Introduction

This document provides detailed information/specifications/requirements on how to design and prepare flash web games for RealGames' post-production department.

Our post-production department handles localization and preparation of all localized builds which are launched (both downloadable- and web games). Our goal is to have a predictable process for the localization of those games and be able to provide a short timeline from delivery to launch. To achieve this we have come up with requirements and specifications in order to guide this process. This document explains, in detail, which requirements we have for Flash web games in order to make them fit into our process.

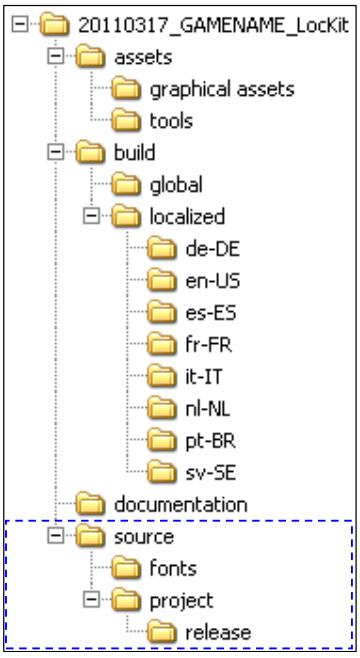
1 Delivery (localization kit)

When delivering games to us, it's highly preferred to provide us with a final multilingual build which is localization-ready. This way we separate the development and localization part which is the most efficient way to work.

We need the final build to be accompanied by some other assets which we require for localization such as graphical source files with separate text layers for easy modification. To shorten lead times, include as many and all kinds of materials which you believe can be useful to us for the localization process; without having to rely on your resources.

By using a logical file structure we can easily find all the materials that are required for localization. The directory tree on the right is a preferred structure of how a typical game should be delivered to us.

A checklist can be found at the end of this document which should help you to prepare all of the assets we need. The combination of assets and game files required for localization is often referred to as **localization kit** or **lockit**.



------ = optional

The lockit structure explained

▪ Assets

Graphical Assets

This folder should contain source files to edit images which need localization. PSD files with TTF fonts are ideal. More information can be found in our general localization requirements document: document_localizationspecifications_v2.1.pdf.

Tools

Generally flash web games do not require tools to be localized nor for asset packs to be rebuilt. In case your game does, please put those in this folder together with extensive usage documentation. Tools should preferably have a command line interface to enhance automation options on our end.

▪ Build

This folder contains the final multilingual build of the game; generally an SWF file and some config files. Furthermore this folder also contains two important subfolders:

- **Global** contains only general resource files. These files should not contain files such as images or XML files with text which needs localization. Only store general images, config files, splash screens etc in this folder.
- **Localized** with a subfolder per language that only contains language specific files. E.g.: localized images, translated text files, etc. You can also place gui-layout files in here so we can easily make changes to the game's layout for a specific language (for example: shrink the position or font size of text fields). This is NOT the place for entire builds. If the game has not been localized into any languages yet, these folders (except en-US) will be empty.

Make sure this build is clean (e.g. does not contain old files, temporary files, versioning files(svn/cvs) or other unnecessary files such as thumbs.db files (windows) or .DS_Store(mac)).

▪ Documentation

This folder should contain important information that we can use during localization. Cheats, information about which files need to be translated, usage of scripts and so on.

▪ Source

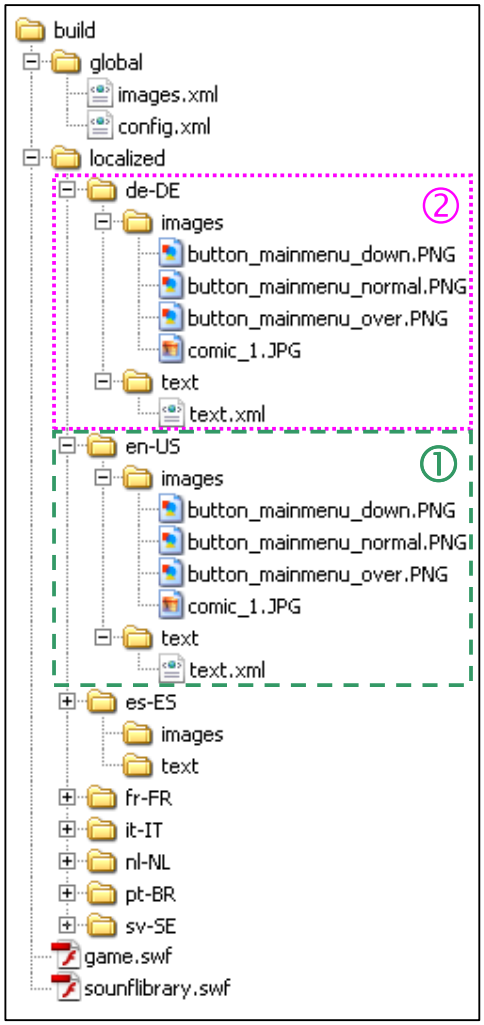
In general we will not need the source code for a web game since a localization-ready build will be enough to serve localization purposes. If we however agree future changes need to be made by us, source will come in very handy. Make sure to include ALL required source code, assets, libraries, fonts and images to rebuild the game. The source should build the release build' main SWF file(s) as the one(s) in the build folder. Make sure the source has a designated and recognizable folder in which the project compiles a release build.

2 Requirements

To be able to localize your game, as smooth and quickly as possible, we have set up some requirements. We ordered those in four subchapters. The first subchapter talks about the way a multilingual build should be structured, the second subchapter describes localization requirements (specifically for web games), the third subchapter talks about technical requirements and the fourth subchapter describes policies that we have for web games.

2.1 Multilingual build structure

Because flash web games will be localized into multiple languages we require a multilingual build which uses a clear but most of all extensible file/folder structure. The root of this structure simply contains your main SWF file.



<global>
The global folder can contain general resources and XML configuration files. Library SWF files can also be stored here.

<localized>
All language specific files should be stored in a language specific folder inside the 'localized' folder. This folder contains various 'language-country' specific folders whose names are based on the various languages and countries we localize in.

Each subfolder contains language specific files like localized images, translated text files, etc. You can also place gui-layout files in here so we can easily make changes to the game's layout for a specific language (for example: shrink the position or font size of text fields). If the game has not been localized into any languages yet all folders, except en-US, will be empty.

By using this structure we can easily group all localized resources for a game together and clearly separate them from other languages. In the example directory layout on the left you can see:

①. This folder contains externalized **English** resources.

②. This folder contains same resources as the en-US folder, but these contain **German** texts.

When a web game is loaded, the 'language' and 'country' flashVars should be used to select the right resource directory to load its localized resources from. *For more info see chapter "flashvars".*

! It is recommended to store various resources using a 'type by language' structure. This means store all images together, all images together etc. This way all resources can be located easily.

2.2 Localization requirements

All text in the game should be externalized so we can easily access them without having to recompile the game. This means all menu-item-texts, button-texts, labels, (dialog/hint)-messages, storyline, comics but also object- and item-names in hidden-object games. This can be done by storing text in an external XML file or by having external images. These external localized resources should be loaded by the game during the loading-phase. This way 1 SWF can easily load all localized resources without having any hardcoded text/images with text.

The most common problems which we run into when localizing can be found in our general localization requirements document: document_localizationspecifications_v2.1.pdf.

Web games however have different characteristics and therefore additional requirements which we would like you to know about. This chapter contains and explains specifications on how we like to see those external resources to be stored for flash web games.

2.2.1 Externalize text

All in-game text should be stored in an external text file (preferably valid XML). This allows us to simply change text without having to recompile a game. This means all menu-item-texts, button-texts, labels, (dialog/hint)-messages, storyline, comics but also object- and item-names in hidden-object games should be externalized.

Make sure to save the XML file as UTF-8 encoded to enable Flash to correctly read all accented Latin-1 characters like “é”, “É”, “ß” etc. Flash comes with a good XML parser which is capable of loading and navigating through an XML.

2.2.2 Reuse texts from the downloadable version

We strongly prefer to reuse the English text files and images from the downloadable version for the web game literally where possible! This way we can easily reuse the translated strings and images from the localized downloadable versions to create localized web-versions which in turn reduces translation costs and overall lead-time as we already have the translations for those texts.

! Make sure reused images are exactly the same as used in the downloadable version and are scaled with constrained proportions to ensure we will be able to simply rescale localized images from the downloadable version to the web game’ image dimensions.

! New texts which are introduced in the web version should be added to the text xml (preferably in a separate section) or localized images folder so we can easily have those translated.

2.2.3 Externalize images

Sometimes certain graphical effects on texts can only be achieved by having text in an image (instead of externalized text in a dynamic label). By using externalized images we can simply change the PNG/JPG image file in the ‘localized’ directory. External images should be preloaded dynamically from the correct “`localized/%LANGUAGE%-%COUNTRYCODE%/images`” directory using the provided language and country flashvars.

If images don’t contain text and therefore do not need to be localized, just keep them inside the main SWF or a general resource SWF.

2.2.4 Embedding fonts

Because of Flash' behavior regarding fonts ALL dynamic text fields which show text have to be embedded to make sure the game displays all text correctly. By embedding fonts in a label a subset of a font' characters outlines is embedded into your game SWF making it independent of external fonts which may be missing on the end-user' system. The more characters are embedded the larger the SWF becomes and therefore we have composed a set of characters which should be embedded to ensure correct displaying of all required localized texts.

Always make sure to embed the following set of characters:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789 ~`!?¡¿º@#$%^&*()-_=+[{]};:'",<.>/¢£€¥§©®™\ ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖŒÙÚÛÜÝŸßàáâãäåæçèéêëìíîïðñòóôõöœùúûüýÿ

2.2.5 Bitmap-fonts

Avoid using bitmapfonts. They are hard to edit/update and often make the game unnecessarily bigger in comparison to when using normal fonts. Moreover bitmapfont sizes cannot be changed as easily as normal labels and will require a lot more code to be displayed (correctly). The extra work involved when using bitmapfonts it is often not worth the extra eyecandy they provide.
Flash features filters to enhance the visuals of text labels which can help you to realise certain effects while still having the flexibility of dynamic text.

2.2.6 Cheats

To speed up testing of web games we want all web games to have a decent set of cheats allowing our localization and QA department to instantly achieve certain things in a game or to be able to quickly jump to specific parts of the game.

The cheats we minimally (if applicable) want to be available are:
- Win level/round;
- Lose level/round;
- Win game;
- Lose game.

In addition we would love to have these cheats:
- Jump to levels (next and/or previous level);
- Achieve any special achievement in the game (un-lockable items, bonuses, etc);
- Add money.

Make sure cheats can be switched on and off easily. See chapter “flashVars” on how to enable/disable cheats.

2.3 Technical requirements

To make sure a Flash web game runs in our environment, is easy to localize and can easily be modified in case we run into issues, we have some requirements regarding the delivered build.

2.3.1 API

We require our Flash web game API to be implemented in all Flash web games. This enables high score-support, dynamic game breaks, score broadcasts and dynamic 'download-button' actions. Please read the API implementation document for in-depth information on the API: (web flash) API implementation - 1.2.pdf.

2.3.2 Flashvars

To control a game's behavior we provide flashVars which should be used by the game. These flashVars are provided through our API. See the API implementation document on how to acquire the flashVars.

These can easily be used by the flash-games allowing them to change certain aspects of the game such as:
- the loaded resource files' language;
- whether or not the show the download button;
- whether or not to show interstitials/advertisement screens.

The following flashvars need to be implemented in the web games:

language	**Purpose:** The language and country variables together make up the string code which corresponds with the directory names in the localized directory where all localized resources reside.
	Example values: "de", "en", "es", "fr", "it", "nl", "pt", "sv"

country	**Purpose:** The language and country variables together make up the string code which corresponds with the directory names in the localized directory where all localized resources reside.
	Example values: "DE", "US", "ES", "FR", "IT", "NL", "BR", "SE"

When loading resources combine the language and country parameters to make path-strings which looks like this:

localized / %language%-%country% / images / %fileToLoad%

This way we can easily change the loaded language at runtime without the need for a separate build per language.

showAds	**Purpose:** Whether or not to show the upsell dialogs (both in-between levels as well as after the game is completed/over). *For more info see chapter: Upsell*
	Example values: true / false

hasDownloadable	**Purpose:** Whether or not to show the 'Download' buttons throughout the entire game. *For more info see chapter: Upsell*
	Example values: true / false

cheatsEnabled	**Purpose:** Whether in-game cheats are enabled. This enables our QA department to (de)activate the built-in cheats enabling them to quickly test certain aspects of the game without the need to rebuild it.
	Example values: false / true

2.3.3 Build size

Try to keep the size of web games under ~20MB. You can achieve this by not implementing all features found in the downloadable version of the game and/or by lowering the encoding quality of used jpeg images (no severe visual degradation like jpeg artifacts should be visible). Disabling certain features can be 'covered' by informing the player about this feature being available in the downloadable version.

2.3.4 Dimensions

Our portals allow Flash web games up to 640x480 pixels to be displayed without scaling. Larger games (800x600 for example) are scaled down by the object tag in the website until both width and height are exactly equal or smaller than 640 and 480.
Smaller games are shown in their exact dimensions. It is however advised to NOT create web games with dimensions smaller than 640x480. Especially hidden-object-games (HOGs) benefit from larger dimensions as some items can otherwise become hard to distinguish.

2.3.5 Player version & ActionScript

We prefer to see web games made for Flash player 9 or 10. Games built for these player versions offer really great performance compared to previous versions allowing for a better user experience. This still allows developers to use either Actionscript 2 or 3.

Our portals currently require our players to have Flash Player 10.0 installed on their systems. If your game requires a newer version of the Flash player to be installed on the players' system please let us know.

2.4 Policies

To meet our channels' demands we have some policies web games should adhere to. If for some reason you really do or do not need or want the features we discuss in this chapter simply implement a configurable switch in a config.xml file which can be flipped to enable/disable the feature.

2.4.1 Branding

Web games can display splash screens before the game starts. Do not display branding images / logo's after the splash screens. If you really want main menu branding, please make sure we can switch it off using a config file. Splash screens must not have links to external URLs or non-clickable URL information (like tooltips).

2.4.2 Player profiles

Do NOT implement player profiles in your web game. This also means you should not ask players to enter their name nor try to resolve the players' name or computer name in any other way.

2.4.3 Savegames

We really like to see save game functionality in all web games. The preferred way to implement save games is to have the players' progress saved every level they complete. Because save games are not very common in web games yet we want to inform our players that their progress is saved, and that they can continue playing where they left off next time they play the game. The easiest way is to, in a level-complete dialog, say something like "your progress has been saved". If you implement popup tips you can also inform players their progress is saved that way.

Serializing of game objects in .sol files using the regular serialization functions available in Flash is the easiest way to store save games safely on the players' computer. Do not use undocumented features to store save-games on a players' system.

2.4.4 Game finished / game over

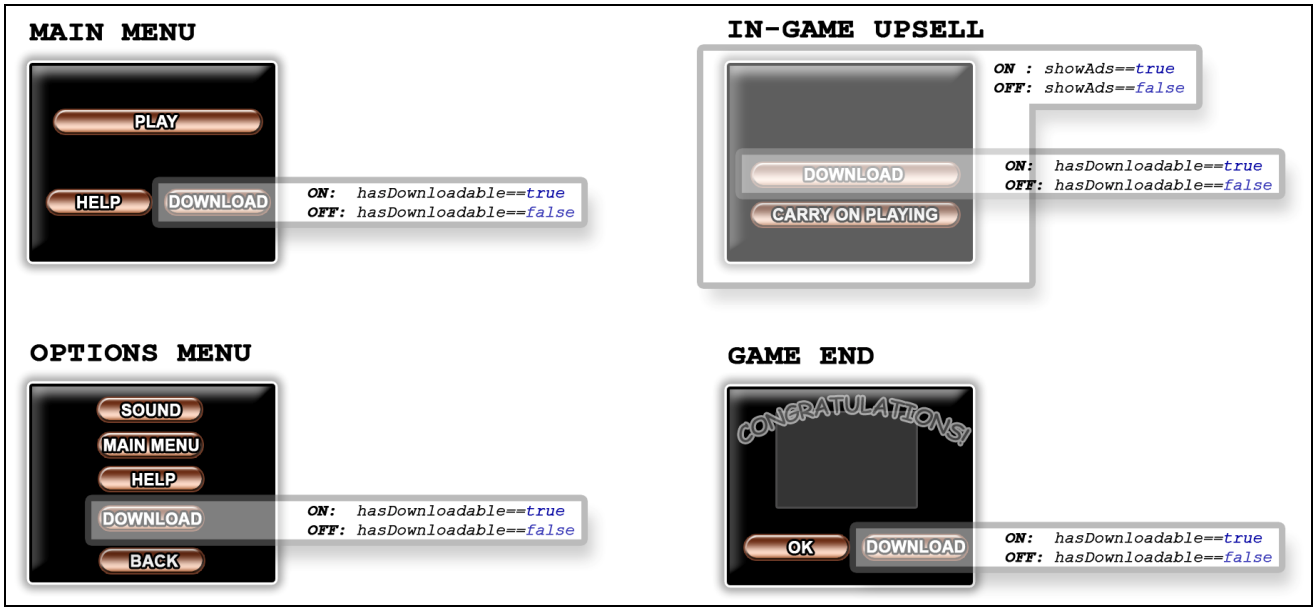
When a player finishes the game or upon game over, the game should show a dialog notifying them about this. If the player reached the end of the game, have a header congratulating them with their achievement or tell them, for example, "You've reached the end of the journey offered in the web version of GameX".

If the web game features a competitive scoring system (score based on skill, time, precision) please show their final score on this dialog as well.

2.4.5 Upsell

Web games often have upsell dialogs to:
- upsell the downloadable version of the web game (regular upsell);
- upsell other downloadable games from a developer ('More games' upsell)

All web games that we launch should have **configurable** upsell screens. We use a dynamic gamebreak system; enabled by our API. This system allows us to show dynamic content inbetween levels instead of hardcoded upsell screens. This means we must be able to turn the upsell screens off using the showAds flashVar. For the channels where we do use the built in upsell dialogs we want to have the option do disable the download button which is often featured on such dialogs. This means the hasDownloadable flashvar should be used to determine whether or not to display the download buttons throughout the game.



3 Source code

This chapter talks about things related to source code for Flash web games. Please note that we prefer receiving a final build of the game with ways to modify the game's behavior so we won't have to recompile the game.

3.1 Adobe Flash IDE

When using the Adobe Flash developer environment, try not to put source code all over the place inside the FLA files. Instead put code in .AS files as much as possible. We rather have multiple AS files than a mix of AS code in buttons, MovieClips and in frames. Coding in AS files is a good help in generating well structured code and is more easily searchable than source code inside a FLA file. Moreover AS files can be compared using comparison software, while binary FLA files cannot!

3.2 IDE & Tools

To allow for workable build processes we require that source can be compiled using one of the following systems:
- Adobe Flash CS5, CS4, CS3;
- Flex 3 (Do not use Flex-projects, use AS-projects instead!);
- FlashDevelop.

Please make sure to include a project file for Flex 3 and FlashDevelop and clear documentation on how to compile your project if it requires special compilation-/build-steps and/or special libraries.

We cannot accept custom build-&pack-tools for Flash web games. If all resources are externalized there's no need to recompile the game/resource files. If you really need such a custom build-process please let us know as soon as possible.

3.3 Main Game SWF

There should only be 1 main game SWF file (having a loader SWF is no problem). This SWF should not contain any text items/images which need to be translated. Localizing text labels, variables and images inside an SWF is very time consuming so make sure all localization-items are stored in the "`localized/%LANGUAGE%-%COUNTRYCODE%/`" directory to allow for easy localization/modification.

Based on the language and country flashvars provided to the game, by the website hosting the game, it will look for text and image resources in a specific resource subdirectory in the 'localized' folder. Adding new languages this way is easy as we will only have to add a directory with new localized resources and send the new language & country flashvars to the game.

See chapter "Flashvars" for a more detailed explanation.

3.4 Resource SWF files

When making flash web games developers often choose to store sounds or similar resources in a separate SWF (to keep the publish times for the main game-SWF short). It is allowed to have resource SWF files as long as they are preloaded by the game to ensure al resources contained in it are loaded before going to the main-menu. Make sure they contain no text or image resources with text. Sound resources consisting of speech-samples are not a problem as they won't be localized.

A loading sequence is perfect for loading resources and together with a (game-themed) loading bar and a "**loading…**" message look nice for the player while they are waiting for it to load. Having no message & progress-bar during loading is not recommended as players might think the game stopped working/doesn't load when they're on a low-bandwidth connection. Please make sure the loading text is externalized so it can be localized.

3.5 Coding Pitfalls

Reading XML files
When loading/parsing XML files make sure to read the actual value of a node (”**.nodeValue**”), instead of just the node; asking the XML document for its value ensures the correct reading of special characters.

Levels (ActionScript 2)
Because all flash web games will be loaded by a loader, it is crucial they can run from Flash’ _level1 (instead of _level0). This means the game itself may not be _level0 variable dependent. Make sure to use “**_root**”, “**this**” and “**_global**” and use “**this._lockroot = true**” if needed. These should enable you to do everything you want without having to hardcode variables into specific levels.

Sound
Developers also often use the following or similar code to create sounds:

```
var mysnd:Sound = new Sound();
```

Instead try to make a new sound providing ‘**this**’ as a parameter to the Sound-constructor so the game will be able to correctly locate sounds when loaded by a loader. Like this:

```
var mysnd:Sound = new Sound(this);
```

Sound cards (ActionScript 3)
Make sure the game is capable of ‘handling’ computers which do not have a sound-card. This will cause problems when initializing Sound components in AS3 because Win 7/Vista pc’s which have no headset/speakers connected will often report no soundcard is present.

Stage (ActionScript 3)
Web games made in AS3 which are loaded by our loader SWF can run into problems when trying to access the ‘**Stage**’. This is caused by the fact that they haven’t been added to it yet when they start loading. Please make sure to implement an **ADDED_TO_STAGE** listener in your game to check whether or not your game(-loader) has been added to Flash its Stage.

4 Checklist

Short recap of requirements

Build	Yes / No / Comment / Details
Is the game a multilingual build which conforms to the suggested file/folder structure?	
Localization	**Yes / No / Comment / Details**
1. Are all game texts externalized? 2. Are the texts from the downloadable version re-used?	
1. Does text still fit if it becomes 30% larger after translation? 2. Is there a character limit for certain words?	
Is text adjustable or scalable to make longer translations fit? Can the font be changed in a gui config file per language?	
No hard-coded text in source code?	
No stitching of words in the language file? If so, what is stitched?	
1. Are all images which contains text externalized? 2. Are images from the downloadable version re-used and scaled to web-size dimensions?	
If images cannor be scaled from the downloadable version, are there layered psd files for each image which contains text?	
Do all fonts required forAccented characters (Latin-1) in ALL fonts, including TTF files for PSDs?	
Has the game already been localized in some languages?	
Is there an exclusivity period for already localized languages? If so, how long and when does it expire?	
Is there a localization manual document?	
Cheats that allow moving through the game quickly?	
Are regional settings configurable and not hard-coded? (Currency, thousand seperator, etc)	
Technical	**Yes / No / Comment / Details**
Are all required flashVars implemented?	
Is the RealGames API fully implemented?	
Are all other API's switched off?	
Policies	**Yes / No / Comment / Details**
Branding screens are configurable?	
Player Profiles disabled / not present?	
Does the game feature savegames?	
Can upsell dialogs be switched off through the flashVars?	
Misc	**Yes / No / Comment / Details**
All tools included (packing/encrypting/building)	
No files that aren't needed in the game folder? (e.g. thumbs.db)	

5 Reference list

More localization information from GameHouse
https://partners.gamehouse.com/localization/

ISO/IEC 8859-1 (Latin-1) character encoding
http://en.wikipedia.org/wiki/ISO/IEC_8859-1

Cyrillic alphabet
http://en.wikipedia.org/wiki/Cyrillic_alphabet

UTF-8 character encoding
http://en.wikipedia.org/wiki/UTF-8

XML standard
http://www.w3.org/TR/2008/REC-xml-20081126/