

Contribuir

Cómo contribuir a LibreDTE

Anonymous · 08/10/2026

Contribuir a LibreDTE



Si deseas contribuir a LibreDTE, te lo agradezco. Después de más de 10 años aportando al software libre en Chile, a veces se hace difícil tener todo el tiempo que quisiera para realizar mejoras a LibreDTE. Por lo que cualquier corrección de error, mejora o nueva funcionalidad que tengas en mente me comprometo a revisarla.

[Esteban Delaf](#), Autor de LibreDTE.

Trabaja con una copia del repositorio

Si deseas contribuir con el proyecto, especialmente resolviendo alguna de las [incidencias abiertas](#), debes:

1. Hacer una copia mediante una bifurcación del [repositorio en GitHub](#) y luego clonar dicho repositorio.

```
git clone https://github.com/TU_USUARIO/libredte-lib-core
cd libredte-lib-core
```

2. Crear una rama para los cambios.

```
git checkout -b nombre-branch-que-blabla
```

3. Programar tu código siguiendo los lineamientos descritos más abajo.

```
declare(strict_types=1);

// Seguir el estándar PSR12 o, idealmente, PER Coding Style 3.0.
```

```
// Seguir los principios KISS, SRP, DRY, YAGNI y TDD.  
// Ejecutar las validaciones y pruebas.
```

4. Guardar los cambios en tu repositorio local.

```
git commit -m 'Se agrega...'
```

5. Subir los cambios a tu repositorio en Github.

```
git push origin nombre-branch-que-blabla
```

6. Crear una [solicitud en el repositorio de LibreDTE](#) para unir tu rama con la oficial de LibreDTE.

Lineamientos de código

El código debe seguir los siguientes lineamientos:

1. Respetar al menos el estándar [PSR12](#). Idealmente seguir [PER Coding Style 3.0](#).
2. Seguir los patrones y principios:
 - **KISS**: Mantén el código simple y directo. Evita complejidades innecesarias.
 - **SRP**: Cada clase y método debe tener una única responsabilidad.
 - **DRY**: Evita la duplicación de código. Si encuentras el mismo código en varios lugares, refactorízalo para que se utilice de manera centralizada.
 - **YAGNI**: No implementes funcionalidades que no sean necesarias en el presente. El código debe ser lo más minimalista posible, construyendo solo lo que realmente se necesita.
 - **TDD**: Escribe las pruebas antes del código. Esto ayuda a garantizar que el código haga lo que se supone que debe hacer.
3. Tener el 100% de cobertura en las pruebas.
4. Escribir la documentación correspondiente en el **código** utilizando PHPDoc, y que será validada por *phpstan* con al menos el nivel 5.

Validación del código

Una vez tengas tu código debes ejecutar los siguientes comandos para validarlo:

```
# Revisión del estilo de código según estándar definido para la biblioteca.  
php-cs-fixer fix -v --dry-run --diff --config=php-cs-fixer.php .
```

```
# Buscar errores potenciales en tu código sin ejecutar tests.  
phpstan analyse --configuration=phpstan.neon --memory-limit=1G  
  
# Pruebas unitarias, funcionales y de integración de toda la biblioteca.  
XDEBUG_MODE=coverage vendor/bin/phpunit --configuration=phpunit.xml
```

Importante

Si tu código no pasa las validaciones anteriores no será aceptado para ser unido al repositorio oficial.

Organización de pruebas

Las pruebas se organizan en 3 tipos:

1. **Unitarias:** deben probar solo una clase. Estas pruebas no deben realizar conexiones al SII.
2. **Funcionales:** prueban una funcionalidad con múltiples clases. Estas pruebas no deben realizar conexiones al SII.
3. **Integración:** prueban una funcionalidad que requiere realizar conexión al SII.

Última actualización el 08/10/2026