GLPI-Hooks

GLPI Hooks: Explicacion, Estructura y Creación

Implementación de Hooks en GLPI

1. Introducción a los Hooks

Los **Hooks** (o "ganchos") son un mecanismo fundamental del core de GLPI que permite a los desarrolladores extender y modificar el comportamiento estándar de la aplicación de una manera modular y segura. Operan bajo el principio de "inversión de control", donde el flujo principal de la aplicación cede temporalmente el control a código externo en puntos de ejecución predefinidos.

La implementación de hooks se realiza exclusivamente a través de **plugins**. Este enfoque garantiza que las personalizaciones no alteren el código fuente de GLPI, lo que asegura la integridad del sistema y facilita futuras actualizaciones sin conflictos.

Beneficios Clave:

- **Modularidad:** El código de la personalización reside de forma aislada dentro de un plugin.
- **Mantenibilidad:** Las actualizaciones del core de GLPI no sobrescriben las funcionalidades añadidas.
- **Interoperabilidad:** Permite la integración con servicios y APIs de terceros en respuesta a eventos específicos de GLPI.

2. Taxonomía de Hooks en GLPI

GLPI proporciona un amplio abanico de puntos de anclaje, clasificados según el evento que los dispara. A continuación, se detallan las categorías principales.

2.1. Hooks de Ciclo de Vida de Ítems (`item_*`)

Estos son los hooks más utilizados y se activan durante las operaciones CRUD (Create, Read, Update, Delete) sobre cualquier objeto que herede de la clase `CommonDBTM` (e.g., `Ticket` , `Computer` , `User`).

Hook	Punto de Ejecución	Caso de Uso Típico
`pre_item_add`	Inmediatamente antes de la inserción (`INSERT`) de un nuevo ítem en la base de datos.	Validación de datos personalizada, modificación de valores por defecto antes del guardado.
`post_item_add`	Inmediatamente después de la inserción de un nuevo ítem.	Notificaciones a sistemas externos (Slack, Teams), triggering de webhooks, logging avanzado.

Hook	Punto de Ejecución	Caso de Uso Típico
`pre_item_update`	Inmediatamente antes de la actualización (`UPDATE`) de un ítem existente.	Prevenir ciertas modificaciones basadas en lógica de negocio, auditar cambios.
`post_item_update`	Inmediatamente después de la actualización de un ítem.	Sincronización de cambios con una CMDB externa, invalidación de cachés.
`pre_item_delete`	Inmediatamente antes de la eliminación (`DELETE`) de un ítem.	Verificación de dependencias, archivo de datos antes de la eliminación.
`post_item_delete`	Inmediatamente después de la eliminación de un ítem.	Limpieza de registros relacionados en sistemas externos.

2.2. Hooks de Visualización (`display_*` , `pre/post_show_*`)

Permiten inyectar o modificar contenido HTML en la interfaz de usuario de GLPI.

Hook	Punto de Ejecución	Caso de Uso Típico
`display_central`	En la página de inicio principal de GLPI.	Añadir dashboards personalizados o paneles de información.
`post_show_item`	Después de renderizar el formulario de visualización de un ítem.	Inyectar un nuevo tab, un botón con acciones personalizadas, o mostrar datos de una API externa.
`pre_show_item`	Antes de renderizar el formulario de visualización de un ítem.	Modificar cabeceras o ejecutar lógica preparatoria antes de la visualización.

2.3. Hooks de Acciones Específicas

Vinculados a lógicas de negocio concretas, principalmente dentro del módulo de tickets.

- `assign_ticket` / `unassign_ticket` : Se activan al asignar o desasignar un técnico o grupo.
- `change_status` : Se activa cuando el campo de estado de un ticket es modificado.
- `add_followup` : Se activa al añadir un nuevo seguimiento a un ticket.

3. Implementación de un Hook: Un Ejemplo Práctico

La implementación se realiza mediante la creación de un plugin. A continuación se detalla la estructura y el código para un plugin de ejemplo: `AuditLogger`.

Objetivo: Registrar en un log cada vez que un ticket de alta urgencia es actualizado.

3.1. Estructura de Archivos del Plugin

La estructura mínima requerida debe ser creada dentro del directorio `plugins/` de GLPI.

```
glpi/
└── plugins/
    └── auditlogger/
        ├── hook.php       # Contiene la lógica del hook.
        └── setup.php      # Manifiesto y registro del plugin.
```

3.2. Manifiesto del Plugin (`setup.php`)

Este archivo actúa como el descriptor del plugin, registrando la correspondencia entre los eventos de GLPI y las funciones a ejecutar.

```php
<?php
// plugins/auditlogger/setup.php

/**
 * Función de inicialización del plugin. Registra los hooks.
 */
function plugin_init_auditlogger() {
    global $PLUGIN_HOOKS;

    // Declara compatibilidad con la protección CSRF de GLPI.
    $PLUGIN_HOOKS['csrf_compliant']['auditlogger'] = true;

    // Registra la función que se ejecutará en el evento 'post_item_update'.
    // Cuando GLPI dispare 'post_item_update', llamará a la función 'plugin_au
    $PLUGIN_HOOKS['post_item_update']['auditlogger'] = 'plugin_auditlogger_pos

    return true;
}

/**
 * Función que proporciona los metadatos del plugin a GLPI.
 */
function plugin_version_auditlogger() {
    return [
        'name'           => 'Audit Logger',
        'version'        => '1.0.0',
        'author'         => 'Equipo de TI',
        'license'        => 'GPLv2+',
        'homepage'       => '',
```

```
        'minGlpiVersion' => '10.0'
    ];
}
```

3.3. Lógica del Hook (hook.php)

Este archivo contiene la implementación de la función registrada en `setup.php` .

```php
<?php
// plugins/auditlogger/hook.php

/**
 * Se ejecuta después de la actualización de cualquier ítem en GLPI.
 *
 * @param CommonDBTM $item El objeto que ha sido actualizado.
 */
function plugin_auditlogger_post_item_update(CommonDBTM $item) {
    // 1. Filtrar por tipo de ítem: Solo nos interesan los Tickets.
    if ($item->getType() != 'Ticket') {
        return; // Salir si no es un ticket para optimizar el rendimiento.
    }

    // 2. Filtrar por condición: Solo tickets con urgencia alta (Urgencia 5 = I
    // El array 'fields' contiene el estado actual del objeto.
    if (!isset($item->fields['urgency']) || $item->fields['urgency'] < 5) {
        return;
    }

    // 3. Obtener datos relevantes del objeto.
    $ticketId     = $item->getID();
    $editorId     = Session::getLoginUserID(); // ID del usuario que realizó la
    $editorName   = User::getName($editorId);
    $ticketName   = $item->fields['name'];

    // 4. Construir el mensaje de log.
    $logMessage = sprintf(
        "[%s] AUDIT: Ticket de alta urgencia actualizado. ID: %d, Título: '%s'
        date('Y-m-d H:i:s'),
        $ticketId,
        $ticketName,
        $editorName['realname'],
        $editorId
    );

    // 5. Escribir en el archivo de log.
    $logFile = GLPI_LOG_DIR . '/auditlogger.log';
    file_put_contents($logFile, $logMessage, FILE_APPEND | LOCK_EX);
}
```

3.4. Proceso de Despliegue

1. **Copia de Archivos:** Copiar el directorio `auditlogger` completo a la carpeta `plugins/` de la instancia de GLPI.
2. **Instalación:** Navegar a `Configuración > Plugins` en la interfaz de GLPI. Localizar el plugin "Audit Logger" y hacer clic en **Instalar**.
3. **Activación:** Tras la instalación, hacer clic en **Activar**.

A partir de este momento, el hook está activo. Cualquier actualización realizada a un ticket con urgencia "Muy Alta" generará una entrada en el archivo `glpi/files/_log/auditlogger.log`, proporcionando una pista de auditoría invaluable para incidentes críticos.

Powered by Wiki.js