Skill - Review SDK files

Skill to (try to) port all custom files to vtenext 26 and make them compatible with vtenext 26. This skill simplifies the upgrade to the new version when coming from versions earlier than 26.
To use it you need to be one of our partners and have the credentials for the **ocean** area.

> To take full advantage of all its features, request the latest version of the `tools/skills/review-sdk-26` folder and copy it into your installation before running the skill.

> Run the skill **before** starting the update procedure via `vteUpdater.sh`

Install to Codex

Create the folder `review-sdk-26` in the skills folder of your IDE example: [...]/.codex/skills/ with the structure:

review-sdk-26/
├── SKILL.md
└── agents/
    └── openai.yaml

SKILL.md

```
---
name: review-sdk-26
description: Revise the SDK code to be compatible with VTENEXT version 26.*
---

# review-sdk-26

Review and update PHP files (>=7.x) to make them:
* Compatible with PHP 8.3
* Aligned with the new project guidelines
* More robust, typed, and modern
```

You are working on existing VTENEXT application code. DO NOT rewrite everything from scratch. Keep the original logic, improving it only where necessary.
The project's root directory is specified in the config.inc.php file in `$root_directory`. If that directory is not accessible, use the directory containing the file config.inc.php.

When to use

This skill should be used **before** the update to VTENEXT 26.* to ensure that all custom PHP files are compatible with PHP 8.3 and follow the new project guidelines.

Input

* List of PHP files. If the user does not provide one, first check that both scope-specific scripts exist. If they do, ask which scope they want to review before collecting the files:
  * **SDK customizations only:** execute `tools/skills/review-sdk-26/get_php_sdk_customizations.php`.
  * **All customer-added PHP files:** execute `tools/skills/review-sdk-26/get_php_all_customizations.php`. This includes the SDK customizations and PHP files that are not present in the SVN manifest for the current build.
* If either scope-specific script is missing, fall back to the previous behavior and execute `tools/skills/review-sdk-26/get_php_customizations.php` instead.
* The list of files should be provided in batches of 10 if there are too many files to review at once.

The all-customer-files scope requires `tools/skills/review-sdk-26/svn_php_files_{CURRENT_BUILD_NUMBER}.txt`, where `{CURRENT_BUILD_NUMBER}` is the current VTENEXT build number (`$enterprise_current_build` in the file `vteversion.php`). Do not execute `get_php_all_customizations.php` if the matching manifest is missing. Ask the user to download it first by executing `tools/skills/review-sdk-26/update_svn_php_manifest.sh {CURRENT_BUILD_NUMBER}`.

It also requires the update deletion package `vte_updater/packages/vte{CURRENT_BUILD_NUMBER}-{NEXT_BUILD_NUMBER}del.tgz`. The script reads `files.txt` and `directories.txt` from this package and excludes files removed by the target release, because they are obsolete core files rather than customer customizations. If the package is missing, ask the user to execute `./vteUpdater.sh --only-file-check`. If multiple matching deletion packages exist, the script uses the package matching the prepared `vte_updater/{CURRENT_BUILD_NUMBER}/files{NEXT_BUILD_NUMBER}` directory; if it cannot determine a unique target, stop and ask the user to remove the ambiguity.

The SVN manifests contain paths relative to the VTENEXT root. Directory exclusions are configured in `excluded_directories.txt`: entries without slashes match a directory name at any depth, while entries with slashes match a root-relative directory. File exclusions and shell-style patterns are configured in `excluded_files.txt`: entries without slashes match a filename at any depth, while entries with slashes match a root-relative path.

Instructions

0. Choose the review scope

* Check whether both `tools/skills/review-sdk-26/get_php_sdk_customizations.php` and `tools/skills/review-sdk-26/get_php_all_customizations.php` exist.
* If either file is missing, do not ask for a scope: execute `tools/skills/review-sdk-26/get_php_customizations.php` and continue with its results.
* If both files exist, unless the user has already specified the scope, ask whether to review only SDK customizations or all customer-added PHP files.
* Use `get_php_sdk_customizations.php` for SDK customizations only.
* Use `get_php_all_customizations.php` for all customer-added PHP files.
* Before using `get_php_all_customizations.php`, verify that `svn_php_files_{CURRENT_BUILD_NUMBER}.txt` exists for the current build. If it is missing, stop the operation and ask the user to download it with `update_svn_php_manifest.sh` before continuing.
* Verify that `vte_updater/packages/vte{CURRENT_BUILD_NUMBER}-{NEXT_BUILD_NUMBER}del.tgz` exists. If it is missing, ask the user to execute `./vteUpdater.sh --only-file-check` and stop. The script automatically excludes the deleted files and directories listed in that package.

1. Prepare the environment

* If the folder `vte_updater/{CURRENT_BUILD_NUMBER}` does not exists, ask to user to execute command `./vteUpdater.sh --only-file-check` and STOP the execution
* Create folder `vte_updater/{CURRENT_BUILD_NUMBER}/post_patches`, if already exists empty it
* If there are too many files, run the skill on 10 files at a time (batch)
* If you are running a batch, do not empty the folder
* Also copies unpatched files to the `post_patches` folder to maintain the same structure and allow manual review of all files
* If you find any hardcoded credentials or sensitive data while analyzing the code, stop everything and let me know to move this data to a separate configuration file

2. PHP 8.3 Compatibility Analysis

* Fix weak comparisons as described
[here](https://usermanual.vtenext.com/books/developers/page/weak-comparisons)
* Add type casts where necessary to functions that require a string as parameter, for example `trim()` `strpos()` and `stripos()`
* Do not replace functions if the existing ones work on PHP 8.3
* For empty string comparisons, replace with the `empty()` function
* For comparisons with `$mode` or `$sdk_mode`, the creation mode is usually checked and the empty string comparison is fine. `=== ''`

example

old code:
```php
if ($recordid == '') {
```

new one:
```php
if (empty($recordid)) {
```

3. Refactoring

RequestHandler

* As described [here](https://usermanual.vtenext.com/books/developers/page/requesthandler), access to the superglobal variables `$_REQUEST`, `$_GET`, and `$_POST` is no longer allowed.
* Replace reading their values with the `RH::r`, `RH::g`, and `RH::p` methods of the `RequestHandler` class found in include/utils/RequestHandler.php.
* When writing to superglobals, use the `RH::push_*` and `RH::pop_*` methods as described in the link above.
* Remove calls to `vtlib_purify()`; RequestHandler already handles them internally.

Database best practices and escaping

* Apply the new best practices as described
[here](https://usermanual.vtenext.com/books/developers/page/escaping-rules)
* In particular, replace `query_result` with `query_result_no_html` and `fetchByAssoc(...)`

```
with `fetchByAssoc(..., -1, false)`
* If HTML is generated in PHP and assigned to a smarty variable, use the
`\Vtenext\Types\HtmlString` class to enclose the HTML string
* In custom reports the method `getSDKBlock()` should be return a `\Vtenext\Types\HtmlString`
object instead of a string

#### Other substitutions
* Remove include/require of the file `Smarty_setup.php` that no longer exists
* Replace references to the `vtigerCRM_Smarty` class with `VteSmarty`
* Replace `session_start()` con `VteSession::start()`
* For reading and writing in session use the methods of the `VteSession` class found in
`include/VteSession.php`

### 4. Coding style

* Apply the guidelines described
[here](https://usermanual.vtenext.com/books/developers/page/coding-style)
* Add license header to the top of the file or update it if already present as described in
the link above
* Remove the closing tag `?>` at the end of PHP files

### 5. Other examples
If you are modifying a method called get_dependents_list or get_related list and towards the
end there is a piece of code similar to `$return_value['CUSTOM_BUTTON'] = $button;`, ensure
that the $button variable is of type `\Vtenext\Types\HtmlString`, for example in this way:
```
$return_value['CUSTOM_BUTTON'] = new \Vtenext\Types\HtmlString($button);
```

### 6. Important Rules

* Do NOT change the functional behavior
* Do NOT introduce external dependencies
* Maintain backward compatibility if possible
* Clear code > "smart" code
* Do NOT directly modify the original files
* Do NOT return complete updated code
* Perform a final check on the files with `php -l` or with `php8.3 -l` if the former is not at
version 8.3
* When you have finished analyzing all the files suggest to re-check the files copied into the
```

```
folder

### 6. Analyze differences in core files

* If the file `vte_updater/{CURRENT_BUILD_NUMBER}/differences.log` exists, it contains a list of core files that were manually modified by technicians to alter the software's standard behavior. These modifications must be preserved by applying them to the files of the new version to which you are updating.

#### Instructions

* Perform a diff between the files listed in `differences.log` and the original versions located in the `vte_updater/{CURRENT_BUILD_NUMBER}/src` directory.
* `{NEXT_BUILD_NUMBER}` it is the build I am about to update VTENEXT; if you cannot locate it, ask the user for it.
* Apply the differences to the files of the new version, which are located in the `vte_updater/{CURRENT_BUILD_NUMBER}/files{NEXT_BUILD_NUMBER}` directory.
* Do NOT modify that directory; instead, place the updated files into `vte_updater/{CURRENT_BUILD_NUMBER}/post_patches`.
* Revise the code to ensure compatibility with VTENEXT version 26.*, focusing ONLY on the portions of the code identified by the diff.
* The rest of the file MUST NOT be analyzed or modified in any way, as it is already updated to version 26.

### 7. Backup original files

To facilitate the diff process for the technician, generate the folder `vte_updater/{CURRENT_BUILD_NUMBER}/post_patches_originals` containing all the files present in the `post_patches` folder; retrieve these files from `vte_updater/{CURRENT_BUILD_NUMBER}/files{NEXT_BUILD_NUMBER}`, or, if they are not found there, from the `$root_directory`.

## Output

* List modified files
* Notify the user that modified files will be found in that folder so they can manually verify them and overwrite the originals after their review
* Once the files have been verified, you can either overwrite them in the `$root_directory` after the update or use the `--post-patches` option with the `vteUpdater.sh` command.
```

openai.yaml

```
interface:
  display_name: "Review SDK files"
  short_description: "Review the code of SDK customizations and make it compatible with
vtenext 26"
  default_prompt: "Use $review-sdk-26 to review code of SDK customizations and make it
compatible with vtenext 26."
```

Prompt

Select the skill by digit $review-sdk... and indicate the path to the folder.
example:

```
[$review-sdk-26]([...]/.codex/skills/review-sdk-26/SKILL.md) in the folder [vtenext folder].
The skill contains links to web pages with details and usage examples; read those pages.
```

The agent will ask you to choose between the two options:

1. **SDK customizations only**
   analyze and review only files containing registered SDK customizations
2. **All customer-added PHP files**
   - analyze and review files containing registered SDK customizations
   - analyze and review NEW files: files not present in the standard repository for the reference version
   - analyze modified core files: patches are applied to the files of the new version, and only the custom code portions are reviewed
   In this case, the agent will ask you to execute command `./vteUpdater.sh --only-file-check` and to download the manifest of standard files for the current version. Your partner account credentials will be required.

All files analyzed by the agent will be copied to the `vte_updater/{CURRENT_BUILD_NUMBER}/post_patches` folder, while the `post_patches_originals` folder will contain a copy of the original files: if it is a core file, it will be the updated version; otherwise, it is taken from the root directory.
This allows you to easily compare these two folders to verify the changes made by the agent.

We recommend that you manually check the files generated by the AI agent; afterwards, you can perform the update by specifying the custom files that should be overwritten, using the following command:
`./vteUpdater.sh [...] --post-patches=vte_updater/{CURRENT_BUILD_NUMBER}/post_patches`

Revision #18
Created 2026-05-12 09:20:45 UTC by manuel.tagliapietra
Updated 2026-10-30 13:23:43 UTC by m.maporti