IBM Tivoli Workload Automation

# Guía del desarrollador: ampliando Tivoli Workload Automation

*Versión 9 Release 2*

SC14-7623-03

IBM Tivoli Workload Automation

# Guía del desarrollador: ampliando Tivoli Workload Automation

*Versión 9 Release 2*

SC14-7623-03

**Nota**

Antes de utilizar esta información y el producto al que se da soporte, lea la información del apartado "Avisos" en la página 27.

Esta edición se aplica a la versión 9, release 2, nivel de modificación 0 de Tivoli Workload Scheduler (número de programa 5698-WSH) y a todos los releases y modificaciones siguientes hasta que se indique lo contrario en nuevas ediciones.

# Contenido

# Figuras

# Tablas

# Acerca de esta publicación

*Guía del desarrollador: Ampliación de Tivoli Workload Automation* describe cómo utilizar la Interfaz de Servicios Web para controlar los objetos de IBM® Tivoli Workload Scheduler e IBM Tivoli Workload Scheduler for z/OS en sus respectivas planificaciones.

## Novedades de este release

Sepa qué novedades hay en este release.

Para obtener información sobre las funciones nuevas o modificadas de este release, consulte la publicación *Tivoli Workload Automation: Visión general*, sección *Resumen de mejoras*.

Para obtener más información sobre los APAR que se resuelven en este release, consulte Tivoli Workload Scheduler Notas del release en http://www-01.ibm.com/support/docview.wss?rs=672&uid=swg27041032 y Dynamic Workload Console Notas del release en http://www-01.ibm.com/support/docview.wss?rs=672&uid=swg27041033.

## Novedades de este release para la ampliación de Tivoli Workload Automation

Esta sección describe lo que ha cambiado en este release con relación a la ampliación de Tivoli Workload Automation con respecto a la versión 8.5 Fix Pack 01.

Ahora tiene la posibilidad de crear un plug-in de tipo de trabajo personalizado. Este plug-in añade un tipo de trabajo nuevo a la selección de definiciones de trabajo de Dynamic Workload Console. También puede crear un archivo JSDL adecuado para utilizarlo en la definición de trabajo en el **compositor**. El plug-in también contiene un componente que ejecuta el trabajo en el agente dinámico.

## Novedades de esta publicación

Ésta es una publicación nueva, aunque ha sido desarrollada a partir de los capítulos de la *Tivoli Workload Scheduler API Guide* y de los temas de Integration Workbench help of version 8.5 which related to plug-ins. Considere toda la información como si fuera nueva.

## A quién va dirigida esta publicación

Esta publicación ofrece información acerca de cómo añadir funciones a los productos Tivoli Workload Automation mediante la creación de plug-ins Java™ plug-ins.

El lector de este manual debe ser un *programador de aplicaciones* experto en Java, con un conocimiento suficiente de la infraestructura de Tivoli Workload Automation y de sus interacciones entre componentes. También puede ser el responsable de dicho programador que desee entender mejor lo que puede obtenerse mediante el uso de plug-ins.

La publicación da por supuesto que el programador de aplicaciones tiene experiencia en la creación y trabajo con plug-ins y Java. También da por supuesto que los conocimientos del producto necesarios para programar la API o la interfaz de servicios web se han obtenido de la documentación del producto. Esta publicación no pretende describir ninguno de los conceptos, procedimientos, y prácticas de Tivoli Workload Automation a los que hace referencia.

Esta publicación también contiene información de utilidad para el *administrador de TI* y para el administrador de TI de *Tivoli Workload Automation*, a efectos de planificación.

## Publicaciones

El producto Tivoli Workload Automation tiene el soporte de una serie de publicaciones.

Para obtener una lista de las publicaciones de la biblioteca del producto Tivoli Workload Automation, consulte *Publicaciones* bajo *Referencia* en la documentación del producto.

Para obtener una lista de los términos del producto Tivoli Workload Automation, consulte *Glosario* bajo *Referencia* en la documentación del producto.

## Accesibilidad

Las funciones de accesibilidad permiten que los usuarios que padecen alguna discapacidad física, como movilidad limitada o visión reducida, utilicen los productos de software de manera satisfactoria.

Con este producto, puede utilizar tecnologías de asistencia para la escucha y navegación de la interfaz. Puede también utilizar el teclado en lugar del ratón para utilizar todas las funciones de la interfaz gráfica de usuario.

Para obtener información completa relacionada con Dynamic Workload Console, consulte el apéndice sobre accesibilidad en el manual *IBM Tivoli Workload Scheduler: Guía de usuario y referencia*.

## Formación técnica de Tivoli

Tivoli proporciona formación técnica.

Para obtener información sobre la formación técnica de Tivoli, consulte el siguiente sitio web de formación de IBM Tivoli:

http://www.ibm.com/software/tivoli/education

## Información de soporte

IBM proporciona varias maneras de obtener soporte cuando se produce un problema.

Si tiene un problema con el software de IBM, deseará resolverlo con rapidez. IBM brinda los medios siguientes para que obtenga el soporte que necesita:

- Búsqueda en bases de conocimiento: puede realizar una búsqueda entre una amplia recopilación de problemas conocidos y soluciones posibles, notas técnicas (Technotes) y otra información.
- Obtención de arreglos: Puede localizar los últimos arreglos que ya están disponibles para el producto.
- Cómo contactar con el Centro de Soporte de software de IBM: Si todavía no puede resolver su problema y necesita trabajar con alguien de IBM, puede utilizar diversas maneras de ponerse en contacto con el Centro de soporte de software de IBM.

Para obtener más información sobre estas tres opciones para resolver problemas, consulte el apéndice sobre información de soporte en la publicación *Tivoli Workload Scheduler: Troubleshooting Guide*.

# Capítulo 1. Introducción a la ampliación de Tivoli Workload Automation

Ofrece una visión general y una introducción a la ampliación de Tivoli Workload Automation.

Puede ampliar Tivoli Workload Automation creando plug-ins que añadan funciones relevantes a las actividades de negocio en dos áreas principales:

- Automatización de carga de trabajo controlada por sucesos
- Tipos de trabajos con opciones avanzadas

También puede crear trabajos Java que implementen un proyecto Java creado por el usuario en la estación de trabajo de destino.

## Automatización de carga de trabajo controlada por sucesos

La automatización de la carga de trabajo controlada por sucesos es una característica de Tivoli Workload Scheduler utilizada para desencadenar Tivoli Workload Scheduler o acciones externas cuando se producen sucesos de Tivoli Workload Scheduler o sucesos externos.

Las acciones de suceso se inician automáticamente cuando se desencadenan debido a que ha tenido lugar una condición de suceso determinada. Una condición de suceso y la acción correspondiente se definen generalmente en una regla de suceso. Cuando una regla está activa, significa que se ejecuta un dispositivo de supervisión para detectar si tiene lugar el suceso definido en la regla. Cuando se detecta el suceso, se inicia la acción definida en la regla.

Diríjase a http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.tivoli.itws.doc_8.5.1.1/awsrgmst84.htm#dqx2evntdrivworkauto para obtener más información acerca de la automatización de carga de trabajo controlada por sucesos.

Tivoli Workload Scheduler se suministra con un conjunto de condiciones y acciones de suceso que el usuario configura en reglas de suceso. Sin embargo, si resultan insuficientes, puede utilizar Integration Workbench para crear plug-ins Java para condiciones y acciones de suceso a fin de realizar las tareas necesarias.

Por ejemplo, suponga que desea enviar un mensaje de Java Message Service cuando falla una secuencia de trabajos. Puede crear un plug-in Java para ejecutar este servicio, implementarlo en Tivoli Workload Scheduler y combinarlo con una regla de suceso con la condición adecuada en Dynamic Workload Console.

## Tipos de trabajos personalizados con opciones avanzadas

Al crear una definición de trabajo, puede elegir tipos de trabajos con opciones estándar y con opciones avanzadas. Estos últimos se implementan de modo diferente a los tipos de trabajos con opciones estándar, en los siguientes aspectos:

- Requieren que el agente dinámico esté instalado en la estación de trabajo en la que deben ejecutarse
- Se implementan mediante un plug-in independiente para cada tipo.

Por ejemplo, los trabajos de tipo Transferencia de archivos sólo pueden ejecutarse en estaciones de trabajo en las que esté instalado el agente dinámico y se ejecutan mediante un plug-in de transferencia de archivos.

Si tiene un trabajo cuyas características no se ajustan a los tipos de trabajos predefinidos en Tivoli Workload Scheduler o Tivoli Workload Scheduler for z/OS (según proceda), puede crear un plug-in con las características necesarias. Una vez implementado en el producto, quedará disponible para seleccionarlo como tipo de trabajo en Dynamic Workload Console.

## Trabajos Java

Al definir un trabajo de planificación nuevo en Tivoli Workload Scheduler o en Tivoli Workload Scheduler for z/OS, uno de los tipos de trabajos que puede elegir es "Java". Para cada trabajo Java que elija definir, debe identificar:

- Un jar que contenga las clases y métodos Java que desea ejecutar en la estación de trabajo de destino (en la que debe instalarse el agente dinámico)
- Un conjunto de parámetros que deben utilizarse como entrada para dichas clases y métodos

El proyecto Java que debe ejecutarse podrá realizar aproximadamente la función esperada, pero para que esta opción entre en vigor debe seguir un conjunto de reglas, que se describen en esta publicación.

# Capítulo 2. Integration Workbench

Describe el Integration Workbench de Software Development Kit.

IBM Tivoli Workload Automation: Software Development Kit viene con un Integration Workbench que puede utilizar para trabajar con la Java interfaz de programación de aplicaciones y la interfaz de servicios web para desarrollar sus propias aplicaciones.

Esta sección describe cómo instalar y utilizar la ayuda de Integration Workbench. La ayuda contiene información detallada acerca de las tareas que pueden realizarse con Integration Workbench y la información de referencia detallada acerca de los métodos y clases disponibles:

## Instalar Integration Workbench

Ofrece una visión general de la instalación de Integration Workbench.

Integration Workbench (parte de Software Development Kit - SDK) se ejecuta en Eclipse. La instalación, que se describe detalladamente en el manual *Tivoli Workload Scheduler: Guía de planificación e instalación*, permite instalar Integration Workbench y una versión empaquetada y soportada de Eclipse en una sola acción. Como alternativa, puede instalar Integration Workbench como *sitio Eclipse* utilizando una versión soportada existente de Eclipse disponible en la red.

En ambos casos, al final de la instalación, en el panel donde se pulsa **Finalizar**, existe una opción que permite visualizar el archivo `readmefirst.html`. Este archivo contiene información relativa al entorno de trabajo y cómo ejecutarlo. Esta información también se suministra aquí, en la sección "Utilizar la ayuda de Integration Workbench".

Para obtener más información acerca de Eclipse, diríjase a http://www.eclipse.org/.

## Utilizar la ayuda de Integration Workbench

Describe cómo acceder al recurso de ayuda de Integration Workbench para la API y los proyectos de plug-in de Tivoli Workload Scheduler.

Para utilizar la ayuda de Integration Workbench, haga lo siguiente:

1. Inicie el entorno de trabajo del siguiente modo:

   **Integration Workbench instalado con Eclipse**

   **UNIX** Inicie el archivo siguiente: <*inicio_TWS*>/TWS/IntegrationWorkbench/eclipse/eclipse

   **Windows**
   Vaya a **Inicio → Tivoli Workload Scheduler → Integration Workbench**

   **Integration Workbench instalado como sitio Eclipse**
   Abra su versión de Eclipse como haría habitualmente.

2. Seleccione la ubicación para guardar el espacio de trabajo de Eclipse. Eclipse solicita esta información cada vez que se ejecuta o se ejecuta el entorno de trabajo que contiene, a menos que seleccione la opción de guardar una ubicación determinada como valor predeterminado.
3. Cuando se abra la ventana de Eclipse, seleccione **Ayuda → Contenido de la ayuda**
4. Expanda **IBM Tivoli Workload Scheduler Integration Workbench**
5. Las opciones visualizadas proporcionan una amplia variedad de información acerca de Tivoli Workload Scheduler Integration Workbench. Por ejemplo, para ver detalles de todas las clases y métodos empleados en la API, expanda **Referencia** y seleccione **Referencia de la API**.

**Nota:** también puede leer esta información abriendo el documento siguiente en un navegador web: <*inicio_TWS*>/TWS/IntegrationWorkbench/readmefirst.html

# Capítulo 3. Plug-ins de gestión de sucesos de Tivoli Workload Scheduler

Describe los diversos tipos de plug-ins de gestión de sucesos.

Los dos objetos principales que forman las reglas de suceso son:

- Condiciones de suceso
- Acciones

## Plug-ins de condición de suceso

Una condición de suceso está representada por una instancia de la clase `EventCondition`, que consta de los elementos siguientes:

- Un `pluginName`, que identifica el plug-in (o proveedor) de supervisor de sucesos que es capaz de capturar el suceso
- Un `eventName`, que identifica la condición en la regla
- Un `eventType`, que califica el suceso que debe capturarse
- Un `filteringPredicate`, que define un filtro que debe aplicarse al contenido del suceso a fin de comprobar que coincide con la condición de suceso
- Un atributo de sólo lectura `scope`, que el plug-in de sucesos calcula cuando se crea o actualiza la regla. Incluye una definición del ámbito de la condición de suceso, cuyo significado es diferente para los diversos tipos de sucesos.

Un plug-in de sucesos es una colección de tipos de sucesos agrupados debido a que comparten características similares, se centran en la misma área operativa o supervisan objetos similares.

## Plug-ins de acción de suceso

Una acción está representada por una instancia de la clase `RuleAction`, que consta de:

- Un `pluginName`, que identifica el plug-in (o proveedor) de acción.
- Uno o varios tipos de acción específicos que deben ejecutarse en un punto temporal específico definido por `responseType` en la regla
- Un atributo de sólo lectura `scope`, que el plug-in de acciones calcula cuando se crea o actualiza la regla. Incluye una definición del ámbito de la acción especificada.

Un plug-in de acciones es una colección de tipos de acciones agrupados debido a que comparten características similares, se centran en la misma área operativa u operan en objetos similares.

## Estructura de una condición o acción de suceso

En Tivoli Workload Scheduler, los plug-ins de condiciones y acciones están estructurados en los componentes siguientes:

**TWSPluginConfiguration.xml**
: Archivo XML utilizado para declarar las condiciones o acciones de suceso y los atributos relacionados con ellas.

**TWSPlugin.properties**
Archivo de propiedades que define las siguientes características configurables del plug-in:

- El nombre
- El tipo (condición o acción)
- El nombre de clase Java que implementa directamente la interfaz del plug-in

**Una o varias unidades de compilación Java**
Una de ellas contiene una clase que implementa la interfaz Java del plug-in de sucesos (este nombre de clase se define en TWSPlugin.properties). La clase Java puede añadir funciones al plug-in y también generar sucesos o acciones. En los plug-ins de ejemplo, estas funciones se incluyen generalmente en otras unidades de compilación contenidas en el mismo paquete Java.

# Proyecto de plug-in de gestión de sucesos

Describe el proyecto de plug-in de gestión de sucesos.

Un proyecto de plug-in de Tivoli Workload Scheduler es un tipo especial de proyecto Java que puede identificarse en la vista Navegador con el icono siguiente:

Para simplificar la creación en Eclipse de plug-ins de gestión de sucesos, los proyectos tienen una estructura específica e incluye una acción de “Preparar el despliegue” en la página 10. Esta acción, partiendo de archivos creados por el usuario en la estructura del proyecto, empaqueta los archivos del plug-in en un formato preparado para copiarse en el procesador de sucesos.

La sección “Estructura de un proyecto de plug-in de gestión de sucesos” describe la estructura de un proyecto de plug-in de Tivoli Workload Scheduler y el contenido de cada carpeta del proyecto.

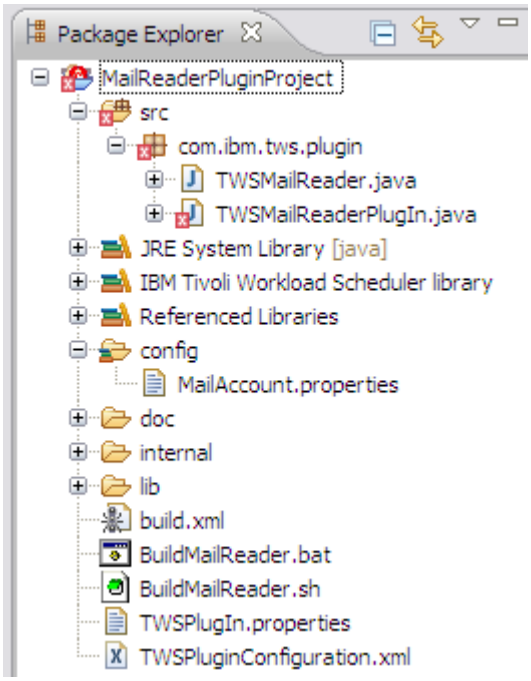
La sección “Preparar el despliegue” en la página 10 describe cómo empaquetar los nuevos plug-ins y desplegarlos en Tivoli Workload Scheduler.

- **“Estructura de un proyecto de plug-in de gestión de sucesos”**
- **“Preparar el despliegue” en la página 10**
- **“Generar y enviar un suceso” en la página 10**

## Estructura de un proyecto de plug-in de gestión de sucesos

Describe la estructura de un proyecto de plug-in de gestión de sucesos.

La figura siguiente muestra la estructura típica de un proyecto de plug-in de gestión de sucesos:




La estructura mostrada corresponde a un plug-in de condiciones de suceso denominado `MailReader`, pero pero los proyectos de plug-in de condiciones y de acciones de suceso tienen la misma estructura.

- Las clases Java contenidas en la carpeta `src` se describen en la sección "Árbol de origen Java (src)".
- La vía de construcción Java del proyecto de plug-in se describe en la sección "Vía de construcción Java" en la página 8.
- El rol y el contenido de otras carpetas de proyecto se describen en la sección "Otras carpetas de proyecto" en la página 8.
- El archivo `build.xml` ANT se describe en la sección "build.xml" en la página 9.
- El archivo `TWSPlugIn.properties` se describe en la sección "TWSPlugin.properties" en la página 9.
- El archivo `TWSPluginConfiguration.xml` se describe en la sección "TWSPluginConfiguration.xml" en la página 9.

## Árbol de origen Java (src)

Describe el árbol de origen Java del plug-in de gestión de sucesos.

Al crear un proyecto de plug-in, éste se configura como proyecto Java con carpetas separadas para archivos de origen y de clase. La carpeta de origen se denomina `src`. Contiene el código Java del plug-in.

También se crea una clase Java junto con el nuevo proyecto de plug-in. Esta clase implementa la interfaz Java para el plug-in de sucesos o de acciones, según corresponda. Esta clase debe estar presente en todos los plug-ins de sucesos o acciones.

Siga la documentación Java (Javadocs) de las interfaces como referencia para averiguar el método que debe implementar. Para el registro y rastreo del código, utilice las API de registro Java JSR-047 estándar.

## Vía de construcción Java

Describe la vía de construcción Java del plug-in de gestión de sucesos.

Los proyectos de plug-in de Tivoli Workload Scheduler se crean con las bibliotecas siguientes:

**Biblioteca del sistema del JRE predeterminado**
Aunque el JRE predeterminado esté establecido de forma predeterminada, recuerde que el procesador de sucesos de IBM Tivoli Workload Scheduler utiliza IBM JDK versión 1.5.

Se recomienda la utilización de IBM JDK versión 1.5 para los proyectos de plug-in de Tivoli Workload Scheduler.

**Biblioteca de IBM Tivoli Workload Scheduler**
Esta biblioteca contiene todos los jar de Tivoli Workload Scheduler necesarios para implementar plug-ins de Tivoli Workload Scheduler o para utilizar las API de Tivoli Workload Scheduler.

La biblioteca también define las reglas de acceso para las clases de los jar: las API públicas se definen como Accessible (accesibles), mientras que las clases internas se definen como Discouraged (desaconsejadas).

La utilización de clases desaconsejadas se marca de forma predeterminada con avisos de compilador. En cualquier caso, el uso de estas clases no está soportado.

**Bibliotecas referenciadas**
Estas bibliotecas contienen todas las demás clases, por ejemplo las necesarias en el caso de un plug-in de lector de correo para descargar, acceder y leer el correo.

Las bibliotecas adicionales necesarias para el código Java del plug-in pueden copiarse en la carpeta `lib` y añadirse a la vía de construcción Java.

## Otras carpetas de proyecto

Describe otras carpetas de proyecto del plug-in de gestión de sucesos.

Además de las carpetas de archivos de clase y de origen estándar, el archivo de `Preparar para el despliegue de TWS` utiliza las carpetas siguientes para empaquetar el plug-in a fin de desplegarlo en el procesador de sucesos.

**config** Utilice este carpeta para almacenar archivos de configuración adicionales (por ejemplo, archivos de propiedades) que el administrador de Tivoli Workload Scheduler debe editar para esta operación. La acción `Preparar para despliegue de TWS` copia estos archivos en la carpeta `dist` externa al jar del plug-in. Para acceder a estos archivos desde el código Java, cárguelos como recursos desde el cargador de clases.

**doc** Utilice esta carpeta para almacenar la documentación de plug-in, como por ejemplo Javadocs o documentación acerca de los sucesos y acciones definidos.

**lib** Contiene bibliotecas jar adicionales necesarias para la clase Java del plug-in. La acción `Preparar para el despliegue de TWS` copia estos jar en la carpeta `dist` para utilizarla en tiempo de ejecución. Los jar de este directorio deben añadirse manualmente a las bibliotecas de vía de construcción Java.

**dist** Este es el directorio de destino del proceso de `Preparar para el despliegue de TWS`. Después de ejecutar la acción `Preparar para el`

despliegue de TWS desde el menú del proyecto, esta carpeta contendrá todos los archivos que deben copiarse en Tivoli Workload Scheduler.

## build.xml

Describe el archivo build.xml del plug-in de gestión de sucesos.

Es es un archivo de compilación ANT estándar. Puede modificarlo según sus necesidades, pero tenga en cuenta que la acción Preparar el despliegue de TWS utiliza el destino dist del archivo.

## TWSPlugin.properties

Describe el archivo TWSPlugin.properties del plug-in de gestión de sucesos.

TWSPlugin.properties es un archivo de propiedades que Tivoli Workload Scheduler lee cuando se carga el plug-in.

Cuando se crea un proyecto de plug-in, éste ya contiene un archivo TWSPlugin.properties establecido con la información recopilada desde el asistente de proyecto de plug-in nuevo. No es necesario modificarlo a menos que se cambia alguna información más adelante.

TWSPlugin.properties define las siguientes características configurables del proveedor:

- El nombre del plug-in. Debe ser el mismo valor devuelto por la clase Java del plug-in con el método getPluginName y el mismo valor definido como nombre del plug-in en TWSPluginConfiguration.xml.
- El tipo: suceso o acción.
- El nombre de la clase Java que implementa directamente la interfaz del plug-in.

### Ejemplo

Este es un ejemplo del archivo TWSPlugin.properties para un plug-in denominado MailReader.

```
TWSPlugIn.name=MailReader
TWSPlugIn.type=event
TWSPlugIn.className=mycompany.mailreader.MailReaderPlugin
```

"Árbol de origen Java (src)" en la página 7

## TWSPluginConfiguration.xml

Describe el archivo TWSPluginConfiguration.xml del plug-in de gestión de sucesos.

TWSPluginConfiguration.xml se utiliza para declarar todos los sucesos o acciones suministrados por el plug-in.

El archivo define los atributos de lista para los sucesos o acciones, sus restricciones sintácticas y, sólo para sucesos, las condiciones de filtro permitidas.

Cuando se despliega en Tivoli Workload Scheduler, este archivo se utiliza para controlar la configuración de los predicados de filtro de sucesos o los atributos de acción dentro de las reglas de suceso.

Edite este archivo para especificar la estructura de los sucesos o acciones suministrados por el plug-in.

## Crear un proyecto de plug-in desde cero

Describe cómo crear un proyecto de plug-in de gestión de sucesos desde cero.

Mediante Integration Workbench puede crear proyectos de plug-in desde cero. Suministrará la información básica acerca del proyecto que desee crear y el asistente hará el resto.

## Crear un proyecto a partir de un ejemplo de plug-in

Describe cómo crear un proyecto de plug-in de gestión de sucesos a partir de un ejemplo.

Mediante Integration Workbench puede crear proyectos basados en los ejemplos suministrados. Siguiendo este procedimiento, evitará la necesidad de crear la estructura de proyecto de plug-in completa desde cero. Elegirá un ejemplo lo más parecido posible a sus requisitos y luego lo modificará según sea necesario.

### Ejemplos entre los que puede elegir

Describe los ejemplos disponibles para crear un proyecto de plug-in de gestión de sucesos.

Los ejemplos entre los que puede elegir son los siguientes:

**MailReaderPlugin**
Genera un suceso de Tivoli Workload Scheduler cuando se recibe un correo electrónico.

**SmsSenderPlugin**
Genera y envía un SMS.

## Preparar el despliegue

Describe cómo preparar el despliegue de un plug-in de gestión de sucesos.

Esta acción está disponible en el menú de proyectos de plug-in de Tivoli Workload Scheduler. Ejecuta el destino `dist` del archivo de compilación ANT `build.xml` del proyecto.

Esta acción coloca en la carpeta `dist` del proyecto los archivos que más adelante deben copiarse en el procesador de sucesos de Tivoli Workload Scheduler. Son los siguientes:

- Un archivo jar con los archivos de clase Java y una copia de `TWSPlugin.properties` y `TWSPluginConfiguration.xml`
- Una copia de las bibliotecas jar de usuario presentes en la carpeta `lib`, si existen
- Una copia del archivo de configuración presente en la carpeta `config`, si existe
- Una carpeta `doc` con la documentación del plug-in en formato html

"build.xml" en la página 9

"Otras carpetas de proyecto" en la página 8

## Generar y enviar un suceso

Describe los diversos métodos que pueden utilizarse para generar y enviar un suceso.

Puede utilizar varios procedimientos para generar sucesos y enviarlos a Tivoli Workload Scheduler. Algunos de ellos son los siguientes:

- Codificar directamente las instrucciones en la definición del plug-in
- Utilizar la acción `sendevent` externamente desde la definición del plug-in de una de las siguientes formas:
  - Utilizar el método `sendEvent` invocado por clases Java
  - Ejecutar el mandato `conman sendevent`

## Conexión a Tivoli Workload Scheduler

Describe cómo conectarse a Tivoli Workload Scheduler desde un plug-in de gestión de sucesos.

Conéctese a Tivoli Workload Scheduler con el código siguiente:

```
private Subject getSubject(String serverName,
                           String serverPort,
                           String uid,
                           String pwd){
 Subject subject = null;
 try {
   LoginContext lc = null;

   Hashtable env = new Hashtable();
   env.put(Context.INITIAL_CONTEXT_FACTORY,INITIAL_CONTEXT_FACTORY);
    env.put(Context.PROVIDER_URL,"corbaloc:iiop:"+serverName+":"+serverPort);

   final InitialContext initialContext = new InitialContext(env);
   Object obj = initialContext.lookup("");


lc = new LoginContext(JAAS_MODULE, new WSCallbackHandlerImpl(uid,pwd));
   lc.login();

   subject = lc.getSubject();

 }   catch(javax.naming.NoPermissionException exc) {
   System.err.println("[TWSConn] - Login Error: "+exc);

 }   catch(Exception exc) {
   System.err.println("[TWSConn] - Error: "+exc);
 }
   return subject;
 }
```

Son necesarios los siguientes datos de entrada de usuario:

- `serverName`: el nombre de instalación de Tivoli Workload Scheduler
- `serverPort`: generalmente, 33116 o 33117
- `uid`: ID del usuario de Tivoli Workload Scheduler
- `pwd`: contraseña de la cuenta de Tivoli Workload Scheduler

## Material de referencia

Describe dónde buscar material de referencia sobre los plug-ins de gestión de sucesos.

La ayuda de Integration Workbench contiene todo el material de referencia necesario.

Para acceder a este material, siga estos pasos:

1. En Integration Workbench, seleccione **Ayuda → Contenido de la ayuda**

2. Expanda **Tivoli Workload Scheduler Integration Workbench** y luego **Referencia**
3. Obtenga material de referencia para cualquiera de los siguientes elementos:
   - La información necesaria para ejecutar los asistentes que crean proyectos de plug-in, ya sea desde cero o a partir de un ejemplo
   - Información acerca de las bibliotecas de los jar de objetos y de tiempo de ejecución
   - Descripción de los esquemas XML
   - Un enlace con información relativa a Tivoli Event Integration Facility
   - Referencia completa de todas las clases y métodos Java utilizados en el plug-in

# Capítulo 4. Plug-in de tipo de trabajo personalizado con opciones avanzadas

Ofrece una visión general del plug-in de tipo de trabajo personalizado con opciones avanzadas.

En Tivoli Workload Scheduler y Tivoli Workload Scheduler for z/OS, puede crear trabajos de tipos diferentes. El tipo de trabajo determina las características de un trabajo. Por ejemplo, un trabajo de tipo transferencia de archivos tiene todas las características necesarias para realizar una transferencia de archivos. Cuando se selecciona crear un trabajo de este tipo, se invita al usuario a especificar los parámetros de transferencia de archivos (qué archivo, desde dónde y a dónde) en un panel dedicado de Dynamic Workload Console.

Al elegir un tipo de trabajo en Dynamic Workload Console, los tipos de trabajos se dividen en dos grupos:

**Trabajos con opciones estándar**
Los trabajos básicos que ejecutan scripts y mandatos.

**Trabajos con opciones avanzadas**
Ejecutan trabajos más especializados. Se implementan con un plug-in, y sólo pueden instalarse en estaciones de trabajo donde se haya instalado el agente dinámico, aunque no tengan que planificarse dinámicamente. En Dynamic Workload Console, se elige entre un conjunto de estos tipos de trabajos el que cubre las actividades más habituales que una organización puede necesitar ejecutar (por ejemplo, Transferencia de archivos, Servicios Web, Base de datos y Java). Sin embargo, si desea planificar un tipo de trabajo cuyos requisitos son diferentes de aquellos de los tipos de trabajos predefinidos, puede crear su propio plug-in para añadir un nuevo tipo de trabajo adecuado a sus necesidades de negocio. Esta operación se realiza con Integration Workbench.

Para realizar las operaciones incluidas en tipo de trabajo personalizado con plug-ins de opciones avanzadas, necesita el acceso **run** definido en el archivo de seguridad de la estación en la que ha planificado ejecutar el plug-in. Debe aplicar las siguientes condiciones:

- Si la operación se realiza en el Conector Tivoli Workload Scheduler, se necesita visualizar y ejecutar un acceso a la CPU que corresponda a la estación de trabajo en la que se ha creado el trabajo.
- Si la operación se realiza en la estación de trabajo en la que se ejecuta el trabajo, se necesita un acceso de terminal en la estación de trabajo del intermediario de la carga de trabajo.

Consulte *Tivoli Workload Scheduler: Administration Guide* para más información acerca de la configuración del archivo de seguridad y la definición de la `ejecución` y las palabras clave de acceso de `objetos`.

## Anatomía de un plug-in de tipo de trabajo personalizado

Un plug-in de tipo de trabajo personalizado consta de:

- Un panel de Dynamic Workload Console en el que el usuario especifica los detalles de la definición de trabajo

- La parte de servidor del plug-in que realiza la validación opcional y crea la definición de trabajo.
- La parte de agente del plug-in que ejecuta el trabajo en la estación de trabajo de destino

## Pasos de creación de un plug-in de tipo de trabajo personalizado

La creación de un plug-in de tipo de trabajo personalizado requiere los pasos siguientes:

**1. Diseñar un panel de interfaz gráfica**

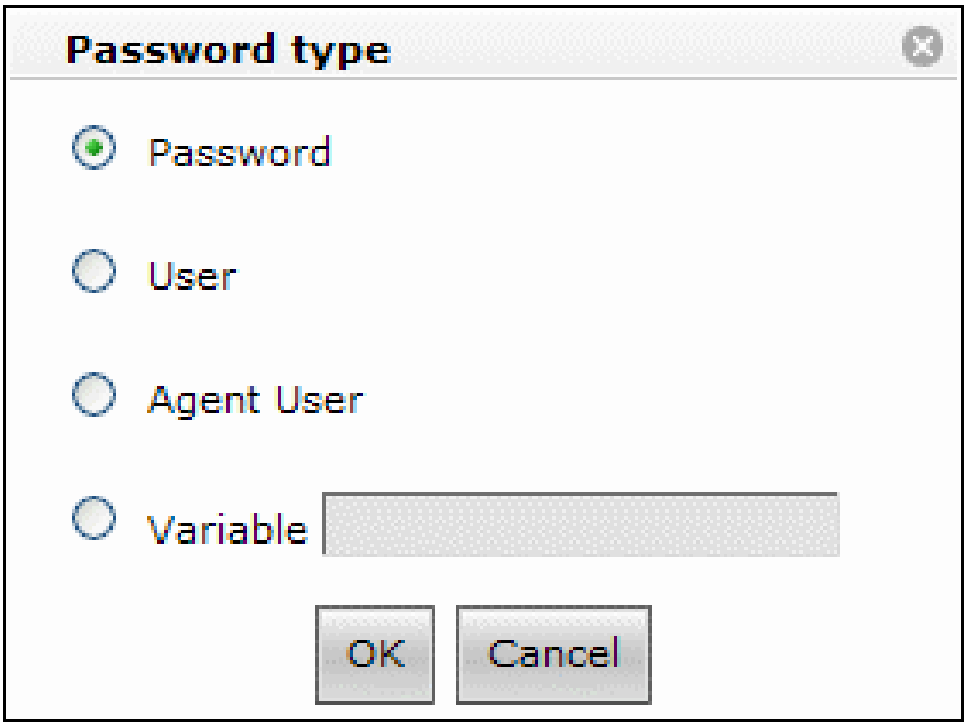
El usuario genera su propio panel para Dynamic Workload Console que se utilizará como entrada para el tipo de trabajo personalizado. Para crear un panel de interfaz de usuario, se utiliza la herramienta VisualBuilder de AUIML (Abstract User Interface Markup Language). Puede añadir:

- Botones
- Recuadros de selección
- Recuadros combinados
- Recuadros de edición
- Recuadros de lista
- Grupos
- Botones de selección
- Listas dinámicas
- Listas de pares dinámicas

Tenga en cuenta las siguientes restricciones:

- Al asignar un nombre a los objetos GUI, no utilice el prefijo `tws`, que está reservado para uso interno. Por la misma razón, evite el uso de los siguientes términos como nombres distintos (aunque pueden formar parte de series):
  - name
  - member
  - estación de trabajo
  - trabajo
  - jobstream
- No se permiten anidamientos de dos niveles para los grupos de selección o de recuadros de selección. Tampoco se pueden colocar los campos necesarios en grupos anidados de dos niveles.

Si incluye los campos `nombre de usuario` y `contraseña` en los paneles, puede beneficiarse del widget siguiente (proporcionado con el producto) que permite al usuario (quien definiría el trabajo en la mayoría de los casos) elegir el origen de la contraseña solicitada. El widget se visualiza al pulsar el botón de puntos suspensivos (...) situado junto al campo de contraseña y tiene el aspecto siguiente:




donde las opciones son las siguientes:

**Contraseña**
Utiliza el valor de contraseña especificado en el campo `Contraseña`.

**Usuario**

Añade la serie `${password:`*`valor_de_campo_de_nombre_usuario`*`}` al campo de contraseña.

**En los agentes dinámicos**
Se resuelve en tiempo de ejecución con el valor de contraseña definido para `Nombre de usuario` en la base de datos de Tivoli Workload Scheduler utilizando el panel de definición `Usuario` o el mandato `composer user`.

También puede especificar el usuario (y su contraseña) de otra estación de trabajo si se ha definido en la base de datos. Consulte la descripción del botón **Variable**.

**Atención:** Las definiciones de usuario carecen de integridad referencial. Esto implica que, si una definición de usuario a la que se hace referencia en la sección de credenciales de un trabajo se cambia o suprime, no se devolverá ningún mensaje de aviso o error hasta que se ejecute el trabajo.

**En los agentes de Tivoli Workload Scheduler for z/OS**
Se resuelve en el tiempo de ejecución con el valor de contraseña definido para `Nombre de usuario` en la base de datos de Tivoli Workload Scheduler for z/OS utilizando la sentencia de inicialización USSREC, donde el valor de `Nombre de usuario` está definido por el parámetro `USRNAM` y la contraseña por `USRPSW`.

**Usuario de agente**

Añade la serie
${agent:password.*valor_de_campo_de_nombre_usuario*}.

Se resuelve en tiempo de ejecución con el valor de contraseña definido para `Nombre de usuario` localmente en el agente dinámico o el agente de Tivoli Workload Scheduler for z/OS que ejecutará el trabajo (o en cualquier agente de una agrupación o agrupación dinámica que pueda ejecutar el trabajo) con el mandato `param`.

**Variable**

Se resuelve en tiempo de ejecución con el valor definido para la variable que especifica en el campo (utilizando la notación ${*nombre_variable*}).

**En los agentes dinámicos**

La variable debe haberse definido de forma local en el agente mediante el mandato `param` o en la base de datos Tivoli Workload Scheduler, utilizando el panel `User` o el mandato `composer username`. Por ejemplo:

- Una variable definida localmente en el agente, especifíquela aquí como:

  ```
  ${agent:file_With_Sections.password.dbPwd}
  ```

- Una variable definida en la base de datos, especifíquela aquí como:

  ```
  ${password:workstation#user}
  ```

Puede utilizar este botón para especificar la contraseña del usuario remoto de una estación de trabajo distinta (siempre que se haya definido en la base de datos) si especifica la cadena siguiente en el campo adyacente:

${password:*nombre_estación_trabajo#valor_campo_nombre_usuario*}

**En los agentes de Tivoli Workload Scheduler for z/OS**

Use este campo si desea utilizar la contraseña definida para un usuario distinto del especificado en el campo `User Name`.

Por ejemplo, si define un trabajo de Transferencia de archivos y los nombres de usuario local y remoto son idénticos (`user1`), puede diferenciar la contraseña si define dos entradas de sentencia de inicialización USRREC (por ejemplo, una para `user1` y otra para `user1remote`). Después de esto, en el campo de la contraseña de usuario remoto, especifique:

```
${password:user1remote}
```

El mecanismo tradicional de sustitución de variable que utiliza variables de usuario definidas en las tablas de variable de la base de datos de Tivoli Workload Scheduler for z/OS no está soportado en este campo.

La variable se resuelve cuando genera un plan y cuando somete un trabajo o una secuencia de trabajos. Mientras se definen trabajos, las variables no se resuelven y no se pueden utilizar en lista ni para conexiones de prueba.

Deba añadir la sentencia siguiente en la codificación del campo de contraseña para visualizar el widget y asociar la opción elegida al campo:

```
BINDING="passwordSelector#modeling.widgets.password.PasswordSelector:
  <nombreUsuario>"
```

**2. Opcionalmente, añadir botones al panel para realizar acciones de servicio**

Puede añadir botones a los paneles personalizados en Dynamic Workload Console, por ejemplo para realizar las siguientes acciones de servicio:

- Abrir una lista de selección de valores posibles para un parámetro del trabajo desde el conector
- Abrir una lista de selección de valores posibles para un parámetro del trabajo desde un agente

Cuando se añade el botón, se asocia la acción correspondiente de un campo de acción (ACTION) del plug-in. La definición de una acción requiere lo siguiente:

- Una definición de la acción en la definición de plug-in AUIML
- Una opción que determine si la acción debe realizarse en el conector (como resultado inmediato y directo de pulsar el botón)
- Un método que realice la acción necesaria

Puede desencadenarse una o varias de las acciones siguientes:

- Un campo puede situarse en estado de error con un mensaje que describe el error
- Puede visualizarse un menú emergente
- La información de diagnóstico puede rastrearse en Dashboard Application Services Hub
- Puede visualizarse una lista de selección

**3. Crear un proyecto de plug-in**

Utilice Integration Workbench para generar un proyecto de plug-in.

Este paso consiste en crear un paquete con los archivos del plug-in y todos lo necesario para desplegarlos. Haga lo siguiente:

1. En la barra de herramientas, seleccione **Archivo→Nuevo→Proyecto...**. Se abrirá la ventana `Proyecto nuevo`.
2. En la lista de asistentes, efectúe una doble pulsación primero en **IBM Tivoli Workload Scheduler** y luego en **Proyecto de ejecutor de TWS**. Ser abrirá el asistente `Proyecto de ejecutor de TWS`.
3. Especifique un nombre para el proyecto.
4. Introduzca un nombre de paquete para el plug-in.
5. Especifique el nombre del ejecutor de trabajos relacionados con el plug-in.
6. Especifique el nombre *nombre-archivo_panel*.AUIML (o utilice la herramienta de búsqueda).
7. Especifique los nombres de uno o varios archivos *nombre-archivo_panel*.properties (o utilice la herramienta de búsqueda para seleccionarlos).
8. Especifique el nombre del archivo de iconos relacionado (o utilice la herramienta de búsqueda). El archivo de iconos sólo puede ser de tipo PNG.
9. Pulse en **Finalizar**.

El asistente crea el proyecto de ejecutor y genera:

- Un archivo .XSD que define el esquema de la definición de trabajo
- Todas las clases necesarias para correlacionar el archivo JSDL
- Un archivo labels.properties donde pueden insertarse las etiquetas

Para insertar más etiquetas, edite el archivo labels.properties ubicado en la carpeta de recursos (RESOURCES) del proyecto. El archivo ya incluye una etiqueta predeterminada. Añada cada una de las etiquetas traducidas con el formato siguiente:

```
código de idioma = etiqueta
```

utilizando el mismo entorno local utilizado en el archivo de propiedades correspondiente.

El ID de plug-in es el nombre del plug-in en minúsculas.

**Edite los archivos del proyecto de plug-in**

Edite los archivos creados para el proyecto:

- Dé un nombre al panel que se mostrará en el menú desplegable **Nuevo ► Definición de trabajo** de Dynamic Workload Console (también puede añadir nombres traducidos)
- Establezca la versión del plug-in que desea utilizar, si no desea utilizar el valor predeterminado 1.0.0
- Añada los mensajes, incluidos los mensajes traducidos, a los que desee hacer referencia en la codificación
- Codifique la validación que desee realizar, tanto en la entrada como en la ejecución
- Cree listas de selección para los valores de campo
- Codifique los requisitos de salida especiales que sean necesarios para este tipo de trabajo

**4. Probar el plug-in**

Utilice el proyecto de prueba (generado automáticamente por el asistente `Proyecto de ejecutor de TWS` al crear el proyecto de plug-in) para probar el plug-in.

**5. Desplegar el plug-in**

Exporte los archivos de plug-in empaquetados del entorno de trabajo de Eclipse a una carpeta de su sistema o de otro sistema, a fin de poder instalarlos posteriormente en los agentes y convertirlos en operativos. Haga lo siguiente:

1. En el panel Explorador de paquetes, pulse el nombre de paquete con el botón derecho del ratón y seleccione **Exportar...**.
2. En la ventana Exportar, efectúe una doble pulsación primero en **IBM Tivoli Workload Scheduler** y luego en **Ejecutor de TWS**. Se visualizará una lista de proyectos de Ejecutor de TWS definidos.
3. Seleccione el proyecto en la lista y especifique la carpeta de destino de su sistema o de otro.
4. Pulse en **Finalizar**.

Con ello se creará un archivo jar denominado *id_plug-in_versión*.jar (donde *id_plug-in* es el nombre del plug-in en minúsculas) en la carpeta de destino del sistema seleccionado. Este nombre no puede modificarse.

**6. Instale el plug-in en los gestores de dominio maestro, gestores de dominio dinámicos, y en las estaciones de trabajo de seguridad relacionadas, si las**

**hubiera**

Copie el archivo .jar del plug-in en los gestores de dominio maestro de la vía de acceso `installation_dir/TWS/applicationJobPlugIn`, gestores de dominio dinámicos, y en las estaciones de trabajo de seguridad relacionadas, si las hubiera. Reinicie WebSphere Application Server parapara efectuar los cambios.

**7. Instalar y configurar el plug-in en los agentes**

Haga lo siguiente:

1. Copie el plug-in en la vía de acceso `../TWA/TWS/JAVAEXT/eclipse/plugins` de cada agente dinámico donde desee ejecutarlo.
2. En cada agente, edite el archivo `config.ini` ubicado en la vía de acceso `../TWA/TWS/JAVAEXT/eclipse/configuration` añadiendo la línea siguiente:

   ```
   id_plug-in@4:start
   ```

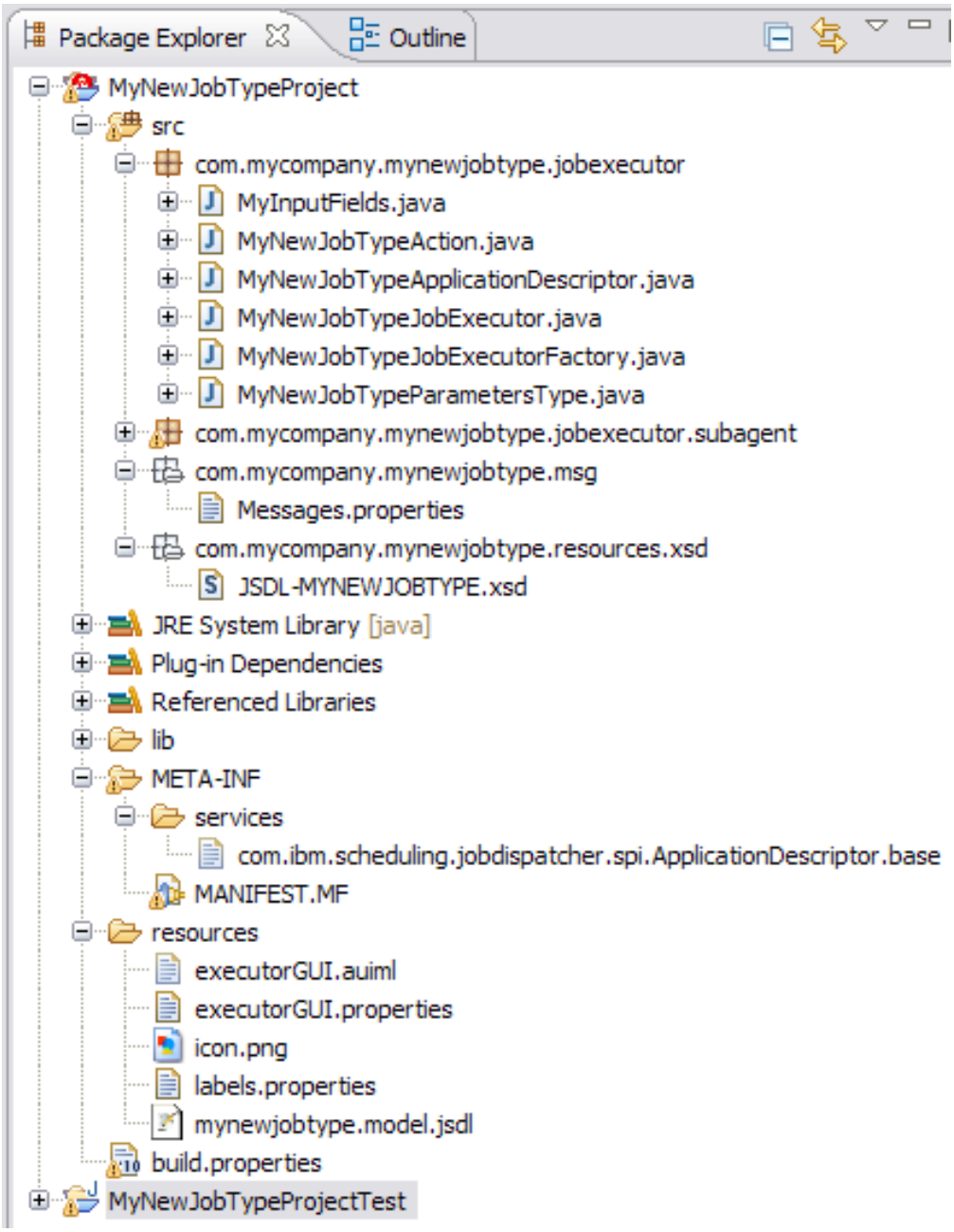
   donde `id_plug-in` es el nombre del plug-in en minúsculas.

**Nota:** Puede instalar una versión del mismo plug-in en el gestor de dominio maestro, en el gestor de dominio dinámico, y en los agentes en los que desee ejecutar el plug-in. Si crea una versión nueva del plug-in, instálelo en el gestor de dominio maestro, en el gestor de dominio dinámico, y en los agentes y elimine la versión anterior.

## Estructura de un plug-in de tipo de trabajo personalizado

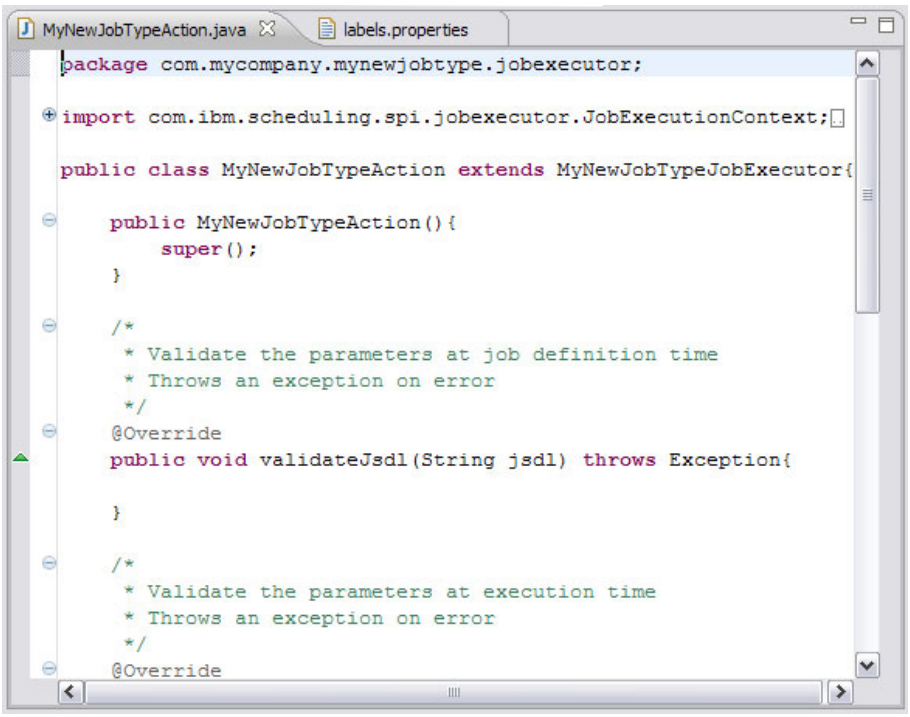
Ofrece una visión general de la estructura de un plug-in de tipo de trabajo personalizado.

Al crear un proyecto de plug-in de tipo de trabajo personalizado en Integration Workbench, verá algo similar a la imagen siguiente:




Las secciones que siguen ofrecen información relativa a cómo modificar el proyecto según sus necesidades.

**<*NombreProyecto*>Action.java**




Este archivo contiene las clases que controlan la validación de trabajos. Como puede observar en la captura de pantalla, puede codificar la validación que debe realizarse cuando se define el trabajo (habitualmente, validación semántica) y cuando se ejecuta el trabajo (por ejemplo, comprobar que un archivo existe).

<*NombreProyecto*>**ApplicationDescriptor.java**

En el descriptor de la aplicación, puede añadir manualmente la propiedad `category` para describir el tipo de plug-in.

La propiedad debe tener el ID de una categoría predefinida o un valor definido por el usuario. Los ID de las categorías de plug-in predefinidas son:

- native
- ERP
- base de datos
- file_transfer
- cloud
- system_management
- business_analytics

Si el plug-in no pertenece a ninguna de las categorías predefinidas, puede especificar una nueva categoría y se añadirá a la lista de tipos de plug-in en la sección Workload Designer de Dynamic Workload Console.

El ejemplo siguiente muestra el descriptor de aplicaciones de un plug-in que corresponde a la categoría `database`:

```
Proveedor de servicios de descriptor de aplicaciones EJB
com.ibm.scheduling.agent.database.jobexecutor.DatabaseApplicationDescriptor
application=database
factory=com.ibm.scheduling.agent.database.jobexecutor.DatabaseJobExecutor
FactorysupportedWorkstations=agent,pool,d-pool
supportedOS=UNIX,WNT,OTHER,IBM_i
isJobStoppable=true
```

```
producesJobOutput=true
category=database
schemaLocation=com/ibm/scheduling/resources/xsd/JSDL-DATABASE.xsd
namespace=http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdldatabase

version=${agent.executors.ver}
label=Database
```

**Padre de <*NombreProyecto*>Action.java**
Contiene diverso material de utilidad, como:

- El registrador
- Un método para descubrir el nombre de registro de trabajo
- La posibilidad de personalizar la etiqueta de mensaje de éxito
- La posibilidad de cambiar los códigos de salida
- Métodos para cancelar el trabajo

**com.<*NombreEmpresa*>.<*NombreProyecto*>.msg**
Contiene el archivo de propiedades al que se añaden los mensajes. Los mensajes pueden traducirse añadiendo el código de entorno local.

**JSDL-<*NOMBREPROYECTO*>.xsd**
Contiene el esquema XML del panel que ha creado. Generalmente no es necesario editar el esquema, pero puede ser un punto de referencia útil.

**MANIFEST.MF**
Este archivo registra la versión del proyecto de plug-in.

**labels.properties**
Ubicación en la que se asigna el nombre del tipo de trabajo, tal como debe visualizarse en el menú desplegable **Crear definición de trabajo ► Nueva ► Definición de trabajo** en Dynamic Workload Console. Aquí también pueden especificarse versiones traducidas, que se utilizarán automáticamente cuando se utilice la consola en sistemas que tengan establecido el entorno local correspondiente.

## Desarrollar el plug-in

Durante el proceso de desarrollo, puede decidir modificar el panel de datos de entrada. En este caso, puede utilizar la opción destinada a volver a generar el plug-in utilizando el panel modificado. Cualquier codificación que añada se tratará del siguiente modo:

- Si ha eliminado un campo del panel, debe suprimir manualmente cualquier codificación asociada que haya añadido.
- Si ha modificado un campo en el panel, debe comprobar si se sigue aplicando alguna codificación asociada y corregirlo si no es así.
- Si ha añadido campos, suministre la codificación adicional necesaria.

## Probar el plug-in

Al crear el proyecto de plug-in, se crea automáticamente un proyecto de prueba. Este contiene un archivo denominado `job.xml`, que puede lanzarse como aplicación Java para ejecutar un trabajo de Tivoli Workload Scheduler. Como alternativa, el archivo puede utilizarse como plantilla de definición de trabajo en el **compositor**.

# Capítulo 5. Definición de trabajos Java

Describe cómo crear trabajos Java que amplíen las prestaciones de Tivoli Workload Automation.

Para definir un trabajo que ejecuta un trabajo Java utilizando Dynamic Workload Console, realice el procedimiento siguiente.

1. En el árbol de navegación de la consola, expanda **Carga de trabajo** > **Diseño** y pulse **Crear definiciones de carga de trabajo**
2. Especifique un nombre de motor, distribuido o z/OS. Se abrirá el Diseñador de carga de trabajo.
3. En el panel Lista de trabajo, seleccione las definiciones de trabajo Java.

   **Nuevo** > **Base de datos e integraciones** > **Java**

   **Nuevo** > **Definición de trabajo** > **Base de datos e integraciones** > **Java**

   Las propiedades del trabajo se visualizan en el panel derecho para su edición.
4. En el panel de propiedades, especifique los atributos para la definición de trabajo que está creando. Puede encontrar información detallada sobre todos los atributos en la ayuda de contexto disponible en cada panel.
5. Pulse **Guardar** para guardar la definición de trabajo en la base de datos.

Al definir un trabajo de este tipo, debe suministrar lo siguiente:

- Un jar preparado por el usuario, que contenga las clases y métodos que desea implementar al ejecutar el trabajo
- Un conjunto de parámetros que suministren información de tiempo de ejecución al trabajo

Las secciones siguientes le indican cómo preparar el .jar. También describen cómo puede utilizar el botón **Obtener información de clase**, que se visualiza en la interfaz, para obtener información sobre las clases en el jar seleccionado.

## Crear un jar de trabajo Java

Describe cómo crear el jar utilizado con un trabajo Java.

Un trabajo Java puede utilizar cualquier jar creado según las siguientes reglas y procedimiento.

### 1. Incluya el jar de Tivoli Workload Automation en la vía de acceso de construcción

la clase que implementa la interfaz de Dynamic Workload Console se encuentra en el siguiente jar:

**En el servidor**

```
<TWA_home>/TWA/TWS/applicationJobPlugIn/
com.ibm.scheduling.agent.java_<version>.jar
```

**En un agente dinámico**

```
<TWA_home>/TWA/TWS/JavaExt/eclipse/plugins/
com.ibm.scheduling.agent.java_<version>.jar
```

donde *<version>* es la versión de Tivoli Workload Scheduler relacionado con el último fixpack que se aplica.

Este jar debe incluirse en la vía de construcción Java.

## 2. Cree la clase que implementa TWSExecutable

Cree una clase que implemente una clase denominada `TWSExecutable`, que tiene la firma siguiente:

```
public abstract interface TWSExecutable
{
  public abstract void validateParameters(Parameters paramParameters)
    throws Exception;

  public abstract void execute(Parameters paramParameters)
    throws Exception;
}
```

**Nota:** Esta clase puede utilizarse en agentes de V8.5.1 fixpack 1 o V8.6.

Esta clase implementa dos métodos:

**validateParameters**
Es el método al que se llama en primer lugar cuando se ejecuta un trabajo Java. Valida la entrada de los parámetros cuando se ha definido el trabajo. Si se produce una excepción, se graba en el registro de trabajo.

**execute**
Es el método que ejecuta realmente el trabajo.

A continuación se indican los métodos de la clase `validateParameters`:

**getParameter**
Se suministra con el argumento de uno de los parámetros y devuelve el valor real.

**getParameterList**
Devuelve una lista de todos los parámetros definidos en forma de pares nombre/valor.

**getOutputFile**
Devuelve la vía de acceso del registro de trabajo de salida.

**Nota:** Si desea grabar en el registro, debe acordarse siempre de cerrarlo.

## 3. Opcionalmente, suministre información a la persona que define el trabajo

La interfaz incluye un botón **Obtener información de clase**. Para implementar este botón, utilice la clase `TWSExecutableInformation`, que tiene la firma siguiente:

```
public abstract interface TWSExecutableInformation
{
  public abstract String getInformation();
}
```

**Nota:** Esta clase sólo puede utilizarse en V8.6.

Esta clase se implementa cuando el usuario pulsa el botón **Obtener información de clase**, por lo que puede utilizarla para suministrar información de ayuda acerca del trabajo en el jar que el usuario ha seleccionado.

## 4. Incluya las bibliotecas necesarias

Si el trabajo Java requiere bibliotecas u otros archivos, cópielos en la carpeta donde ha guardado el jar. Cuando la persona que define el trabajo identifique el jar, el producto cargará no sólo el jar, sino también todos los demás elementos de la carpeta del agente.

A continuación se muestra un ejemplo del código descrito en este tema:

```
package com.test.Trial;

import java.io.BufferedWriter;
import java.io.FileWriter;

import com.ibm.scheduling.agent.java.jobexecutor.TWSExecutable;
import com.ibm.scheduling.agent.java.jobexecutor.TWSExecutableInformation;
import com.ibm.scheduling.agent.java.parametersdomain.Parameters;

public class Trial implements TWSExecutable, TWSExecutableInformation{

  /*
   * Escribe el parámetro "parm1" en el registro
   */

 @Override
 public void execute(Parameters arg0) throws Exception {
  String parm1 = arg0.getParameter("parm1");
  String filename = arg0.getOutputFile();

  BufferedWriter out = new BufferedWriter(new FileWriter(filename));
  out.write(parm1);
  out.close();

 }

  /*
   * Valida el parámetro "parm1", lanzando la excepción en el registro
   */

 @Override
 public void validateParameters(Parameters arg0) throws Exception {
  String parm1 = arg0.getParameter("parm1");
  if(parm1.equals("XXX"))
   throw new Exception("El parámetro parm1 no es correcto");

 }

  /*
   * Indica al usuario de la interfaz que ha pulsado el botón Obtener
   * información de clase la acción que realiza la clase.
   */

 @Override
 public String getInformation() {
  String msg = "Esta clase graba el parámetro parm1 en el registro de salida";

  return msg;
 }

}
```

# Avisos

Esta información se ha desarrollado para productos y servicios que se ofrecen en EE.UU. Es posible que en otros países IBM no ofrezca los productos, los servicios o las características que se describen en este documento. Consulte con el representante local de IBM para obtener información sobre los productos y servicios actualmente disponibles en su zona. Cualquier referencia a un producto, programa o servicio de IBM no pretende afirmar ni implicar que sólo se pueda utilizar ese producto, programa o servicio de IBM. En su lugar, se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no infrinja ningún derecho de propiedad intelectual de IBM. Sin embargo, será responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio que no sea de IBM.

IBM puede tener patentes o solicitudes de patente pendientes que traten el tema que se describe en este documento. La posesión de este documento no confiere ninguna licencia sobre dichas patentes. Puede enviar consultas sobre licencias, por escrito, a:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 EE.UU.

Para realizar consultas sobre licencias referentes a información de doble byte (DBCS), puede ponerse en contacto con el Departamento de Propiedad Intelectual de IBM de su país o escribir a:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokio 103-8510, Japón

**El párrafo siguiente no es aplicable al Reino Unido ni a ningún país o región en donde tales disposiciones sean incompatibles con la legislación local**:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL", SIN GARANTÍAS DE NINGÚN TIPO, NI EXPLÍCITAS NI IMPLÍCITAS, INCLUIDAS, AUNQUE SIN LIMITARSE A ELLAS, LAS GARANTÍAS DE NO CONTRAVENCIÓN, COMERCIALIZACIÓN O ADECUACIÓN A UN PROPÓSITO DETERMINADO.

Algunos estados no permiten la exclusión de garantías expresas o implícitas en determinadas transacciones, por lo que es posible que esta declaración no sea aplicable en su caso.

Esta información podría incluir imprecisiones técnicas o errores tipográficos. Periódicamente se efectúan cambios en la información aquí contenida; dichos cambios se incorporarán a las nuevas ediciones de la publicación. IBM puede realizar mejoras y/o cambios en el(los) producto(s) y/o en el (los) programa(s) descritos en esta publicación en cualquier momento sin previo aviso.

Todas las referencias hechas en este documento a sitios web que no son de IBM se proporcionan únicamente a título informativo y no representan en modo alguno una recomendación de dichos sitios web. La información contenida en esos sitios web no forma parte de la información para el presente producto de IBM y el usuario es responsable de la utilización de esos sitios web.

IBM puede utilizar o distribuir la información que se le proporcione en la forma que considere apropiada, sin incurrir por ello en ninguna obligación para con el remitente.

Los propietarios de licencias de este programa que deseen obtener información sobre el mismo con el fin de habilitar: (i) el intercambio de información entre programas creados de forma independiente y otros programas (incluido este) y (ii) el uso mutuo de la información intercambiada, se deben poner en contacto con:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 EE.UU.

Dicha información puede estar disponible, sujeta a los términos y condiciones apropiados, incluido en algunos casos el pago de una tarifa.

IBM proporciona el programa bajo licencia descrito en esta documentación, así como todo el material bajo licencia disponible, según los términos del Acuerdo de Cliente de IBM, del Acuerdo Internacional de Programas bajo Licencia de IBM o de cualquier otro acuerdo equivalente entre ambas partes.

Este manual contiene ejemplos de datos e informes que se utilizan en operaciones comerciales diarias. Para ilustrarlos de la forma más completa posible, los ejemplos incluyen nombres de personas, empresas, marcas y productos. Todos estos nombres son ficticios y cualquier similitud con nombres y direcciones utilizados por una empresa real es totalmente fortuita.

## Marcas registradas

IBM, el logotipo de IBM e ibm.com son marcas registradas de International Business Machines Corporation en los Estados Unidos y/o en otros países. Si estos u otros términos con marca registrada de IBM aparecen por primera vez en esta información con un símbolo de marca registrada (® o ™), estos símbolos indican que son marcas registradas de EE. UU. y propiedad de IBM según la jurisprudencia relativa a marcas en el momento en el que se publicó esta información. Tales marcas registradas pueden también ser marcas registradas o de legislación común en otros países. Puede obtener una lista actual de marcas registradas de IBM en la web "Copyright and trademark information" en http://www.ibm.com/legal/copytrade.shtml.


Java y todas las marcas registradas y los logotipos basados en Java son marcas registradas o marcas comerciales registradas de Oracle y/o sus afiliados.

Microsoft y Windows son marcas registradas de Microsoft Corporation en Estados Unidos y/o en otros países.

UNIX es una marca registrada de The Open Group en los Estados Unidos y/o en otros países.

# Índice

Número de Programa: 5698-T08, 5698-WSH y 5698-WSE

Impreso en España

SC14-7623-03