1-click upgrade module

Notes taken during rework

These notes have been taken during the refacto on the autoupgrade module. They are written for the latest version, but are also mostly compatible with the previous ones.

> Entrypoints

The 1-click upgrade wants to be a module AND be as independent as possible. Which means its content can be reached from several way:

autoupgrade.php

Every module developer will recognize its content. This file is used to manage the module part of the project: installation, uninstallation, then automatic redirection to the next file.

AdminSelfUpgrade.php

This controller uses the AdminController features from PrestaShop. It allows the upgrade configuration to be displayed in the back office. The merchant will see the configuration options.

It was used to handle everything, displaying answers, handle and dispatching requests, making coffee etc. In the latest version, we aim to limit its responsibilities, which should be displaying the configuration page.

ajax-upgradetab.php

This file is called from ajax requests. This is the only responsible of the initialization and upgrade / rollback steps. This file is not a PrestaShop controller for a simple but important reason, we do not want to rely on the core during an upgrade or a rollback, this would increase the risk of crashes.

cli-upgrade.php

This is the equivalent of ajax-upgradetab.php file for CLI calls. It will instantiate some specific features to the CLI, like a logger displaying informations on the fly. This entrypoint is also useful for testing, or for a user who does not want to use the web version.

> What version to use

The objective is to have a single module version to handle these PrestaShop upgrades:

- 1.6 >> to >> 1.6 / 1.7
- 1.7 >> to >> 1.7

The other versions are not part of our support plan anymore. We recommend to use the previous versions of the module available on the PrestaShop marketplace.

> Technical choices

Compatibility with PrestaShop 1.6 & 1.7

We took the versions supported at the moment we wrote this documentation (and the next module versions). We want to help the highest number of merchants to upgrade to the latest major version.

Twig as template engine

Even if Smarty is still provided by the core, we decided to use an independant engine template.

- Replacing the core template engine won't break the module later
- We limit the dependencies between the core / module

Core classes avoided

Because the upgrade will modify classes you might need, we must avoid relying on them.
This is what we would do in a perfect world, but here we can't! Mainly because some features absolutely need the core. What we tried to do is avoiding when possible the core, mainly until the UpgradeDb step, in order to avoid some undefined methods coming from the new files.

The best compromise was to separate requests (even the CLI one) in several steps. By doing so, we can start the new fresh core with its updated classes.
When we published the beta version of the module making an upgrade 1.6 to 1.7, the class Upgrade was introduced to the core (~ 1.7.1.0)

Interface still unfriendly

The perimeter of the new module version only concerned its backend. The first objective was to make the PHP code easier to understand and improve the robustness of the upgrade process where possible. The interface will be updated in another version.

New module structure

By looking at the content in the next module version, you will find a lot of files in the “classes” folder. That was necessary in order to remove all the stuff from AdminSelfUpgrade, the class previously responsible of everything.
We grouped our files in main topics:

- All classes responsible of the display (Twig related)
- Classes used for the module to work
- Classes interacting with PrestaShop core
- Tasks (Can be considered as controllers for upgrade, rollback etc.)

New features

Although the last major version of the module was supposed to be a refacto (meaning we don’t change the module behavior, only its content), we added some cool features at the same time to improve the support.

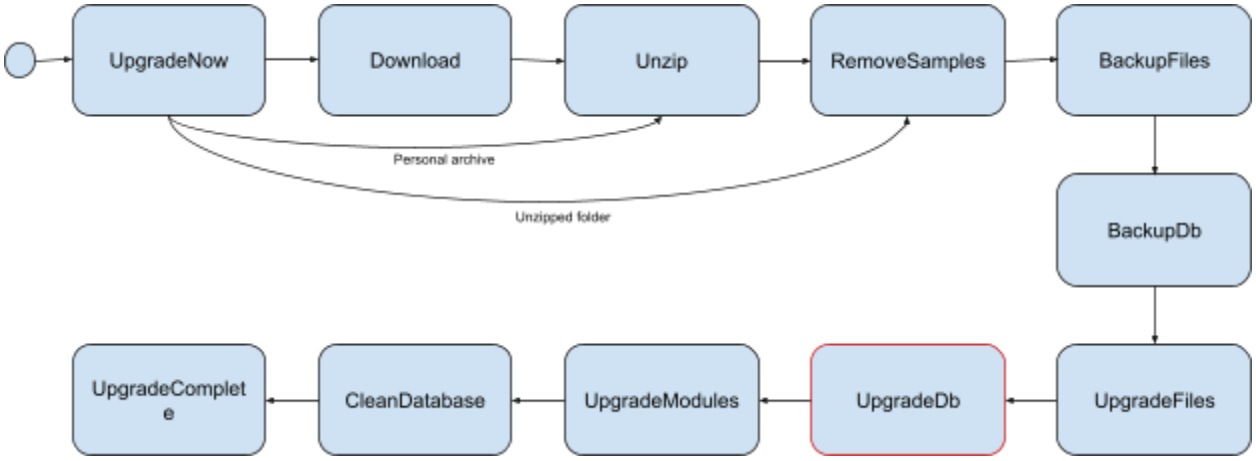
- New loggers: Reporting is now using the PSR-7, allowing incomers in the module code to recognize some common code in PHP. New loggers have been created:
    - LegacyLogger: The existing logger still stores its content in lists before being displayed in a sigle row, and at the same time in a log file. By doing this, we still can get details on what happened if the script execution stopped prematurerly.
    - StreamLogger: New logger, used in CLI mode. We do not need to use memory to store the logs, we can display them directly on the terminal.
- Error handler: Our main issue was the support. In case of HTTP error 500, which means an internal error occured, the module was unable to display anything to

the user. On our support platform, he was requested to open his PHP logs in order to get error details, and this step was difficult for non-tech people. Now, the error is displayed directly on the screen.

> Available steps

All these steps are called from the [entrypoint ajax-upgradetab.php](#).

Upgrade



The following steps will be executed during the upgrade:

- UpgradeNow: Start of the whole process. The next step will be chosen depending on the configuration.
- Download: Download the proper archive depending on the selected channel
- Unzip: Unzip the downloaded archive, what else did you expect?
- RemoveSamples: The changes made by the merchant on his shop, like images, must not be deleted! This step removes the example files from the downloaded archive.
- BackupFiles: In case the upgrade does not went well, or if the merchant wants to rollback later, we save the files of the shop. Note this can be filtered, for example if a file is too big.
- BackupDb: Like the files, we save the current database structure and content, in case a rollback is needed.
- UpgradeFiles: Now the current content is saved, we can alter the shop content. This step will run several time. The first call will initialize the files list, then the next ones will copy a part of this list.
- UpgradeDb: This step does much more than you can think. Its initial purpose is to run all the upgrade SQL files available in the install folder. Then, it will run some additional steps, such as theme enabling, cache deletion, language update...
- UpgradeModules: With the list downloaded from the PrestaShop Marketplace in the first steps, we now request updated module zips and update the installed ones.
- CleanDatabase: This step run some SQL queries in order to remove obsolete or wrong data. Note the concerned data is not coming from the upgrade itself, but from the use of PrestaShop.
- UpgradeComplete: Hey, who would believe it? The upgrade completed and we want you to know it. This step will display a success message and will end the process.

Rollback

- Rollback: This is the entrypoint of the rollback process. It will find all the available backups regarding the given parameters (basically, the restore backup must be sent from the backup name you generated). If a backup matches the given parameters, the process starts the file restoration.
- NoRollbackFound: A classic task used to display a message saying no backup matches the given parameters, and terminates the process.
- RollbackFiles: Like the step UpgradeFiles, this step copies the files from the archive and remove the files absent from the original environment.
- RollbackDb: This task reads and runs the files generated by BackupDb.
- RollbackComplete: The upgrade may have failed or you were not completely satisfied by the new version, but we brought everything back. You can reuse the shop as before.

Other / Misc

These steps don't follow any process and are independant between each others.

- CheckFilesVersion: This task is responsible to detect all local files has been modified by the merchant / developer. This is an important pre-requesite which warn about the potential loss of changes.
- Error: This task can be called from any other one, and will display a specific message saying the upgrade / restore process failed. In case of upgrade, a restore will be suggested.
- UpdateConfig: This task will store in a configuration file the different options selected in the web version of the module.

> External resources

Channels details

Note: there are two versions! This was a temporary solution for a specific 1.7 version of the module, but today there is no information in the main resource of PrestaShop 1.7.x.x. Another must be used.

> Local temporary assets

In order to work properly, the 1-click upgrade module needs to write some files to your filesystem server. These files are stored in the following folders, all available in the <admin folder>/autoupgrade path.

Folders

- backup: Folder in which the current state of the shop will be saved before upgrade. It contains files archive, DB structure & data.
- Download: Destination folder of the downloaded PrestaShop archive, before unzip.
- Latest: Working directory of the autoupgrade. This is where the “lastest” version of PrestaShop will be unziped, before copy.
- Tmp: Temporary resources not specifically used for upgrade. For instance, logs will be stored in that folder.