On-Screen Display.

5. Localizing external help files

Localization engine uses file system redirection idea to localize external help files. It means that when application displays any external help file (e.g. floating tooltip stored into `.\Help\BUTTON_APPLY` and displayed by application help system when you hover the cursor over **<Apply>** button in the main window), localization engine try to redirect access to this file in the root application folder to a file with the same relative path stored into you localization pack's folder (for example, `.\Localization\RUS\Help\BUTTON_APPLY` if Russian localization pack is selected). Due to this approach, you may localize external help files by copying root folder's `.\Help` subfolder to your localization pack folder then translating its contents one by one with any text editor.

6. Updating localization pack

After releasing new version of RTMUI based multilanguage application you'll need to update your localization pack to make it compatible with it. Normally, you'll have to perform the following routines during updating your localization pack:

1. Find new file system based translated components (e.g. context help files) and add the translations to your localization pack.
2. Find and remove obsolete file system based translated components (e.g. obsolete context help files, which no longer exist in new version).
3. Find and update changed file system based translated components (e.g. context help files may contain new hints, so it must be reflected in the translation too).

If you're using translation database based UI localization approach, you'll also need:

1. Find new or changed translation database entries and reflect the changes in your translation database.
2. Find and remove obsolete translation database entries from your translation database.

RivaTuner's localization engine allows you to simplify these routines with specially designed localization pack comparison mode, which is aimed to detect all types of changes mentioned above by means of comparing two localization packs. It is recommended to compare your localization pack with original Russian localization pack, which is included in the distributive as a sample for third party localization pack creators and which is always up to date. Localization pack comparison mode can be activated by executing the following command in the application root folder:

EVGAPrecision.exe /CL <PackID>

The command compares localization pack, which is currently selected in the application properties, with another localization pack, specified by **<PackID>** and stores differences list into `.\Localization\LocalizationDifferences.log` file. So you'll just need to compare localization pack you're about to update with sample Russian localization by executing the following command in the root application folder, for example:

EVGAPrecision.exe /CL RUS

After executing this command application will generate `.\Localization\LocalizationDifferences.log` report files, containing the info you need to analyze. Differences report analysis is very simple: it is tabbed text file containing the sections starting with the following headlines:

Searching for changed translated files - precedes the list of file system based translated components you'll need to change in your localization pack.

Searching for ... pack translated files missing in RUS pack - precedes the list of obsolete file system based translated components you'll need to remove from your localization pack.

Searching for RUS pack translated files missing in ... pack - precedes the list of new file system based translated components you'll need to add to your localization pack.

Searching for ... pack translation database entries missing in RUS pack - precedes the list of obsolete or updated translation database entries you'll need to remove from your localization pack. You'll see database component filename and line number for each entry.

Searching for RUS pack translation database entries missing in ... pack - precedes the list of new or updated translation database entries you'll need to add to your localization pack. You'll see database component filename and line number for each entry.