

# Entwickeln mit VMware vRealize Orchestrator

vRealize Orchestrator 7.3

Dieses Dokument unterstützt die aufgeführten Produktversionen sowie alle folgenden Versionen, bis das Dokument durch eine neue Auflage ersetzt wird. Die neuesten Versionen dieses Dokuments finden Sie unter

<http://www.vmware.com/de/support/pubs>.

DE-002550-00

**vmware**<sup>®</sup>

Die neueste technische Dokumentation finden Sie auf der VMware-Website unter:

<http://www.vmware.com/de/support/>

Auf der VMware-Website finden Sie auch die aktuellen Produkt-Updates.

Falls Sie Anmerkungen zu dieser Dokumentation haben, senden Sie Ihre Kommentare und Vorschläge an:

[docfeedback@vmware.com](mailto:docfeedback@vmware.com)

Copyright © 2008–2017 VMware, Inc. Alle Rechte vorbehalten. [Informationen zu Copyright und Marken.](#)

**VMware, Inc.**

3401 Hillview Ave.  
Palo Alto, CA 94304  
[www.vmware.com](http://www.vmware.com)

**VMware Global, Inc.**

Zweigniederlassung Deutschland  
Freisinger Str. 3  
85716 Unterschleißheim/Lohhof  
Germany  
Tel.: +49 (0) 89 3706 17000  
Fax: +49 (0) 89 3706 17333  
[www.vmware.com/de](http://www.vmware.com/de)

# Inhalt

## Entwickeln mit VMware vRealize Orchestrator 9

<b>1</b>	<b>Entwickeln von Workflows</b>	<b>11</b>
	Schlüsselkonzepte von Workflows	13
	Workflowparameter	13
	Workflowattribute	14
	Workflowschema	14
	Workflowpräsentation	14
	Workflowtoken	14
	Phasen im Workflowentwicklungsprozess	15
	Best Practices für die Entwicklung von Workflows	15
	Zugriffsrechte für den Orchestrator-Client	15
	Testen von Workflows während der Entwicklung	16
	Erstellen und Bearbeiten eines Workflows	16
	Erstellen eines Workflows	16
	Bearbeiten eines Workflows	17
	Bearbeiten eines Workflows aus der Standardbibliothek	17
	Registerkarten im Workfloweditor	18
	Bereitstellung von allgemeinen Workflowinformationen	19
	Definieren von Attributen und Parametern	20
	Workflowparameter definieren	21
	Workflowattribute definieren	21
	Beschränkungen für die Namensgebung für Attribute und Parameter	22
	Workflowschema	23
	Anzeigen des Workflowschemas	24
	Erstellen eines Workflows im Workflowschema	24
	Schemaelemente	28
	Eigenschaften von Schemaelementen	31
	Verknüpfungen und Bindungen	34
	Entscheidungen	40
	Ausnahmebehandlung	43
	Verwenden von Fehlerhandlern	44
	Foreach-Elemente und zusammengesetzte Typen	45
	Hinzufügen einer Switch-Aktivität zu einem Workflow	48
	Entwickeln von Plug-Ins	49
	Überblick über Plug-Ins	49
	Inhalt und Struktur von Plug-Ins	56
	Orchestrator Plug-In-API-Referenz	61
	Elemente der Plug-In-Definitionsdatei vso.xml	72
	Best Practices zur Entwicklung von Orchestrator-Plug-Ins	89
	Abrufen der Eingabeparameter von Benutzern beim Start eines Workflows	102
	Erstellen des Dialogfelds „Eingabeparameter“ auf der Registerkarte „Präsentation“	103

Setzen von Parametereigenschaften	104
Anfordern von Benutzerinteraktionen während einer Workflowausführung	108
Hinzufügen einer Benutzerinteraktion zu einem Workflow	109
Festlegen des Attributs „security.group“	109
Festlegen des Attributs „timeout.date“ auf ein absolutes Datum	110
Berechnen einer relativen Zeitüberschreitung für Benutzerinteraktionen	111
Festlegen des Attributs „timeout.date“ auf ein relatives Datum	113
Definieren der externen Eingaben für eine Benutzerinteraktion	113
Definieren des Ausnahmeverhaltens bei der Benutzerinteraktion	114
Erstellen des Dialogfelds „Eingabeparameter“ für eine Benutzerinteraktion	115
Antworten auf eine Anforderung für eine Benutzerinteraktion	116
Aufrufen von Workflows innerhalb von Workflows	117
Workflowelemente, die Workflows aufrufen	118
Synchrones Aufrufen eines Workflows	120
Asynchrones Aufrufen eines Workflows	121
Planen eines Workflows	122
Voraussetzungen für das Abrufen eines Remoteworkflows aus einem anderen Workflow	123
Gleichzeitiges Aufrufen mehrerer Workflows	124
Ausführen eines Workflows für eine Auswahl von Objekten	125
Implementieren der Workflows „Workflows nacheinander starten“ und „Workflows parallel starten“	126
Entwickeln von Workflows mit langer Ausführungsdauer	127
Setzen eines relativen Zeitpunkts oder Datums für Timer-basierte Workflows	128
Erstellen eines Timer-basierten Workflows mit langer Ausführungsdauer	129
Erstellen eines Auslöseobjekts	130
Erstellen eines auslöserbasierten Workflows mit langer Ausführungszeit	132
Konfigurationselemente	133
Erstellen von Konfigurationselementen	133
Benutzerberechtigungen für Workflows	134
Setzen der Benutzerberechtigungen für einen Workflow	135
Validieren von Workflows	135
Validieren eines Workflows und Behebung von Validierungsfehlern	136
Debuggen von Workflows	137
Debuggen eines Workflows	137
Beispiel: Workflow-Debugging	138
Ausführen von Workflows	139
Ausführen eines Workflows im Workfloweditor	139
Ausführen eines Workflows	140
Fortsetzen einer fehlgeschlagenen Workflowausführung	141
Festlegen des Verhaltens für das Fortsetzen einer fehlgeschlagenen Workflowausführung	141
Festlegen von benutzerdefinierten Eigenschaften zur Fortsetzung fehlgeschlagener Workflowausführungen	142
Fortsetzen einer fehlgeschlagenen Workflowausführung	142
Generieren einer Workflow-Dokumentation	143
Verwenden des Workflowversionsverlaufs	143
Wiederherstellen von gelöschten Workflows	144
Entwickeln eines einfachen Beispielworkflows	144
Erstellen des Beispiels für einen einfachen Workflow	146
Erstellen Sie das Schema des Beispiels für einen einfachen Workflow	147

Erstellen der Zonen des Beispiels für einen einfachen Workflow	149
Definieren der Parameter für das einfache Workflowbeispiel	151
Definieren der Entscheidungsbindungen im einfachen Workflowbeispiel	152
Binden der Aktionselemente für das einfache Workflowbeispiel	152
Binden der skriptfähigen Aufgabenelemente für das einfache Workflowbeispiel	155
Definieren der Ausnahmebindungen im einfachen Workflowbeispiel	163
Setzen der Schreibschutzeigenschaften für Attribute des Beispiels für einen einfachen Workflow	164
Setzen der Parametereigenschaften des Beispiels für einen einfachen Workflow	165
Setzen des Layouts für das Dialogfeld der Eingabeparameter des Beispiels für einen komplexen Workflow	166
Validieren und Ausführen des Beispiels für einen einfachen Workflow	167
Entwickeln eines komplexen Workflows	169
Erstellen des Beispiels für einen komplexen Workflow	170
Erstellen einer benutzerdefinierten Aktion für das Beispiel eines komplexen Workflows	171
Erstellen des Schemas für das Beispiel eines komplexen Workflows	172
Erstellen der Zonen für das Beispiel eines komplexen Workflows	174
Definieren der Parameter für das komplexe Workflowbeispiel	176
Definieren der Bindungen für das komplexe Workflowbeispiel	176
Setzen der Attributeigenschaften im Beispiel für einen komplexen Workflow	187
Erstellen des Layouts der Eingabeparameter für das Beispiel eines komplexen Workflows	187
Validieren und Ausführen des Beispiels für einen komplexen Workflow	188

## 2 Skripterstellung 191

Orchestrator-Elemente, für die eine Skripterstellung erforderlich ist	192
Einschränkungen der Mozilla Rhino-Implementierung in Orchestrator	192
Verwenden der Orchestrator-API zur Skripterstellung	193
Zugriff auf das Skriptmodul über den Workfloweditor	194
Zugriff auf das Skriptmodul über den Aktions- oder Richtlinieneditor	194
Zugriff auf den Explorer der Orchestrator-API	195
Verwenden Sie den Orchestrator-API-Explorer, um Objekte zu finden.	195
Schreiben von Skripten	196
Hinzufügen von Parametern zu Skripten	197
Zugreifen auf das Orchestrator-Server-Datensystem über JavaScript und Workflows	198
Zugreifen auf Java-Klassen über JavaScript	199
Zugreifen auf Betriebssystembefehle über JavaScript	199
Verwenden von XPath-Ausdrücken mit dem vCenter Server -Plug-In	199
Verwenden von XPath-Ausdrücken mit dem vCenter Server -Plug-In	200
Richtlinien für Ausnahmebehandlung	200
JavaScript-Beispiele für Orchestrator	201
Allgemeine Skriptbeispiele	202
E-Mail-Skriptbeispiele	204
Datensystem-Skriptbeispiele	205
LDAP-Skriptstellungsbeispiele	205
Protokollierung-Skriptstellungsbeispiele	206
Netzwerk-Skriptstellungsbeispiele	206
Beispiele für Workflow-Skripterstellung	206

<b>3</b>	<b>Entwicklungsaktionen</b>	<b>209</b>
	Wiederverwenden von Aktionen	209
	Zugriff auf die Ansicht „Aktionen“	210
	Komponenten der Ansicht „Aktionen“	210
	Erstellen von Aktionen	210
	Erstellen einer Aktion	211
	Suchen nach Elementen, die eine Aktion implementieren	211
	Richtlinien für Aktionscodierung	212
	Verwenden des Aktions-Versionsverlaufs	213
	Wiederherstellen von gelöschten Aktionen	214
<b>4</b>	<b>Erstellen von Ressourcenelementen</b>	<b>215</b>
	Anzeigen eines Ressourcenelements	215
	Importieren eines externen Objekts zur Verwendung als Ressourcenelement	216
	Bearbeiten der Ressourcenelementinformationen und Zugriffsrechte	216
	Hinzufügen eines Ressourcenelements zu einer Datei	217
	Aktualisieren von Ressourcenelementen	218
	Hinzufügen eines Ressourcenelements zu einem Workflow	218
<b>5</b>	<b>Erstellen von Paketen</b>	<b>221</b>
	Erstellen eines Pakets	221
	Setzen der Benutzerberechtigungen für ein Paket	222
<b>6</b>	<b>Entwickeln von Plug-Ins</b>	<b>225</b>
	Überblick über Plug-Ins	225
	Struktur eines Orchestrator-Plug-Ins	226
	Verfügbarmachen einer externen API in Orchestrator	228
	Komponenten eines Plug-Ins	228
	Rolle der Datei vso.xml	229
	Rollen des Plug-In-Adapters	230
	Rollen der Plug-In-Factory	231
	Rolle von Finder-Objekten	231
	Rolle von Skriptobjekten	232
	Rolle von Ereignishandlern	232
	Inhalt und Struktur von Plug-Ins	233
	Definieren der Anwendungszuweisung in der Datei vso.xml	234
	Format der Plug-In-Definitionsdatei vso.xml	234
	Benennen von Plug-In-Objekten	235
	Benennungskonventionen für Plug-In-Objekte	236
	Dateistruktur des Plug-Ins	237
	Orchestrator Plug-In-API-Referenz	237
	IAop-Schnittstelle	237
	IDynamicFinder-Schnittstelle	238
	IPluginAdaptor-Schnittstelle	238
	IPluginEventPublisher-Schnittstelle	239
	IPluginFactory-Schnittstelle	240
	IPluginNotificationHandler-Schnittstelle	241
	IPluginPublisher-Schnittstelle	242

WebConfigurationAdaptor-Schnittstelle	242
PluginTrigger-Klasse	242
PluginWatcher-Klasse	243
QueryResult-Klasse	244
SDKFinderProperty-Klasse	245
PluginExecutionException-Klasse	246
PluginOperationException-Klasse	246
Enumeration „HasChildrenResult“	246
ScriptingAttribute-Anmerkungstyp	247
ScriptingFunction-Anmerkungstyp	247
ScriptingParameter-Anmerkungstyp	248
Elemente der Plug-In-Definitionsdatei vso.xml	248
Modulelement	248
Element „description“	249
Element „deprecated“	249
URL-Element	250
installation-Element	250
Element „action“	250
Element „finder-datasources“	251
Element „finder-datasource“	251
inventory-Element	252
Element „finders“	253
Element „finder“	253
Eigenschaftenelement	254
Eigenschaftselement	254
relations-Element	255
relation-Element	255
ID-Element	256
inventory-children-Element	256
relation-link-Element	256
Element „events“	257
Element „trigger“	257
trigger-properties-Element	257
trigger-property-Element	257
Element „gauge“	258
scripting-objects-Element	258
Objektelement	259
Element „constructors“	259
Element „constructor“	260
Element „parameters“ des Konstruktors	260
Konstruktor-Parameterelement	260
Element „attributes“	260
Element „attribute“	261
Methodenelement	261
Methodenelement	262
Element „example“	262
Element „code“	263
Methodenparameterelement	263
Methodenparameterelement	263

Singleton-Element	264
Element „enumerations“	264
Element „enumeration“	264
Element „entries“	265
Element „entry“	265
Best Practices zur Entwicklung von Orchestrator-Plug-Ins	265
Methoden zum Erstellen eines Orchestrator-Plug-Ins	266
Typen von Orchestrator-Plug-Ins	268
Plug-In-Implementierung	271
Empfehlungen zur Entwicklung von Orchestrator-Plug-Ins	275
Dokumentierung von Plug-In-Benutzeroberflächen-Zeichenfolgen und -APIs	278
<b>7 Erstellen von Plug-Ins mit Maven</b>	<b>281</b>
Erstellen eines Orchestrator Plug-Ins mit Maven aus einem Archetyp	281
Maven-Archetypen	282
Best Practices für die Maven-basierte Plug-In-Entwicklung	282
<b>Index</b>	<b>285</b>



# Entwickeln mit VMware vRealize Orchestrator

---

*Entwickeln mit VMware vRealize Orchestrator* stellt Informationen und Anweisungen für die Entwicklung benutzerdefinierter VMware® vRealize Orchestrator-Workflows und -Aktionen bereit.

Darüber hinaus enthält die Dokumentation Informationen zu Orchestrator-Elementen, für die Skripte erforderlich sind, und stellt JavaScript-Beispiele bereit. *Entwickeln mit VMware vRealize Orchestrator* bietet außerdem eine Anleitung zum Erstellen von Ressourcen und Paketen.

## Zielgruppe

Diese Informationen richten sich an Entwickler, die benutzerdefinierte Orchestrator-Workflows und -Aktionen sowie benutzerdefinierte Bausteine erstellen möchten.



# Entwickeln von Workflows

---

Sie entwickeln Workflows in der Orchestrator-Client-Schnittstelle. Bei der Workflowentwicklung verwenden Sie den Workfloweditor, das integrierte Mozilla Rhino JavaScript-Skriptmodul und die Orchestrator- und vCenter Server-APIs.

- [Schlüsselkonzepte von Workflows](#) auf Seite 13

Workflows bestehen aus einem Schema, Attributen und Parametern. Das Workflowschema ist die Hauptkomponente eines Workflows, da in ihm alle Workflow-Elemente und die logischen Verbindungen zwischen diesen definiert werden. Die Workflow-Attribute und -Parameter sind die Variablen, die Workflows zum Übertragen von Daten verwenden. Orchestrator speichert bei jeder Ausführung eines Workflows einen Workflow-Token, in dem die Details dieser speziellen Ausführung des Workflows aufgezeichnet werden.

- [Phasen im Workflowentwicklungsprozess](#) auf Seite 15

Der Prozess zur Entwicklung eines Workflows umfasst eine Serie von Phasen. Sie können den Phasen in einer anderen Reihenfolge folgen oder eine Phase überspringen. Dies hängt weitgehend von der Art des Workflows ab, den Sie entwickeln. Sie können beispielsweise einen Workflow ohne benutzerdefinierte Skripterstellung erstellen.

- [Best Practices für die Entwicklung von Workflows](#) auf Seite 15

VMware empfiehlt verschiedene Best Practices für die Entwicklung von Orchestrator-Workflows durch mehrere Benutzer und in einer geclusterten Umgebung.

- [Zugriffsrechte für den Orchestrator-Client](#) auf Seite 15

Standardmäßig können nur Mitglieder der Orchestrator-Administrator-LDAP-Gruppe auf den Orchestrator-Client zugreifen.

- [Testen von Workflows während der Entwicklung](#) auf Seite 16

Sie können Workflows zu jedem Zeitpunkt während des Entwicklungsprozesses testen, auch wenn Sie den Workflow nicht fertiggestellt oder kein Endelement eingeschlossen haben.

- [Erstellen und Bearbeiten eines Workflows](#) auf Seite 16

Sie erstellen Workflows im Orchestrator-Client und bearbeiten sie im Workfloweditor. Der Workfloweditor ist die Entwicklungsumgebung (IDX) des Orchestrator-Clients für das Entwickeln von Workflows.

- [Bereitstellung von allgemeinen Workflowinformationen](#) auf Seite 19

Sie stellen einen Namen und eine Beschreibung für den Workflow bereit, definieren Attribute und bestimmte Aspekte des Verhaltens des Workflows, legen die Versionsnummer fest, prüfen die Signatur und bestimmen die Benutzerberechtigungen in der Registerkarte **Allgemein** im Workfloweditor.

- [Definieren von Attributen und Parametern](#) auf Seite 20  
Nach dem Erstellen eines Workflows müssen Sie seine globalen Attribute, Eingabeparameter und Ausgabeparameter definieren.
- [Workflowschema](#) auf Seite 23  
Ein Workflowschema ist eine grafische Darstellung eines Workflows, die ihn als Flussdiagramm von miteinander verknüpften Workflowelementen zeigt. Das Workflowschema definiert den logischen Fluss eines Workflows.
- [Entwickeln von Plug-Ins](#) auf Seite 49  
Orchestrator ermöglicht dank seiner offenen Plug-In-Architektur die Integration in Management- und Verwaltungslösungen. Sie nutzen den Orchestrator-Client zum Ausführen und Erstellen von Plug-In-Workflows und zum Zugriff auf die Plug-In-API.
- [Abrufen der Eingabeparameter von Benutzern beim Start eines Workflows](#) auf Seite 102  
Wenn ein Workflow Eingabeparameter erfordert, öffnet er ein Dialogfeld, in dem Benutzer die erforderlichen Eingabeparameterwerte zum Ausführen eingeben. Sie können den Inhalt und das Layout oder die Präsentation dieses Dialogfelds auf der Registerkarte **Präsentation** im Workfloweditor organisieren.
- [\(Optional\) Anfordern von Benutzerinteraktionen während einer Workflowausführung](#) auf Seite 108  
Manchmal benötigt ein Workflow während seiner Ausführung zusätzliche Eingabeparameter aus einer externen Quelle. Diese Eingabeparameter können aus einer anderen Anwendung oder einem anderen Workflow stammen, oder der Benutzer kann sie direkt angeben.
- [Aufrufen von Workflows innerhalb von Workflows](#) auf Seite 117  
Workflows können während ihrer Ausführung andere Workflows aufrufen. Ein Workflow kann einen anderen Workflow entweder starten, weil er das Ergebnis des anderen Workflows als Eingabeparameter für seine eigene Ausführung benötigt, oder er kann einen Workflow starten und seine Ausführung unabhängig fortsetzen lassen. Workflows können einen Workflow auch zu einem bestimmten zukünftigen Zeitpunkt starten oder mehrere Workflows gleichzeitig starten.
- [Ausführen eines Workflows für eine Auswahl von Objekten](#) auf Seite 125  
Sie können sich wiederholende Aufgaben durch Ausführen eines Workflows für eine Auswahl von Objekten automatisieren. Beispiele: Sie erstellen einen Workflow, der Snapshots aller virtuellen Maschinen in einem Ordner mit virtuellen Maschinen erstellt, oder Sie erstellen einen Workflow, der alle virtuellen Maschinen eines bestimmten Hosts einschaltet.
- [Entwickeln von Workflows mit langer Ausführungsdauer](#) auf Seite 127  
Ein Workflow in einem Wartezustand verbraucht Systemressourcen, da er ständig das Objekt abruft, von der er eine Antwort benötigt. Wenn Sie wissen, dass ein Workflow möglicherweise lange Zeit wartet, bevor er die erforderliche Antwort erhält, können Sie ihm Workflowelemente mit langer Ausführungsdauer hinzufügen.
- [Konfigurationselemente](#) auf Seite 133  
Ein Konfigurationselement ist eine Liste von Attributen, mit deren Hilfe Sie Konstanten über eine gesamte Orchestrator-Serverbereitstellung hinweg konfigurieren können.
- [Benutzerberechtigungen für Workflows](#) auf Seite 134  
Orchestrator definiert Berechtigungsebenen, die Sie auf Gruppen anwenden können, um Ihnen den Zugriff auf Workflows zu gestatten oder zu verweigern.
- [Validieren von Workflows](#) auf Seite 135  
Orchestrator bietet ein Tool zur Workflowvalidierung. Durch das Validieren eines Workflows können Sie Fehler im Workflow erkennen und überprüfen, ob die Daten von einem Element zum nächsten korrekt fließen.

- [Debuggen von Workflows](#) auf Seite 137  
Orchestrator bietet ein Tool zum Debuggen von Workflows. Sie können einen Workflow debuggen, um Ein- und Ausgabeparameter und Attribute beim Starten von Aktivitäten zu überprüfen, Parameter- oder Attributwerte bei Ausführung eines Workflows im Bearbeitungsmodus ersetzen und Workflows ab der letzten fehlgeschlagenen Aktivität fortsetzen.
- [Ausführen von Workflows](#) auf Seite 139  
Ein Orchestrator-Workflow läuft nach einem logischen Ereignisfluss ab.
- [Fortsetzen einer fehlgeschlagenen Workflowausführung](#) auf Seite 141  
Falls ein Workflow fehlschlägt, bietet Orchestrator eine Option, den Workflow ab der letzten fehlgeschlagenen Aktivität fortzusetzen.
- [Generieren einer Workflow-Dokumentation](#) auf Seite 143  
Ein ausgewählter Workflow oder Workflow-Ordner kann jederzeit als Dokumentation im PDF-Format exportiert werden.
- [Verwenden des Workflowversionsverlaufs](#) auf Seite 143  
Mithilfe des Versionsverlaufs können Sie einen Workflow auf einen früher gespeicherten Zustand zurücksetzen. Sie können den Workflowzustand auf eine frühere oder spätere Workflowversion zurücksetzen. Weiterhin können Sie die Unterschiede zwischen dem aktuellen Workflowzustand und einer gespeicherten früheren Version des Workflows vergleichen.
- [Wiederherstellen von gelöschten Workflows](#) auf Seite 144  
Sie können Workflows wiederherstellen, die aus der Workflow-Bibliothek gelöscht wurden.
- [Entwickeln eines einfachen Beispielworkflows](#) auf Seite 144  
Das Entwickeln eines einfachen Beispielworkflows zeigt die häufigsten Schritte im Entwicklungsprozess eines Workflows.
- [Entwickeln eines komplexen Workflows](#) auf Seite 169  
Das Entwickeln eines komplexen Beispielworkflows zeigt die häufigsten Schritte im Entwicklungsprozess eines Workflows und erweiterte Szenarien, etwa das Erstellen von benutzerdefinierten Entscheidungen und Schleifen.

## Schlüsselkonzepte von Workflows

Workflows bestehen aus einem Schema, Attributen und Parametern. Das Workflowschema ist die Hauptkomponente eines Workflows, da in ihm alle Workflow-Elemente und die logischen Verbindungen zwischen diesen definiert werden. Die Workflow-Attribute und -Parameter sind die Variablen, die Workflows zum Übertragen von Daten verwenden. Orchestrator speichert bei jeder Ausführung eines Workflows einen Workflow-Token, in dem die Details dieser speziellen Ausführung des Workflows aufgezeichnet werden.

### Workflowparameter

Workflows erhalten Eingabeparameter und generieren Ausgabeparameter, wenn sie ausgeführt werden.

#### Eingabeparameter

Die meisten Workflows benötigen zur Ausführung einen bestimmten Satz an Eingabeparametern. Ein Eingabeparameter ist ein Argument, das der Workflow beim Starten verarbeitet. Der Benutzer, eine Anwendung oder ein anderer Workflow bzw. eine Aktion übergibt Eingabeparameter an einen Workflow, die dieser verarbeitet, wenn er gestartet wird.

Wenn beispielsweise ein Workflow eine virtuelle Maschine zurücksetzt, benötigt der Workflow als Eingabeparameter den Namen der virtuellen Maschine.

## Ausgabeparameter

Die Ausgabeparameter eines Workflows stellen das Ergebnis der Workflow-Ausführung dar. Die Ausgabeparameter können sich ändern, wenn ein Workflow oder ein Workflow-Element ausgeführt wird. Während ihrer Ausführung können Workflows die Ausgabeparameter anderer Workflows als Eingabeparameter empfangen.

Wenn beispielsweise ein Workflow einen Snapshot einer virtuellen Maschine erstellt, ist der Ausgabeparameter des Workflows der erstellte Snapshot.

## Workflowattribute

Workflowelemente verarbeiten von Ihnen empfangenen Daten als Eingabeparameter und setzen die sich daraus ergebenden Daten als Workflowattribute oder Ausgabeparameter.

Schreibgeschützte Workflowattribute agieren als globale Konstanten für den Workflow. Beschreibbare Attribute agieren als globale Variablen eines Workflows.

Sie können Attribute verwenden, um Daten zwischen den Elementen eines Workflows zu transportieren. Sie können Attribute auf folgende Arten beziehen:

- Definieren von Attributen beim Erstellen eines Workflows
- Setzen des Ausgabeparameters eines Workflowelements als Workflowattribut
- Arten von Attributen aus einem Konfigurationselement

## Workflowschema

Ein Workflowschema ist eine grafische Darstellung, die den Workflow als Flussdiagramm von miteinander verknüpften Workflowelementen zeigt. Das Workflowschema ist das wichtigste Element eines Workflows, da es seine Logik festlegt.

## Workflowpräsentation

Wenn Benutzer einen Workflow ausführen, geben sie die Werte für die Eingabeparameter des Workflows in der Workflowpräsentation an. Wenn Sie die Workflowpräsentation organisieren, berücksichtigen Sie den Typ und die Anzahl der Eingabeparameter des Workflows.

## Workflowtoken

Ein Workflowtoken stellt einen Workflow dar, der ausgeführt wird oder wurde.

Ein Workflow ist eine abstrakte Beschreibung eines Vorgangs, der durch eine generische Folge von Schritten und einen generischen Satz erforderlicher Eingabeparameter definiert wird. Wenn Sie einen Workflow mit einem Satz echter Eingabeparameter ausführen, erhalten Sie eine Instanz dieses abstrakten Workflows, die sich entsprechend der spezifischen angegebenen Eingabeparameter verhält. Diese spezifische Instanz eines abgeschlossenen oder ausgeführten Workflows wird Workflowtoken genannt.

## Workflowtoken-Attribute

Workflowtoken-Attribute sind die spezifischen Parameter, mit denen ein Workflowtoken ausgeführt wird. Die Workflowtoken-Attribute sind eine Zusammenfassung der globalen Attribute des Workflows und der spezifischen Eingabe- und Ausgabeparameter, mit denen Sie das Workflowtoken ausführen.

## Phasen im Workflowentwicklungsprozess

Der Prozess zur Entwicklung eines Workflows umfasst eine Serie von Phasen. Sie können den Phasen in einer anderen Reihenfolge folgen oder eine Phase überspringen. Dies hängt weitgehend von der Art des Workflows ab, den Sie entwickeln. Sie können beispielsweise einen Workflow ohne benutzerdefinierte Skripterstellung erstellen.

Im Allgemeinen durchläuft die Entwicklung eines Workflows folgende Phasen:

- 1 Erstellen Sie einen neuen Workflow oder ein Duplikat eines bestehenden Workflows aus der Standardbibliothek.
- 2 Geben Sie allgemeine Informationen über den Workflow ein.
- 3 Definieren Sie die Eingabeparameter des Workflows.
- 4 Entwerfen Sie das Workflowschema und verknüpfen Sie es, um den logischen Fluss im Workflow zu definieren.
- 5 Binden Sie die Eingabe- und Ausgabeparameter jedes Schemaelements an Workflowattribute.
- 6 Schreiben Sie die erforderlichen Skripte für Elemente für skriptfähige Aufgaben oder benutzerdefinierte Entscheidungselemente.
- 7 Erstellen Sie die Workflowpräsentation, um das Layout des Dialogfelds für die Eingabeparameter zu definieren, über das die Benutzer Eingaben für den Workflow vornehmen.
- 8 Validieren Sie den Workflow.

## Best Practices für die Entwicklung von Workflows

VMware empfiehlt verschiedene Best Practices für die Entwicklung von Orchestrator-Workflows durch mehrere Benutzer und in einer geclusterten Umgebung.

- Jeder Entwickler verfügt zu Testzwecken über eine dedizierte eigenständige Orchestrator-Instanz, in der er Workflows erstellen und entwickeln kann.
- Workflows werden als Maven-Projekte auf einem Kontrollsystem mit gemeinsam genutztem Quellcode gespeichert.
- Um eine optimale Leistung der Orchestrator-Produktionsbereitstellung zu erzielen, sollten Workflows in einem geplanten Zeitfenster importiert werden.
- Beim Importieren von Workflows in einen Orchestrator-Cluster verbinden Sie den Orchestrator-Client mit einem der Knoten. Hierbei verwenden Sie nicht die Adresse des virtuellen Lastausgleichsservers, sondern den jeweiligen lokalen Hostnamen oder die IP-Adresse.

---

**HINWEIS** Alle Änderungen eines Workflows werden bei der nächsten Ausführung des Workflows wirksam.

---

## Zugriffsrechte für den Orchestrator-Client

Standardmäßig können nur Mitglieder der Orchestrator-Administrator-LDAP-Gruppe auf den Orchestrator-Client zugreifen.

Der Orchestrator-Administrator kann anderen Benutzergruppen Zugriff auf den Orchestrator-Client gewähren, indem er zumindest die **Anzeige**-Berechtigungen festlegt.

Damit Sie auf den Orchestrator-Client zugreifen können, muss der Administrator Sie entweder der Orchestrator-Administrator-LDAP-Gruppe hinzufügen oder **Anzeige**, **Überprüfen**, **Bearbeiten**, **Ausführen** oder **Admin**-Berechtigungen für eine Gruppe, in der Sie bereits Mitglied sind, festlegen.

## Testen von Workflows während der Entwicklung

Sie können Workflows zu jedem Zeitpunkt während des Entwicklungsprozesses testen, auch wenn Sie den Workflow nicht fertiggestellt oder kein Endelement eingeschlossen haben.

Standardmäßig prüft Orchestrator, ob ein Workflow gültig ist, bevor dieser ausgeführt werden kann. Sie können die automatische Validierung während der Workflowentwicklung deaktivieren, um Teilworkflows für Testzwecke auszuführen.

---

**HINWEIS** Vergessen Sie nicht, die automatische Validierung wieder zu aktivieren, wenn Sie mit der Entwicklung des Workflows fertig sind.

---

### Vorgehensweise

- 1 Klicken Sie im Menü des Orchestrator-Clients auf **Extras > Benutzereinstellungen**.
- 2 Klicken Sie auf die Registerkarte **Workflows**.
- 3 Heben Sie die Auswahl des Kontrollkästchens **Workflow vor dem Ausführen validieren** auf.

Sie haben die automatische Workflowvalidierung deaktiviert.

## Erstellen und Bearbeiten eines Workflows

Sie erstellen Workflows im Orchestrator-Client und bearbeiten sie im Workfloweditor. Der Workfloweditor ist die Entwicklungsumgebung (IDX) des Orchestrator-Clients für das Entwickeln von Workflows.

Sie öffnen den Workfloweditor, indem Sie einen bestehenden Workflow bearbeiten.

- [Erstellen eines Workflows](#) auf Seite 16  
Sie können Workflows in der hierarchischen Liste für Workflows im Orchestrator-Client ausführen.
- [Bearbeiten eines Workflows](#) auf Seite 17  
Sie bearbeiten einen Workflow, um Änderungen an einem vorhandenen Workflow vorzunehmen oder einen neuen leeren Workflow zu entwickeln.
- [Bearbeiten eines Workflows aus der Standardbibliothek](#) auf Seite 17  
Orchestrator stellt eine Standardbibliothek mit Workflows bereit, die zum Automatisieren von Vorgängen in der virtuellen Infrastruktur genutzt werden können. Die Workflows in der Standardbibliothek sind schreibgeschützt.
- [Registerkarten im Workfloweditor](#) auf Seite 18  
Der Workfloweditor besteht aus Registerkarten, auf denen Sie die Komponenten des Workflows bearbeiten können.

### Erstellen eines Workflows

Sie können Workflows in der hierarchischen Liste für Workflows im Orchestrator-Client ausführen.

#### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Workflows**.
- 3 (Optional) Klicken Sie mit der rechten Maustaste auf das Stammelement der hierarchischen Liste der Workflows oder einen Ordner in der Liste und wählen Sie **Neuer Ordner**, um einen neuen Workflowordner zu erstellen.
- 4 (Optional) Geben Sie den Namen des neuen Ordners ein.



- 5 Klicken Sie mit der rechten Maustaste auf den neuen Ordner oder einen bestehenden Ordner und wählen Sie **Neuer Workflow**.
- 6 Geben Sie dem neuen Workflow einen Namen und klicken Sie auf **OK**.

Ein neuer, leerer Workflow wird in dem von Ihnen ausgewählten Ordner erstellt.

### Weiter

Sie können den Workflow bearbeiten.

## Bearbeiten eines Workflows

Sie bearbeiten einen Workflow, um Änderungen an einem vorhandenen Workflow vorzunehmen oder einen neuen leeren Workflow zu entwickeln.

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Workflows**.
- 3 Erweitern Sie die hierarchische Workflowliste und navigieren Sie zu dem Workflow, den Sie bearbeiten möchten.
- 4 Um den Workflow für die Bearbeitung zu öffnen, klicken Sie mit der rechten Maustaste auf den Workflow und wählen Sie **Bearbeiten**.

Der Workfloweditor wird der Workflow zur Bearbeitung geöffnet.

## Bearbeiten eines Workflows aus der Standardbibliothek

Orchestrator stellt eine Standardbibliothek mit Workflows bereit, die zum Automatisieren von Vorgängen in der virtuellen Infrastruktur genutzt werden können. Die Workflows in der Standardbibliothek sind schreibgeschützt.

Erstellen Sie zum Bearbeiten eines Workflows aus der Standardbibliothek ein Duplikat des Workflows. Sie können duplizierte Workflows oder benutzerdefinierte Workflows bearbeiten.

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Workflows**.
- 3 (Optional) Klicken Sie mit der rechten Maustaste auf das Stammelement der hierarchischen Liste der Workflows und wählen Sie **Neuer Ordner**, um einen Ordner mit dem zu bearbeitenden Workflow zu erstellen.
- 4 Erweitern Sie die hierarchische Liste der Standardworkflows unter **Bibliothek**, um zu dem zu bearbeitenden Workflow zu navigieren.
- 5 Klicken Sie mit der rechten Maustaste auf den zu bearbeitenden Workflow.  
Die Option **Bearbeiten** ist abgeblendet. Der Workflow ist schreibgeschützt.
- 6 Klicken Sie mit der rechten Maustaste auf den Workflow und wählen Sie **Workflow duplizieren**.
- 7 Geben Sie einen Namen für den duplizierten Workflow ein.

Standardmäßig gibt Orchestrator dem duplizierten Workflow den Namen *Copy of Workflow\_Name*.

- 8 Klicken Sie auf den Wert **Workflow-Ordner**, um nach einem Ordner zum Speichern des duplizierten Workflows zu suchen.

Wählen Sie den Ordner aus, den Sie in [Schritt 3](#) erstellt haben. Wenn Sie keinen Ordner erstellt haben, wählen Sie einen Ordner aus, der nicht in der Bibliothek der Standardworkflows enthalten ist.

- 9 Klicken Sie auf **Ja** oder **Nein**, um den Versionsverlauf des Workflows für das Duplikat zu übernehmen.

Option	Beschreibung
<b>Ja</b>	Der Versionsverlauf des ursprünglichen Workflows wird im Duplikat repliziert.
<b>Nein</b>	Die Version des Duplikats wird auf 0.0.0 zurückgesetzt.

- 10 Klicken Sie auf **Duplizieren**, um den Workflow zu duplizieren.

- 11 Klicken Sie mit der rechten Maustaste auf das Duplikat und wählen Sie **Bearbeiten**.

Der Workfloweditor wird geöffnet. Sie können den duplizierten Workflow bearbeiten.

Sie haben einen Workflow aus der Standardbibliothek dupliziert. Sie können den duplizierten Workflow bearbeiten.

## Registerkarten im Workfloweditor

Der Workfloweditor besteht aus Registerkarten, auf denen Sie die Komponenten des Workflows bearbeiten können.

**Tabelle 1-1.** Registerkarten im Workfloweditor

Registerkarte	Beschreibung
<b>Allgemein</b>	Beachten Sie den Workflownamen, geben Sie eine Beschreibung der Workflowaktion ein, legen Sie die Versionsnummer fest, bestimmen Sie die Benutzerberechtigungen, definieren Sie das Verhalten des Workflows, wenn der Orchestrator-Server neu startet, und definieren Sie die globalen Attribute des Workflows.
<b>Eingaben</b>	Definieren Sie die Parameter, die der Workflow beim Ausführen benötigt. Diese Eingabeparameter sind die Daten, die der Workflow verarbeitet. Das Verhalten des Workflows verändert sich gemäß diesen Parametern.
<b>Ausgaben</b>	Definieren Sie die Werte, die der Workflow generiert, wenn er seinen Ausführungsvorgang abschließt. Andere Workflows oder Aktionen können diese Werte für ihr eigenes Ausführen verwenden.
<b>Schema</b>	Erstellen Sie den Workflow. Sie haben den Workflow erstellt, indem Sie Workflowschemaelemente aus der Workflowpalette auf der linken Seite der Registerkarte <b>Schema</b> gezogen haben. Durch Klicken auf ein Element im Schemadiagramm können Sie das Verhalten des Elements in der unteren Hälfte der Registerkarte <b>Schema</b> definieren und bearbeiten.
<b>Präsentation</b>	Definieren Sie das Layout des Dialogfelds für Benutzereingaben, das erscheint, wenn Benutzer den Workflow ausführen. Sie ordnen die Parameter und Attribute in Präsentationsschritte und Gruppen, um die Erkennung von Parametern im Dialogfeld für die Eingabeparameter zu erleichtern. Sie definieren die Integritätsregeln für die Eingabeparameter, die Benutzer in der Präsentation bereitstellen können, indem Sie die Parametereigenschaften festlegen.

**Tabelle 1-1.** Registerkarten im Workfloweditor (Fortsetzung)

Registerkarte	Beschreibung
Parameterreferenzen	Zeigen Sie an, welche Workflowelemente die Attribute und Parameter im logischen Fluss des Workflows verbrauchen. Diese Registerkarte zeigt auch die Integritätsregeln für diese Parameter und Attribute, die Sie auf der Registerkarte <b>Präsentation</b> definiert haben.
Workflowtoken	Zeigen Sie Details über jeden Ausführungsvorgang eines Workflows an. Diese Informationen umfassen den Status des Workflows, den Benutzer, der ihn gestartet hat, den Geschäftsstatus des aktuellen Elements sowie Datum und Uhrzeit für Start und Ende des Workflows.
Ereignis	Zeigen Sie Informationen über jedes einzelne Ereignis an, das eintritt, wenn der Workflow ausgeführt wird. Diese Informationen umfassen eine Beschreibung des Ereignisses, den Benutzer, der es ausgelöst hat, die Art und den Ursprung des Ereignisses sowie die Zeit und das Datum ihres Eintritts.
Berechtigungen	Legen Sie die Berechtigungen für Benutzer oder Benutzergruppen für die Arbeit mit dem Workflow fest.

## Bereitstellung von allgemeinen Workflowinformationen

Sie stellen einen Namen und eine Beschreibung für den Workflow bereit, definieren Attribute und bestimmte Aspekte des Verhaltens des Workflows, legen die Versionsnummer fest, prüfen die Signatur und bestimmen die Benutzerberechtigungen in der Registerkarte **Allgemein** im Workfloweditor.

### Voraussetzungen

Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.

### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Allgemein**.
- 2 Klicken Sie auf die Zahl unter **Version**, um eine Versionsnummer für den Workflow festzulegen.  
Das Dialogfeld **Versionskommentar** wird geöffnet.
- 3 Geben Sie einen Kommentar für diese Version des Workflows ein und klicken Sie auf **OK**.  
Geben Sie beispielsweise **Ersterstellung** ein, wenn Sie den Workflow gerade erstellt haben.  
Eine neue Version des Workflows wird erstellt. Sie können den Status des Workflows später auf diese Version zurücksetzen.

- 4 Definieren Sie, wie sich der Workflow verhält, wenn der Orchestrator-Server erneut startet, indem Sie den Wert für die Option **Serverneustart-Verhalten** bestimmen.
  - Belassen Sie den Standardwert für **Ausführen des Workflows wiederaufnehmen**, damit der Workflow an der Stelle wieder aufgenommen wird, an der er beim Stoppen des Servers abgebrochen wurde.
  - Klicken Sie auf **Ausführen des Workflows wiederaufnehmen** und wählen Sie **Ausführen des Workflows nicht wiederaufnehmen (als FEHLGESCHLAGEN markieren)**, damit der Workflow beim Neustart des Orchestrator-Servers nicht erneut aufgenommen wird.

Hindern Sie den Workflow an einem Neustart, wenn der Workflow von der Umgebung abhängt, in der er ausgeführt wird. Beispiel: Wenn ein Workflow einen bestimmten vCenter Server benötigt und Sie Orchestrator neu konfigurieren, sodass er sich mit einem anderen vCenter Server verbindet, führt der Neustart des Workflows nach dem Neustart des Orchestrator-Servers zu einem Fehlschlag des Workflows.
- 5 Geben Sie eine detaillierte Beschreibung des Workflows in das Textfeld **Beschreibung** ein.
- 6 Klicken Sie unten im Workfloweditor auf **Speichern**.
 

Eine grüne Meldung links unten im Workfloweditor bestätigt, dass Sie Ihre Änderungen gespeichert haben.

Sie haben Aspekte des Workflowverhaltens definiert, die Version eingestellt und die Vorgänge definiert, die Benutzer mit dem Workflow durchführen können.

#### Weiter

Sie müssen die Attribute und Parameter des Workflows definieren.

## Definieren von Attributen und Parametern

Nach dem Erstellen eines Workflows müssen Sie seine globalen Attribute, Eingabeparameter und Ausgabeparameter definieren.

Workflowattribute speichern Daten, die von Workflows intern verarbeitet werden. Workflow-Eingabeparameter sind Daten, die von einer externen Quelle zur Verfügung gestellt werden, etwa einem Benutzer oder einem anderen Workflow. Workflow-Ausgabeparameter sind Daten, die der Workflow nach seiner Ausführung liefert.

- [Workflowparameter definieren](#) auf Seite 21  
Sie können Eingabe- und Ausgabeparameter verwenden, um Daten in den und aus dem Workflow zu holen.
- [Workflowattribute definieren](#) auf Seite 21  
Workflowattribute sind die Daten, die von Workflows verarbeitet werden.
- [Beschränkungen für die Namensgebung für Attribute und Parameter](#) auf Seite 22  
Sie können OGNL-Ausdrücke verwenden, um Eingabeparameter dynamisch während einer Workflowausführung zu ermitteln. Der Orchestrator-OGNL-Parser verwendet bestimmte Schlüsselwörter während der OGNL-Verarbeitung, die Sie nicht in Workflowattribut- oder -Parameternamen verwenden können.

## Workflowparameter definieren

Sie können Eingabe- und Ausgabeparameter verwenden, um Daten in den und aus dem Workflow zu holen.

Sie können die Parameter eines Workflows im Workfloweditor definieren. Die Eingabeparameter sind die Anfangsdaten, die für die Workflowausführung erforderlich sind. Wenn Benutzer den Workflow ausführen, geben sie die Werte für die Eingabeparameter an. Die Ausgabeparameter sind die Daten, die der Workflow nach der Ausführung zurückgibt.

### Voraussetzungen

Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.

### Vorgehensweise

- 1 Klicken Sie auf die entsprechende Registerkarte im Workfloweditor.
  - Klicken Sie auf **Eingaben**, um Eingabeparameter zu erstellen.
  - Klicken Sie auf **Ausgaben** um Ausgabeparameter zu erstellen.
- 2 Klicken Sie mit der rechten Maustaste in die Registerkarte „Parameter“ und wählen Sie **Parameter hinzufügen** aus.
- 3 Klicken Sie auf den Parameternamen, um ihn zu ändern.  
Der Standardname ist `arg_in_X` für Eingabeparameter und `arg_out_X` für Ausgabeparameter, wobei X eine Zahl ist.
- 4 (Optional) Um den Wert des Parametertyps zu ändern, klicken Sie auf den Wert und wählen einen anderen aus der Liste verfügbarer Werte aus.  
Der Wert für den Parametertyp ist standardmäßig „Zeichenfolge“.
- 5 Fügen Sie im Textfeld **Beschreibung** eine Beschreibung für den Parameter hinzu.
- 6 (Optional) Wenn Sie entscheiden, lieber ein Attribut statt einem Parameter zu verwenden, klicken Sie mit der rechten Maustaste auf den Parameter und wählen Sie **Als Attribut verschieben**, um die Änderung vorzunehmen.

Sie haben einen Eingabe- oder Ausgabeparameter für den Workflow definiert.

### Weiter

Erstellen Sie nach dem Definieren der Workflowparameter das Workflowschema.

## Workflowattribute definieren

Workflowattribute sind die Daten, die von Workflows verarbeitet werden.

---

**HINWEIS** Sie können beim Erstellen des Workflowschemas auch Workflowattribute in den Workflowschema-Elementen definieren. Es ist oft einfacher, ein Attribut zu definieren, wenn Sie das Workflowschema-Element erstellen, das es verarbeitet.

---

### Voraussetzungen

Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.

### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Allgemein**.  
Der Bereich „Attribute“ wird in der unteren Hälfte der Registerkarte **Allgemein** angezeigt.

- 2 Klicken Sie mit der rechten Maustaste in den Bereich „Attribute“ und wählen Sie **Attribut auswählen**.  
Ein neues Attribut wird in der Attributliste angezeigt, mit Zeichenfolge als Standardtyp.
- 3 Klicken Sie auf den Attributnamen, um ihn zu ändern.  
Der Standardname ist attX, wobei X eine Zahl ist.

---

**HINWEIS** Workflowattribute dürfen nicht den gleichen Namen wie ein Parameter des Workflows haben.

---

- 4 Klicken Sie auf den Attributtyp, um einen neuen Typ aus einer Liste möglicher Werte auszuwählen.  
Der Standardattributtyp ist „Zeichenfolge“.
- 5 Klicken Sie auf den Attributwert, um einen Wert entsprechend des Attributtyps festzulegen oder auszuwählen.
- 6 Fügen Sie im Textfeld **Beschreibung** eine Beschreibung des Attributs hinzu.
- 7 Wenn das Attribut eine Konstante statt einer Variable ist, klicken Sie auf das Kontrollkästchen links neben dem Attributnamen, um den Wert schreibgeschützt zu machen.  
Das Sperrsymbol identifiziert die Spalte schreibgeschützter Kontrollkästchen.
- 8 (Optional) Wenn Sie entscheiden, dass das Attribut ein Eingabe- oder Ausgabeparameter statt eines Attributs sein soll, klicken Sie mit der rechten Maustaste auf das Attribut und wählen Sie **Als EINGABE/AUSGABE-Parameter verschieben**, um das Attribut in einen Parameter zu ändern.

Sie haben ein Attribut für den Workflow definiert.

#### Weiter

Sie können die Eingabe- und Ausgabeparameter des Workflows definieren.

## Beschränkungen für die Namensgebung für Attribute und Parameter

Sie können OGNL-Ausdrücke verwenden, um Eingabeparameter dynamisch während einer Workflowausführung zu ermitteln. Der Orchestrator-OGNL-Parser verwendet bestimmte Schlüsselwörter während der OGNL-Verarbeitung, die Sie nicht in Workflowattribut- oder -Parameternamen verwenden können.

Die Verwendung eines OGNL-Schlüsselwortes als Präfix für einen Attributnamen unterbricht die OGNL-Verarbeitung nicht. Sie können beispielsweise einen Parameter `trueParameter` nennen. Bei reservierten Schlüsselwörtern wird die Groß- und Kleinschreibung nicht beachtet.

Die folgenden Schlüsselwörter können nicht in Workflowattribut- und -Parameternamen verwendet werden.

**Tabelle 1-2.** Unzulässige Schlüsselwörter in Attribut- und Parameternamen

Unzulässige Schlüsselwörter	Unzulässige Schlüsselwörter	Unzulässige Schlüsselwörter
■ abstract	■ eof	■ _memberAccess
■ back_char_esc	■ esc	■ native
■ back_char_literal	■ exponent	■ package
■ boolean	■ export	■ private
■ byte	■ extends	■ public
■ char	■ false	■ root
■ char_literal	■ final	■ short
■ class	■ flt_literal	■ static
■ _classResolver	■ flt_suff	■ string_esc
■ const	■ ident	■ string_literal
■ context	■ implements	■ synchronized
■ debugger	■ import	■ this
■ dec_digits	■ in	■ _traceEvaluations
■ dec_flt	■ int	■ true
■ default	■ int_literal	■ _typeConverter
■ delete	■ interface	■ volatil
■ digit	■ _keepLastEvaluation	■ with
■ double	■ _lastEvaluation	■ WithinBackCharLiteral
■ dynamic_subscript	■ letter	■ WithinCharLiteral
■ enum	■ long	■ WithinStringLiteral

## Workflowschema

Ein Workflowschema ist eine grafische Darstellung eines Workflows, die ihn als Flussdiagramm von miteinander verknüpften Workflowelementen zeigt. Das Workflowschema definiert den logischen Fluss eines Workflows.

- [Anzeigen des Workflowschemas](#) auf Seite 24  
Sie sehen das Schema eines Workflows auf der Registerkarte **Schema** für diesen Workflow im Orchestrator-Client.
- [Erstellen eines Workflows im Workflowschema](#) auf Seite 24  
Workflowschemas bestehen aus einer Abfolge von Schemaelementen. Workflowschema-Elemente sind die Bausteine des Workflows. Sie können für Entscheidungen, Skriptaufgaben, Aktionen, Ausnahmehandler oder sogar andere Workflows stehen.
- [Schemaelemente](#) auf Seite 28  
Der Workfloweditor präsentiert die Workflowschemaelemente in Menüs auf der Registerkarte **Schema**. Sie können die auf der Registerkarte **Schema** verfügbaren Schemaelemente bearbeiten, um einen Workflow zu erstellen.
- [Eigenschaften von Schemaelementen](#) auf Seite 31  
Schemaelemente haben Eigenschaften, die Sie auf der Registerkarte **Schema** der Workflowpalette definieren und bearbeiten können.
- [Verknüpfungen und Bindungen](#) auf Seite 34  
Verknüpfungen zwischen Elementen bestimmen den logischen Fluss des Workflows. Bindungen füllen Elemente mit Daten aus anderen Elementen, indem Eingabe- und Ausgabeparameter an Workflowattribute gebunden werden.
- [Entscheidungen](#) auf Seite 40  
Workflows können Entscheidungsfunktionen implementieren, die verschiedene Vorgehensweisen entsprechend einer Booleschen true oder false-Anweisung implementieren.

- [Ausnahmebehandlung](#) auf Seite 43  
Ausnahmebehandlung erfasst alle Fehler, die beim Ausführen eines Schemaelements auftreten. Sie definiert, wie sich das Schemaelement beim Auftreten des Fehlers verhält.
- [Verwenden von Fehlerhandlern](#) auf Seite 44  
Sie können einen standardmäßigen Fehlerhandler verwenden, um das Verhalten bei Fehlern bei bestimmten Workflowschema-Elementen festzulegen. Sie können einen globalen Fehlerhandler verwenden, um das Verhalten bei Fehlern festzulegen, die nicht durch den Standard-Fehlerhandler erfasst werden.
- [Foreach-Elemente und zusammengesetzte Typen](#) auf Seite 45  
Sie können ein Foreach-Element in den Workflow einfügen, sodass ein untergeordneter Workflow ausgeführt wird, der Arrays von Parametern oder Attributen durchläuft. Um die Verständlichkeit und Lesbarkeit eines Workflows zu verbessern, können Sie mehrere Workflowparameter verschiedenen Typs, die logisch verbunden sind, in einem zusammengesetzten Typ zusammenfassen.
- [Hinzufügen einer Switch-Aktivität zu einem Workflow](#) auf Seite 48  
Sie können einem Workflowschema eine einfache Switch-Aktivität hinzufügen, die die Switch-Fälle auf Basis von Workflowattributen oder -parametern definiert.

## Anzeigen des Workflowschemas

Sie sehen das Schema eines Workflows auf der Registerkarte **Schema** für diesen Workflow im Orchestrator-Client.

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Navigieren Sie zu einem Workflow in der hierarchischen Liste der Workflows.
- 3 Klicken Sie auf den Workflow.  
Informationen über diesen Workflow erscheinen im rechten Bereich.
- 4 Wählen Sie die Registerkarte **Schema** im rechten Bereich.

Sie sehen die grafische Darstellung des Workflows.

## Erstellen eines Workflows im Workflowschema

Workflowschemas bestehen aus einer Abfolge von Schemaelementen. Workflowschema-Elemente sind die Bausteine des Workflows. Sie können für Entscheidungen, Skriptaufgaben, Aktionen, Ausnahmehandler oder sogar andere Workflows stehen.

Sie können Workflows im Workfloweditor erstellen, indem Sie Schemaelemente von der Workflowpalette links im Workfloweditor in das Workflowschema-Diagramm ziehen.

### Bearbeiten eines Workflowschemas

Sie bauen einen Workflow durch Erstellen einer Sequenz von Schemaelementen, die den logischen Fluss des Workflows definieren.

Standardmäßig sind alle Elemente im Workflowschema verknüpft. Links zwischen den Elementen sind als Pfeile dargestellt. Wenn Sie ein neues Element zum Workflowschema hinzufügen, müssen Sie es auf einen Pfeil oder ein bestehendes Workflowelement ziehen, das nicht mit dem nächsten Element verknüpft ist. Nach dem Hinzufügen von Workflowelementen zum Schema können Sie bestehende Links löschen und neue erstellen, um den logischen Fluss des Workflows zu definieren.



Sie können ein Element oder eine Auswahl von Elementen aus einem Schema eines vorhandenen Workflows in ein Schema des Workflows kopieren, den Sie bearbeiten. Siehe „[Kopieren von Workflowschema-Elementen](#)“, auf Seite 26.

Ein Workflowschema muss mindestens ein Element **Workflow beenden** haben, es können aber auch mehrere sein.

### Voraussetzungen

Öffnen Sie einen Workflow für die Bearbeitung im Workfloweditor.

### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Schema**.
- 2 Ziehen Sie ein Schemaelement aus dem Menü **Generisch** in den linken Bereich in das Workflowschema.
- 3 Doppelklicken Sie auf das Element, das Sie in das Workflowschema gezogen haben, geben Sie einen passenden Namen ein und drücken Sie die Eingabetaste.

Sie müssen Elementen individuelle Namen im Workflowkontext geben.

Sie können die Elemente **Warte-Timer**, **Warteereignis**, **Workflow beenden** oder **Ausnahme auslösen** nicht ändern.

- 4 (Optional) Klicken Sie mit der rechten Maustaste auf ein Element im Schema und wählen Sie **Kopieren**.
- 5 (Optional) Klicken Sie mit der rechten Maustaste auf eine geeignete Stelle im Schema und wählen Sie **Einfügen**.

Kopieren und Einfügen vorhandener Schemaelemente ist eine schnelle Möglichkeit, ähnliche Elemente zum Schema hinzuzufügen. Alle Einstellungen des kopierten Elements werden im eingefügten Element angezeigt, außer dem Business-Zustand. Passen Sie die Einstellungen der eingefügten Elemente entsprechend an.

- 6 Ziehen Sie Schemaelemente aus den Menüs **Standard**, **Protokoll** oder **Netzwerk** in das Workflowschema.

Sie können die Namen der Elemente in den Menüs **Standard**, **Protokoll** oder **Netzwerk** bearbeiten. Ihre Skripterstellung können Sie nicht ändern.

- 7 Ziehen Sie Schemaelemente aus dem Menü **Generisch** in das Workflowschema.

Wenn Sie Aktionen oder Workflows in das Workflowschema ziehen, wird ein Dialogfeld angezeigt, in dem Sie danach suchen können, um sie einzufügen.

- 8 Geben Sie im Textfeld **Filter** den Namen oder Teil des Namens des Workflows oder der Aktion ein.

Die Workflows oder Aktionen, die mit der Suche übereinstimmen, werden im Dialogfeld angezeigt.

- 9 Doppelklicken Sie zum Auswählen auf einen Workflow oder eine Aktion.

Sie haben den Workflow oder die Aktion in das Workflowschema eingefügt.

- 10 Wiederholen Sie dieses Verfahren, bis Sie alle erforderlichen Schemaelemente dem Workflowschema hinzugefügt haben.

### Weiter

Definieren Sie die Eigenschaften der hinzugefügten Elemente und verknüpfen und binden Sie sie zusammen.

## Kopieren von Workflowschema-Elementen

Sie können ein Element oder eine Auswahl von Elementen aus einem Schema eines vorhandenen Workflows in ein Schema des Workflows kopieren, den Sie bearbeiten.

### Voraussetzungen

Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.

### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Schema**.
- 2 Wählen Sie im linken Fensterbereich den Workflow, aus dem Sie die Schemaelemente kopieren möchten.
  - Klicken Sie auf **Alle Workflows** und wählen Sie den Workflow aus der hierarchischen Liste der Workflows.
  - Geben Sie den Namen des Workflows in das Suchtextfeld ein und drücken Sie die Eingabetaste.
- 3 Klicken Sie mit der rechten Maustaste auf den ausgewählten Workflow und wählen Sie **Öffnen**.  
Ein Fenster mit den Workfloweigenschaften wird angezeigt.
- 4 Klicken Sie im Workflowfenster auf die Registerkarte **Schema**.
- 5 Wählen Sie eines oder mehrere Workflowschema-Elemente aus, klicken Sie mit der rechten Maustaste auf die Auswahl und wählen Sie **Kopieren**.
- 6 Klicken Sie mit der rechten Maustaste auf die Registerkarte **Schema** des Workflows, den Sie bearbeiten, und wählen Sie **Einfügen**.

Sie haben Workflowschema-Elemente von einem Workflow in einen anderen kopiert.

### Weiter

Sie müssen die kopierten Schemaelemente mit dem vorhandenen Workflowschema verknüpfen und sie daran binden.

## Höherstufen von Eingabe- und Ausgabeparametern

Sie können die Eingabe- und Ausgabeparameter eines untergeordneten Elements in den übergeordneten Workflow höherstufen.

Sie können ein benutzerdefiniertes Attribut höherstufen, das Sie im Workfloweditor auf der Registerkarte **Allgemein** definiert haben. Sie können vordefinierte Attribute nur höherstufen, indem Sie einen Eingabeparameter durch ein Attribut mit übereinstimmendem Typ ersetzen.

---

**HINWEIS** Wenn Sie ein vordefiniertes Attribut höherstufen und ihm einen benutzerdefinierten Wert zuweisen, wird ein dupliziertes Attribut erstellt, damit der Wert des ursprünglichen Attributs nicht überschrieben wird. Das duplizierte Attribut behält den Namen des ursprünglichen Attributs, wobei der numerische Wert am Ende des Attributnamens um 1 erhöht wird.

---

### Voraussetzungen

Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.

### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Schema**.

- 2 Fügen Sie dem Workflowschema ein Workflow- oder Aktionselement hinzu.

Die folgende Benachrichtigung wird im Schema-Fensterbereich oben angezeigt.

Möchten Sie die Parameter der Aktivität dem aktuellen Workflow als Eingabe/Ausgabe hinzufügen?

- 3 Klicken Sie in der Benachrichtigung auf **Einrichten**.

Ein Popup-Fenster mit den verfügbaren Optionen wird angezeigt.

- 4 Wählen Sie den Zuordnungstyp für jeden einzelnen Eingabeparameter aus.

Option	Beschreibung
<b>Eingabe</b>	Das Argument wird einem Workflow-Eingabeparameter zugeordnet.
<b>Überspringen</b>	Das Argument wird einem NULL-Wert zugeordnet.
<b>Wert</b>	Das Argument wird einem Attribut zugeordnet, dessen Wert Sie über die Spalte „Wert“ festlegen können.

- 5 Wählen Sie den Zuordnungstyp für jeden einzelnen Ausgabeparameter aus.

Option	Beschreibung
<b>Output</b>	Das Argument wird einem Workflow-Ausgabeparameter zugeordnet.
<b>Überspringen</b>	Das Argument wird einem NULL-Wert zugeordnet.
<b>Lokale Variable</b>	Das Argument wird einem Attribut zugeordnet.

- 6 Klicken Sie auf **Heraufstufen**.

Sie haben Parameter in den übergeordneten Workflow höhergestuft.

## Ändern von Suchergebnissen

Sie verwenden das Textfeld **Suchen**, um Elemente wie Workflows oder Aktionen zu finden. Wenn eine Suche ein Teilergebnis zurückbringt, können Sie die Anzahl von Ergebnissen ändern, die von der Suche zurückgegeben werden.

Wenn Sie die Suche für ein Element verwenden, wird in einem grünen Meldungsfeld angezeigt, dass die Suche alle Ergebnisse auflistet. Ein gelbes Meldungsfeld zeigt an, dass die Suche nur Teilergebnisse anzeigt.

### Vorgehensweise






- 1 (Optional) Wenn Sie einen Workflow im Workfloweditor bearbeiten, klicken Sie auf **Speichern und schließen**, um den Editor zu beenden.
- 2 Wählen Sie im Menü des Orchestrator-Clients **Extras > Benutzereinstellungen**.
- 3 Klicken Sie auf die Registerkarte **Allgemein**.
- 4 Geben Sie im Textfeld **Maximale Größe Finder** die Anzahl der Ergebnisse ein, die bei einer Suche zurückgegeben werden sollen.
- 5 Klicken Sie im Dialogfeld „Benutzereinstellungen“ auf **Speichern und schließen**.

Sie haben die Anzahl der Ergebnisse geändert, die bei einer Suche zurückgegeben werden.






## Schemaelemente

Der Workfloweditor präsentiert die Workflowschemaelemente in Menüs auf der Registerkarte **Schema**. Sie können die auf der Registerkarte **Schema** verfügbaren Schemaelemente bearbeiten, um einen Workflow zu erstellen.









**Tabelle 1-3.** Schemaelemente und Symbole

Schemaelement, Name	Beschreibung	Symbol	Position im Workfloweditor
<b>Start des Workflows</b>	Der Startpunkt des Workflows. Alle Workflows enthalten dieses Element. Ein Workflow kann nur ein Startelement haben. Startelemente haben eine Ausgabe und keine Eingabe. Sie können aus dem Workflowschema nicht entfernt werden.		Auf der Registerkarte <b>Schema</b> immer vorhanden
<b>Skriptfähige Aufgabe</b>	Allgemeine Aufgaben, die Sie definieren. In diesem Element schreiben Sie JavaScript-Funktionen.		Die <b>generische</b> Workflowpalette
<b>Entscheidung</b>	Eine boolesche Funktion. Entscheidungselemente übernehmen einen Eingabeparameter und geben <code>true</code> oder <code>false</code> zurück. Die Art der Entscheidung, die vom Element getroffen wird, hängt von der Art des Eingabeparameters ab. Mit Entscheidungselementen wird der Workflow je nach dem vom Entscheidungselement empfangenen Eingabeparameter in verschiedene Richtungen gelenkt. Wenn der empfangene Eingabeparameter einem erwarteten Wert entspricht, wird der Workflow entlang einer bestimmten Route fortgesetzt. Wenn die Eingabe nicht den erwarteten Wert hat, wird der Workflow auf einem alternativen Pfad fortgesetzt.		Die <b>generische</b> Workflowpalette
<b>Benutzerdefinierte Entscheidung</b>	Eine boolesche Funktion. Benutzerdefinierte Entscheidungen können mehrere Eingabeparameter übernehmen und entsprechend einem benutzerdefinierten Skript verarbeiten. Gibt entweder <code>true</code> oder <code>false</code> zurück.		Die <b>generische</b> Workflowpalette
<b>Entscheidungsaktivität</b>	Eine boolesche Funktion. Eine Entscheidungsaktivität führt einen Workflow aus und bindet seine Ausgabeparameter an einen <code>true</code> - oder einen <code>false</code> -Pfad.		Die <b>generische</b> Workflowpalette



**Tabelle 1-3.** Schemaelemente und Symbole (Fortsetzung)

Schemaelement, Name	Beschreibung	Symbol	Position im Workfloweditor
<b>Benutzerinteraktion</b>	Damit können Benutzer neue Eingabeparameter an den Workflow übergeben. Sie können entwerfen, wie das Benutzerinteraktionselement die Eingabeparameter abfragt und Integritätsregeln für die Parameter einbauen, die vom Benutzer eingegeben werden können. Sie können Berechtigungen festlegen, um zu bestimmen, welche Benutzer die Eingabeparameter eingeben dürfen. Wenn ein laufender Workflow zu einem Benutzerinteraktionselement kommt, wechselt er in einen passiven Status und fordert den Benutzer zur Eingabe auf. Sie können einen Zeitüberschreitungswert festlegen, innerhalb dessen die Benutzer eine Eingabe vornehmen müssen. Der Workflow wird nach Maßgabe der Daten wieder aufgenommen, die der Benutzer übergibt. Andernfalls wird eine Ausnahme ausgegeben, wenn der Zeitüberschreitungswert erreicht ist. Während der Workflowtoken auf die Benutzereingabe wartet, befindet er sich im Status <code>waiting</code> .		Die <b>generische</b> Workflowpalette
<b>Warte-Timer</b>	Wird von Workflows mit langer Ausführungszeit benutzt. Wenn ein laufender Workflow ein Warte-Timer-Element erreicht, wechselt er in einen passiven Status. Sie legen ein absolutes Datum fest, an dem der Workflow weiter ausgeführt wird. Während der Workflowtoken auf das Wiederaufnahmedatum wartet, befindet er sich im Status <code>waiting-signal</code> .		Die <b>generische</b> Workflowpalette
<b>Warteereignis</b>	Wird bei Workflows mit langer Ausführungsdauer verwendet. Wenn ein laufender Workflow ein Warteereignis-Element erreicht, wechselt er in einen passiven Status. Sie definieren ein Auslöserereignis, auf das der Workflow wartet, bevor er weiter ausgeführt wird. Während der Workflowtoken auf das Ereignis wartet, befindet er sich im Status <code>waiting-signal</code> .		Die <b>generische</b> Workflowpalette
<b>Ende des Workflows</b>	Der Endpunkt eines Workflows. Sie können in ein Schema mehrere Endelemente einbauen, um die verschiedenen möglichen Ergebnisse des Workflows abzubilden. Endelemente haben eine Eingabe ohne Ausgabe. Wenn ein Workflow ein Ende des Workflowelements erreicht, wechselt der Workflowtoken in den Status <code>completed</code> .		Die <b>generische</b> Workflowpalette
<b>Ausgelöste Ausnahme</b>	Erstellt eine Ausnahme und stoppt den Workflow. Dieses Element kann im Workflowschema mehrfach vorkommen. Ausnahmeelemente haben einen Eingabeparameter, der nur den „string“-Datentyp haben kann, und keinen Ausgabeparameter. Wenn ein Workflow ein Ausnahmeelement erreicht, wechselt der Workflowtoken in den Status <code>failed</code> .		Die <b>generische</b> Workflowpalette

**Tabelle 1-3.** Schemaelemente und Symbole (Fortsetzung)

Schemaelement, Name	Beschreibung	Symbol	Position im Workfloweditor
<b>Workflowanmerkung</b>	Damit können Sie Abschnitte eines Workflows mit Anmerkungen versehen. Sie können das Anmerkungsfeld ausdehnen, um Abschnitte des Workflows abzustecken. Sie können die Hintergrundfarbe der Anmerkungen ändern, um die Workflowzonen voneinander zu unterscheiden. Workflowanmerkungen bieten nur visuelle Informationen, um Sie beim Verständnis des Schemas zu unterstützen.		Die <b>generische</b> Workflowpalette
<b>Aktionselement</b>	Ruft eine Aktion aus der Orchestrator-Aktionsbibliothek auf. Wenn ein Workflow ein Aktionselement erreicht, ruft er diese Aktion auf und führt sie aus.		Die <b>generische</b> Workflowpalette
<b>Workflowelement</b>	Startet synchron einen anderen Workflow. Wenn ein Workflow ein Workflowelement in seinem Schema erreicht, führt er diesen Workflow als Teil seines eigenen Prozesses aus. Der Originalworkflow wird erst fortgesetzt, nachdem der aufgerufene Workflow abgeschlossen ist.		Die <b>generische</b> Workflowpalette
<b>Foreach-Element</b>	Führt einen Workflow für jedes Element aus einem Array aus. Beispiel: Sie können den Workflow „Virtuelle Maschine umbenennen“ auf allen virtuellen Maschinen aus einem Ordner ausführen.		Die <b>generische</b> Workflowpalette
<b>Asynchroner Workflow</b>	Startet einen Workflow asynchron. Wenn ein Workflow ein asynchrones Workflowelement erreicht, startet er diesen Workflow und fährt mit dem Ausfüllen seines eigenen Schemas fort. Der ursprüngliche Workflow wartet nicht, bis der aufgerufene Workflow abgeschlossen ist.		Die <b>generische</b> Workflowpalette
<b>Workflow planen</b>	Erstellt eine Aufgabe, mit der das Ausführen des Workflows zu einem bestimmten Zeitpunkt geplant wird. Danach setzt der Workflow seinen normalen Ablauf fort.		Die <b>generische</b> Workflowpalette
<b>Verschachtelte Workflows</b>	Mehrere Workflows werden gleichzeitig gestartet. Sie können entscheiden, lokale und externe Workflows zu verschachteln, die sich auf verschiedenen Orchestrator-Servern befinden. Sie können auch Workflows mit verschiedenen Anmeldedaten ausführen. Der Workflow wartet, bis alle verschachtelten Workflows abgeschlossen sind, bevor er weiter ausgeführt wird.		Die <b>generische</b> Workflowpalette
<b>Handlerfehler</b>	Behandelt einen Fehler für ein spezifisches Workflowelement. Der Workflow kann den Fehler behandeln, indem er eine Ausnahme erstellt, einen anderen Workflow aufruft oder ein benutzerdefiniertes Skript ausführt.		Die <b>generische</b> Workflowpalette

**Tabelle 1-3.** Schemaelemente und Symbole (Fortsetzung)

Schemaelement, Name	Beschreibung	Symbol	Position im Workfloweditor
<b>Standard-Fehlerhandle</b>	Behandelt Workflowfehler, die von Standard-Fehlerhandles nicht erfasst werden. Sie können alle verfügbaren Schemaelemente verwenden, um Fehler zu behandeln.		Die <b>generische</b> Workflowpalette
<b>Switch</b>	Wechselt zu alternativen Workflowpfaden basierend auf einem Workflowattribut oder einem Parameter.		Die <b>generische</b> Workflowpalette
<b>Vordefinierte Aufgabe</b>	Nicht bearbeitbare, skriptfähige Elemente, die von Workflows häufig benutzte Standardaufgaben durchführen. Folgende Aufgaben sind vordefiniert: <b>Einfach</b> <ul style="list-style-type: none"> <li>■ Schlaf</li> <li>■ Anmeldedaten ändern</li> <li>■ Warten bis Datum</li> <li>■ Warten bis benutzerdefiniertes Ereignis</li> <li>■ Indikator erhöhen</li> <li>■ Indikator herabsetzen</li> </ul> <b>Protokoll</b> <ul style="list-style-type: none"> <li>■ Systemprotokoll</li> <li>■ Systemwarnung</li> <li>■ Systemfehler</li> <li>■ Serverprotokoll</li> <li>■ Serverwarnung</li> <li>■ Serverfehler</li> <li>■ System+Serverprotokoll</li> <li>■ System+Serverwarnung</li> <li>■ System+Serverfehler</li> </ul> <b>Netzwerk</b> <ul style="list-style-type: none"> <li>■ HTTP post</li> <li>■ HTTP get</li> </ul>		Die Workflowpaletten <b>Basis</b> , <b>Protokoll</b> und <b>Netzwerk</b>

## Eigenschaften von Schemaelementen

Schemaelemente haben Eigenschaften, die Sie auf der Registerkarte **Schema** der Workflowpalette definieren und bearbeiten können.


### Bearbeiten Sie die globalen Eigenschaften eines Schemaelements

Sie definieren Sie die globalen Eigenschaften eines Schemaelements auf seiner Registerkarte „Info“.

#### Voraussetzungen

Vergewissern Sie sich, dass die Registerkarte **Schema** des Workfloweditors Elemente enthält.

#### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Schema**.
  - 2 Wählen Sie ein zu bearbeitendes Element durch Klicken auf das Symbol **Bearbeiten** ().
- Ein Dialogfeld wird geöffnet, das die Eigenschaften des Elements auflistet.

3 Klicken Sie auf die Registerkarte **Info**.

4 Geben Sie einen Namen für das Schemaelement in das Textfeld **Name**.

Dieser Name wird im Schemaelement im Workflowschema-Diagramm angezeigt.

5 Wählen Sie aus dem Dropdown-Menü **Interaktion** eine Beschreibung aus.

Die Eigenschaft **Interaktion** ermöglicht das Auswählen zwischen Standardbeschreibungen, wie dieses Element mit Objekten außerhalb des Workflows interagiert. Diese Eigenschaft ist nur als Information gedacht.

6 (Optional) Geben Sie eine Beschreibung des Business-Status im Textfeld **Business-Status** an.

Die Eigenschaft **Business-Status** ist eine kurze Beschreibung davon, was dieses Element tut. Wenn ein Workflow ausgeführt wird, zeigt das Workflowtoken den Business-Status jedes Element bei seiner Ausführung an. Diese Funktion ist hilfreich beim Verfolgen des Workflowstatus.

7 (Optional) Geben Sie im Textfeld **Beschreibung** eine Beschreibung für das Schemaelement ein.

## Registerkarten für Eigenschaften von Schemaelementen

Sie greifen auf die Eigenschaften eines Schemaelements zu, indem Sie auf ein Element klicken, das Sie in das Workflowschema gezogen haben. Die Eigenschaften des Elements erscheinen in Registerkarten unten im Workfloweditor.

Verschiedene Schemaelemente haben verschiedene Eigenschaftenregisterkarten.

**Tabelle 1-4.** Registerkarten für Eigenschaften nach Schemaelement

Registerkarte für Eigenschaften eines Schemaelements	Beschreibung	Gilt für den Schemaelementtyp
<b>Attribute</b>	Attribute, die Elemente von einer externen Quelle benötigen, beispielsweise einem Benutzer, einem Ereignis oder einem Timer. Die Attribute können ein Grenzwert für die Zeitüberschreitung, ein Zeitpunkt und ein Datum, ein Auslöser oder Anmeldezeiten von Benutzern sein.	<ul style="list-style-type: none"> <li>■ Benutzerinteraktion</li> <li>■ Warteereignis</li> <li>■ Warte-Timer</li> </ul>
<b>Entscheidung</b>	Definiert die Entscheidungsanweisung. Der Eingabeparameter, den das Entscheidungselement empfängt, stimmt mit der Entscheidungsanweisung überein oder auch nicht. Daraus ergeben sich zwei mögliche Aktionsverläufe.	Entscheidung
<b>Ende des Workflows</b>	Stoppt den Workflow, entweder weil der Workflow erfolgreich abgeschlossen wurde oder weil ein Fehler aufgetreten ist und eine Ausnahme zurückgegeben wurde.	<ul style="list-style-type: none"> <li>■ Ende</li> <li>■ Ausnahme</li> </ul>



**Tabelle 1-4.** Registerkarten für Eigenschaften nach Schemaelement (Fortsetzung)

Registerkarte für Eigenschaften eines Schemaelements	Beschreibung	Gilt für den Schemaelementtyp
<b>Ausnahme</b>	Verhalten dieses Schemaelements bei einer Ausnahme.	<ul style="list-style-type: none"> <li>■ Aktion</li> <li>■ Asynchroner Workflow</li> <li>■ Ausnahme</li> <li>■ Verschachtelte Workflows</li> <li>■ Vordefinierte Aufgabe</li> <li>■ Workflow planen</li> <li>■ Skriptfähige Aufgabe</li> <li>■ Benutzerinteraktion</li> <li>■ Warteereignis</li> <li>■ Warte-Timer</li> <li>■ Workflow</li> </ul>
<b>Externe Eingaben</b>	Eingabeparameter für Eingaben des Benutzers an einer bestimmten Stelle während des Ausführens des Workflows.	Benutzerinteraktion
<b>EIN</b>	Die EIN-Bindung für dieses Element. Die EIN-Bindung definiert die Art, in der ein Schemaelement Eingaben von dem Element erhält, das im Workflow vor ihm kommt.	<ul style="list-style-type: none"> <li>■ Aktion</li> <li>■ Asynchroner Workflow</li> <li>■ Benutzerdefinierte Entscheidung</li> <li>■ Vordefinierte Aufgabe</li> <li>■ Workflow planen</li> <li>■ Skriptfähige Aufgabe</li> <li>■ Workflow</li> </ul>
<b>Info</b>	Die allgemeinen Eigenschaften und Beschreibungen des Schemaelements. Die Informationen auf der Registerkarte <b>Informationen</b> hängen vom Typ des Schemaelements ab.	<ul style="list-style-type: none"> <li>■ Aktion</li> <li>■ Asynchroner Workflow</li> <li>■ Benutzerdefinierte Entscheidung</li> <li>■ Entscheidung</li> <li>■ Verschachtelte Workflows</li> <li>■ Hinweis</li> <li>■ Vordefinierte Aufgabe</li> <li>■ Workflow planen</li> <li>■ Skriptfähige Aufgabe</li> <li>■ Benutzerinteraktion</li> <li>■ Warteereignis</li> <li>■ Warte-Timer</li> <li>■ Workflow</li> </ul>
<b>AUS</b>	Die AUS-Bindung für dieses Element. Die AUS-Bindung definiert die Art, in der das Schemaelement Ausgabeparameter an die Workflowattribute oder an die Ausgabeparameter des Workflows bindet.	<ul style="list-style-type: none"> <li>■ Aktion</li> <li>■ Asynchroner Workflow</li> <li>■ Vordefinierte Aufgabe</li> <li>■ Workflow planen</li> <li>■ Skriptfähige Aufgabe</li> <li>■ Workflow</li> </ul>
<b>Präsentation</b>	Definiert das Layout des Dialogfelds für die Eingabeparameter, das der Benutzer sieht, wenn der Workflow während des Ausführens eine Benutzereingabe benötigt.	Benutzerinteraktion

**Tabelle 1-4.** Registerkarten für Eigenschaften nach Schemaelement (Fortsetzung)

Registerkarte für Eigenschaften eines Schemaelements	Beschreibung	Gilt für den Schemaelementtyp
<b>Skripterstellung</b>	Zeigt die JavaScript-Funktion, die das Verhalten dieses Schemaelements definiert. Bei asynchronen Workflows, einem geplanten Workflow und Aktions-elementen ist dieses Skript schreibgeschützt. Bei skriptfähigen Aufgaben und benutzerdefinierten Entscheidungselementen bearbeiten Sie den JavaScript-Code auf dieser Registerkarte.	<ul style="list-style-type: none"> <li>■ Aktion</li> <li>■ Asynchroner Workflow</li> <li>■ Benutzerdefinierte Entscheidung</li> <li>■ Vordefinierte Aufgabe</li> <li>■ Workflow planen</li> <li>■ Skriptfähige Aufgabe</li> </ul>
<b>Visuelle Bindung</b>	Zeigt eine grafische Darstellung der Art der Bindung der Parameter und Attribute dieses Schemaelements an Parameter und Attribute der Elemente, die vor und nach ihm im Workflow kommen. Dies ist eine weitere Darstellung der EIN- und AUS-Bindungen des Elements.	<ul style="list-style-type: none"> <li>■ Aktion</li> <li>■ Asynchroner Workflow</li> <li>■ Vordefinierte Aufgabe</li> <li>■ Workflow planen</li> <li>■ Skriptfähige Aufgabe</li> <li>■ Workflow</li> </ul>
<b>Workflows</b>	Wählt die Workflows aus, die zu verschachteln sind.	Verschachtelte Workflows

## Verknüpfungen und Bindungen

Verknüpfungen zwischen Elementen bestimmen den logischen Fluss des Workflows. Bindungen füllen Elemente mit Daten aus anderen Elementen, indem Eingabe- und Ausgabeparameter an Workflowattribute gebunden werden.

Zum Verständnis von Verknüpfungen und Bindungen müssen Sie den Unterschied zwischen dem logischen Fluss eines Workflows und dem Datenfluss eines Workflows verstehen.

### Logischer Fluss eines Workflows

Der logische Fluss eines Workflows ist der Fortschritt des Workflows von einem Element zum nächsten im Schema, während der Workflow ausgeführt wird. Sie definieren den logischen Fluss des Workflows, indem Sie Elemente im Schema verknüpfen.

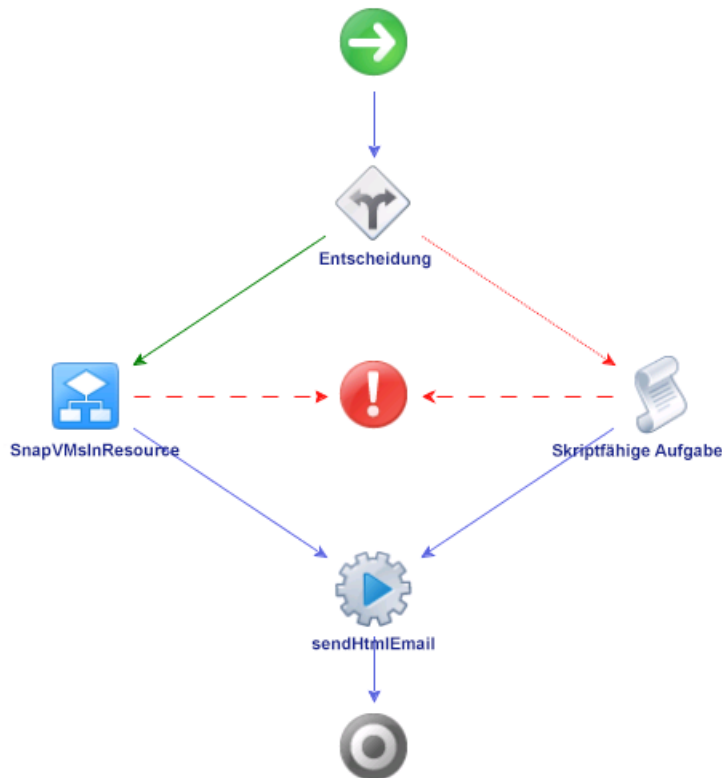
Der Standardpfad ist der Pfad, den der Workflow im logischen Fluss nimmt, wenn alle Elemente wie erwartet ablaufen. Der Ausnahmepfad ist der Pfad, den der Workflow im logischen Fluss nimmt, wenn Elemente nicht wie erwartet ablaufen.

Verschiedene Pfeilformen im Workflowschema bezeichnen die verschiedenen Pfade, die der Workflow durch seinen logischen Fluss nehmen kann.

- Ein blauer Pfeil bezeichnet den Standardpfad, den der Workflow von einem Element zum nächsten nimmt.
- Ein grüner Pfeil bezeichnet den Pfad, den der Workflow nimmt, wenn ein boolesches Entscheidungselement `true` zurückgibt.
- Ein roter gepunkteter Pfeil bezeichnet den Pfad, den der Workflow nimmt, wenn ein boolesches Entscheidungselement `false` zurückgibt.
- Ein roter gestrichelter Pfeil bezeichnet den Ausnahmepfad, den der Workflow nimmt, wenn ein Workf-lowelement nicht richtig abläuft.

Die folgende Abbildung zeigt ein Beispiel für ein Workflowschema, in dem die verschiedenen Pfade dargestellt werden, die ein Workflow nehmen kann.

**Abbildung 1-1.** Verschiedene Workflowpfade durch den logischen Fluss des Workflows



Dieser Beispielworkflow kann folgende Pfade durch seinen logischen Fluss nehmen.

- Standardpfad, true-Entscheidungsergebnis, keine Ausnahmen.
  - a Das Entscheidungselement gibt true zurück.
  - b Der Workflow SnapVMsInResourcePool wird erfolgreich ausgeführt.
  - c Die Aktion sendHtmlEmail wird erfolgreich ausgeführt.
  - d Der Workflow endet erfolgreich im Status completed.
- Standardpfad, Entscheidungsergebnis False, keine Ausnahmen.
  - a Das Entscheidungselement gibt false zurück.
  - b Der Vorgang, der das skriptfähige Aufgabenelement definiert, wird erfolgreich ausgeführt.
  - c Die Aktion sendHtmlEmail wird erfolgreich ausgeführt.
  - d Der Workflow endet erfolgreich im Status completed.
- Entscheidungsergebnis True, Ausnahme.
  - a Das Entscheidungselement gibt true zurück.
  - b Der Workflow SnapVMsInResourcePool ist auf einen Fehler gestoßen.
  - c Der Workflow gibt eine Ausnahme zurück und stoppt im Status failed.
- Entscheidungsergebnis False, Ausnahme.
  - a Das Entscheidungselement gibt false zurück.
  - b Der Vorgang, der das skriptfähige Aufgabenelement definiert, trifft auf einen Fehler.

- c Der Workflow gibt eine Ausnahme zurück und stoppt im Status failed.

## Elementlinks

Links verbinden Schemaelemente und definieren den logischen Fluss des Workflows von einem Element zum nächsten.

Elemente können in der Regel nur einen ausgehenden Link zu einem anderen Element im Workflow festlegen und einen Ausnahmelink zu einem Element, das sein Ausnahmeverhalten definiert. Der ausgehende Link definiert den Standardpfad des Workflows. Der Ausnahmelink definiert den Ausnahmepfad des Workflows. In den meisten Fällen kann ein Schemaelement eingehende Standardpfadlinks von mehreren Elementen empfangen.

Die folgenden Elemente sind Ausnahmen von den vorangegangenen Aussagen.

- Das Element „Workflow starten“ kann keine eingehenden Links empfangen und hat keinen Ausnahmelink.
- Ausnahmeelemente können mehrere eingehende Ausnahmelinks empfangen und haben keine ausgehenden oder Ausnahmelinks.
- Entscheidungselemente haben zwei ausgehende Links, die die Pfade des Workflows je nach dem Ergebnis true oder false der Entscheidung definieren. Entscheidungen haben keinen Ausnahmelink.
- Die Elemente „Workflow beenden“ können keine ausgehenden Links oder Ausnahmelinks haben.

## Erstellen von Standardpfadlinks

Standardpfadlinks legen die normale Ausführung des Workflows fest.

Wenn Sie ein Element mit einem anderen verknüpfen, erfolgte die Verknüpfung der Elemente immer in der Reihenfolge, in der sie im Workflow ausgeführt werden. Sie starten immer mit dem Element, das zuerst ausgeführt wird, um eine Verknüpfung zwischen zwei Elementen herzustellen.

### Voraussetzungen

- Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.
- Vergewissern Sie sich, dass die Registerkarte **Schema** des Workfloweditors Elemente enthält.

### Vorgehensweise

- 1 Führen Sie den Mauszeiger auf das Element, für das Sie eine Verbindung mit einem anderen Element herstellen möchten.  
Ein blauer und ein roter Pfeil erscheinen auf der rechten Seite des Elements.
- 2 Führen Sie den Mauszeiger auf den blauen Pfeil.  
Der blaue Pfeil wird vergrößert.
- 3 Klicken Sie mit der linken Maustaste auf dem blauen Pfeil, halten Sie die linke Maustaste gedrückt und bewegen Sie den Mauszeiger auf das Zielelement.  
Ein blauer Pfeil erscheint zwischen den beiden Elementen, und ein grünes Rechteck wird um das Zielelement aufgezogen.
- 4 Lassen Sie die linke Maustaste los.  
Der blaue Pfeil bleibt zwischen den beiden Elementen bestehen.

Ein Standardpfad verknüpft nun die Elemente.

## Weiter

Die Elemente sind verbunden, aber Sie haben den Datenfluss noch nicht definiert. Sie müssen die EIN- und AUS-Bindungen definieren, um eingehende und ausgehende Daten an Workflowattribute zu binden.

## Datenfluss eines Workflows

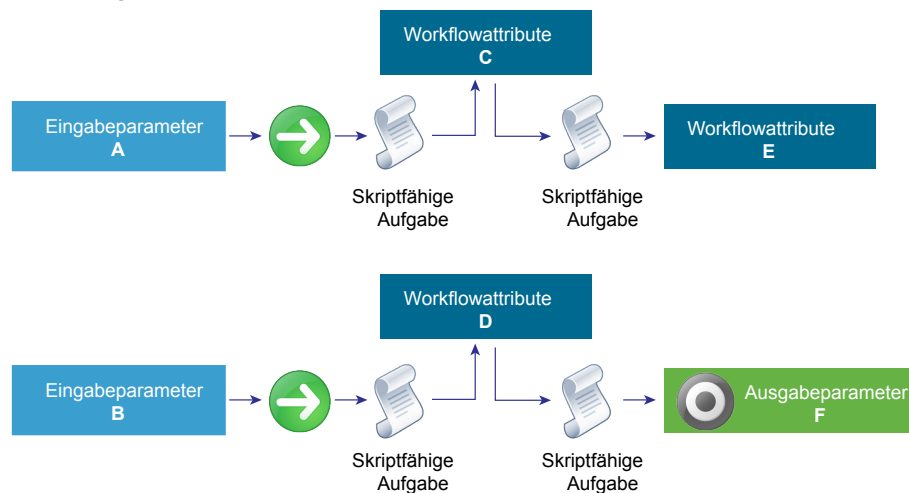
Der Datenfluss eines Workflows ist die Art, in der Workflowelementeingabe- und -ausgabeparameter bei jeder Elementausführung des Workflows an Workflowattribute gebunden werden. Sie definieren den Datenfluss eines Workflows mit Schemaelementbindungen.

Wenn ein Element im Workflowschema ausgeführt wird, benötigt es Daten im Form von Eingabeparametern. Es nimmt die Daten für seine Eingabeparameter, indem es sich an ein Workflowattribut bindet, das Sie beim Erstellen des Workflows festlegen, oder an ein Attribut, das ein vorheriges Element im Workflow bei der Ausführung festgelegt hat.

Das Element verarbeitet die Daten, wandelt sie möglicherweise um und generiert die Ergebnisse der Ausführung in Form von Ausgabeparametern. Das Element bindet seine resultierenden Ausgabeparameter an neue Workflowattribute, die es erstellt. Andere Elemente im Schema können diese neuen Workflowattribute als ihre Eingabeparameter an sich binden. Der Workflow kann die Attribute als seine Ausgabeparameter am Ende der Ausführung generieren.

Die folgende Abbildung zeigt einen sehr einfachen Workflow. Der blaue Pfeil stellt die Elementverknüpfung und den logischen Fluss des Workflows dar. Die rote Linien zeigen den Datenfluss des Workflows.

**Abbildung 1-2.** Beispiel eines Workflow-Datenflusses



Die Daten fließen durch den Workflow wie folgt.

- 1 Der Workflow startet mit Eingabeparametern a und b.
- 2 Das erste Element verarbeitet Parameter a und bindet das resultierende Ergebnis an Workflowattribut c.
- 3 Das erste Element verarbeitet Parameter b und bindet das resultierende Ergebnis an Workflowattribut d.
- 4 Das zweite Element nimmt Workflowattribut c als Eingabeparameter, verarbeitet es und bindet das resultierende Ausgabeparameter an Workflowattribut e.
- 5 Das zweite Element nimmt Workflowattribut d als Eingabeparameter, verarbeitet es und generiert Ausgabeparameter f.
- 6 Der Workflow endet und generiert Workflowattribut f als seinen Ausgabeparameter, das Ergebnis seiner Ausführung.

## Elementbindungen

Sie müssen alle Workflowelement-Eingabe- und Ausgabeparameter an Workflowattribute binden. Bindungen legen Daten in den Elementen fest und definieren die Ausgabe und das Ausnahmeverhalten der Elemente. Verknüpfungen definieren den logischen Fluss des Workflows, während Bindungen den Datenfluss definieren.

Um Daten in einem Element festzulegen, Ausgabeparameter vom Element nach der Verarbeitung zu generieren und eventuelle Fehler bei Ausführung des Elements zu verarbeiten, müssen Sie die Elementbindung einrichten.

### IN-Bindungen

Legen Sie die eingehenden Daten eines Schemaelements fest. Sie binden die lokalen Eingabeparameter des Elements an die Quell-Workflowattribute. Die Registerkarte **IN** zeigt die Eingabeparameter des Elements in der Spalte „Lokale Parameter“ an. Die Registerkarte **IN** zeigt die Workflowattribute an, an die der lokale Parameter in der Spalte „Quellparameter“ gebunden wird. Die Registerkarte zeigt auch den Parametertyp und eine Beschreibung des Parameters an.

### OUT-Bindungen

Ändern Sie Workflowattribute und generieren Sie Ausgabeparameter, nachdem ein Element ausgeführt wurde. Die Registerkarte **OUT** zeigt die Ausgabeparameter des Elements in der Spalte „Lokale Parameter“ an. Die Registerkarte **OUT** zeigt die Workflowattribute an, an die der lokale Parameter in der Spalte „Quellparameter“ gebunden wird. Die Registerkarte zeigt auch den Parametertyp und eine Beschreibung des Parameters an.

### Ausnahmebindungen

Link zu Ausnahmehandler, wenn bei einem Element während der Ausführung eine Ausnahme auftritt.

IN-Bindungen lesen Werte aus dem gebundenen Quellparameter. OUT-Bindungen schreiben Werte in den gebundenen Quellparameter.

Sie müssen IN-Bindungen verwenden, um jedes Attribut oder jeden Eingabeparameter, das bzw. den Sie verwenden, in einem Schemaelement an ein Workflowattribut zu binden. Wenn das Element die Werte der Eingabeparameter ändert, die es bei Ausführung erhält, müssen Sie sie an ein Workflowattribut mit einer OUT-Bindung binden. Durch Binden der Ausgabeparameter des Elements an Workflowelemente können andere Elemente, die ihm im Workflowschema folgen, diese Ausgabeparameter als ihre Eingabeparameter verwenden.

Ein häufiger Fehler beim Erstellen von Workflows ist, Ausgabeparameterwerte nicht zu binden, um so die Änderungen des Elements an den Workflowattributen widerzuspiegeln.

---

**WICHTIG** Wenn Sie ein Element hinzufügen, für das Eingabe- und Ausgabeparameter eines Typs erforderlich sind, den Sie bereits im Workflow definiert haben, legt Orchestrator die Bindungen für diese Parameter fest. Sie müssen überprüfen, ob Orchestrator die korrekten Parameter bindet, falls der Workflow verschiedene Parameter des gleichen Typs definiert, an den das Element gebunden werden kann.

---


## Definieren von Elementbindungen

Nach dem Verknüpfen von Elementen zum Erstellen des logischen Flusses des Workflows definieren Sie Elementbindungen, um die Art der Verarbeitung von empfangenen und generierten Daten durch jedes Element zu definieren.

### Voraussetzungen

Verifizieren Sie, dass Sie ein Workflowschema in der Registerkarte **Schema** des Workfloweditors haben und dass Sie Links zwischen den Elementen erstellt haben.

## Vorgehensweise

- 1 Klicken Sie auf das Symbol **Bearbeiten** () des Elements, für das Sie die Bindungen festlegen möchten.  
Ein Dialogfeld wird geöffnet, das die Eigenschaften des Elements auflistet.
- 2 Klicken Sie auf die Registerkarte **IN**.  
Die Inhalte der Registerkarte **IN** hängen vom ausgewählten Elementtyp ab.
  - Wenn Sie eine vordefinierte Aufgabe, Workflow oder Aktionselement ausgewählt haben, listet die Registerkarte **IN** die möglichen lokalen Eingabeparameter für diesen Elementtyp auf, die Bindung ist jedoch nicht festgelegt.
  - Wenn Sie einen anderen Elementtyp auswählen, können Sie aus einer Liste von bereits für den Workflow definierten Eingabeparametern und Attributen auswählen, indem Sie mit der rechten Maustaste auf die Registerkarte **IN** klicken und **An Workflowparameter/-attribut binden** auswählen.
  - Wenn das erforderliche Attribut noch nicht vorhanden ist, können Sie es erstellen, indem Sie mit der rechten Maustaste auf die Registerkarte **IN** klicken und **An Workflowparameter/-attribut binden > Parameter/Attribut in Workflow erstellen** auswählen.
- 3 Wenn ein passender Parameter schon vorhanden ist, wählen Sie einen zu bindenden Eingabeparameter aus und klicken Sie auf **Nicht festgelegt** im Textfeld **Quellparameter**.  
Eine Liste möglicher Quellparameter und Attribute für die Bindung wird angezeigt.
- 4 Wählen Sie einen Quellparameter aus, um ihn an den lokalen Eingabeparameter aus der vorgeschlagenen Liste zu binden.
- 5 (Optional) Wenn der Quellparameter, an den gebunden werden soll, noch nicht definiert ist, können Sie ihn erstellen, indem Sie auf den Link **Parameter/Attribut in Workflow erstellen** im Dialogfeld zur Parameterauswahl klicken.
- 6 Klicken Sie auf die Registerkarte **OUT**.  
Die Inhalte der Registerkarte **OUT** hängen vom ausgewählten Elementtyp ab.
  - Wenn Sie eine vordefinierte Aufgabe, Workflow oder Aktionselement ausgewählt haben, listet die Registerkarte **OUT** die möglichen lokalen Ausgabeparameter für diesen Elementtyp auf, die Bindung ist jedoch nicht festgelegt.
  - Wenn Sie einen anderen Elementtyp auswählen, können Sie aus einer Liste von für den Workflow definierten Ausgabeparametern und Attributen auswählen, indem Sie mit der rechten Maustaste auf die Registerkarte **OUT** klicken und **An Workflowparameter/-attribut binden** auswählen.
  - Wenn das erforderliche Attribut nicht vorhanden ist, können Sie es erstellen, indem Sie mit der rechten Maustaste auf die Registerkarte **IN** klicken und **An Workflowparameter/-attribut binden > Parameter/Attribut in Workflow erstellen** auswählen.
- 7 Wählen Sie einen Parameter für die Bindung.
- 8 Klicken Sie auf **Quellparameter > Nicht festgelegt**.
- 9 Wählen Sie einen Quellparameter aus, um ihn an den Eingabeparameter zu binden.
- 10 (Optional) Wenn der Parameter, an den gebunden werden soll, noch nicht definiert ist, können Sie ihn erstellen, indem Sie auf die Schaltfläche **Parameter/Attribut in Workflow erstellen** im Dialogfeld zur Parameterauswahl klicken.

Sie haben die Eingabeparameter definiert, die das Element empfängt, und die Ausgabeparameter, die es generiert, und sie an Workflowattribute und -parameter gebunden.

**Weiter**

Sie können Verzweigungen im Workflowpfad erstellen, indem Sie Entscheidungen definieren.

**Entscheidungen**

Workflows können Entscheidungsfunktionen implementieren, die verschiedene Vorgehensweisen entsprechend einer Booleschen `true` oder `false`-Anweisung implementieren.

Entscheidungen sind Verzweigungen im Workflow. Workflowentscheidungen erfolgen entsprechend der Eingaben von Ihnen, Ihren Workflows, Anwendungen oder der Umgebung, in der der Workflow ausgeführt wird. Der Wert des Eingabeparameters, den das Entscheidungselement erhält, legt fest, welche Verzweigung der Workflow nimmt. Eine Workflowentscheidung kann beispielsweise den Leistungsstatus einer bestimmten virtuellen Maschine als Eingabe erhalten. Wenn die virtuelle Maschine eingeschaltet ist, nimmt der Workflow einen bestimmten Pfad durch seinen logischen Fluss. Wenn die virtuelle Maschine ausgeschaltet ist, lautet der Workflowpfad anders.

Entscheidungen sind immer Boolesche Funktionen. Das einzig mögliche Ergebnis jeder Entscheidung sind `true` oder `false`.

**Benutzerdefinierte Entscheidungen**

Benutzerdefinierte Entscheidungen unterscheiden sich von Standardentscheidungen dadurch, dass Sie die Entscheidungsanweisung in einem Skript definieren. Benutzerdefinierte Entscheidungen geben entsprechend der definierten Anweisung `true` oder `false` zurück, wie das folgende Beispiel zeigt.

```
if (decision_statement){
    return true;
}else{
    return false;
}
```

**Erstellen von Links für Entscheidungselemente**

Entscheidungselemente unterscheiden sich von anderen Elementen in einem Workflow. Sie haben als Ausgabeparameter nur `true` oder `false`. Entscheidungselemente haben keine Ausnahmelinks.

**Voraussetzungen**

Vergewissern Sie sich, dass die Registerkarte **Schema** des Workfloweditors Elemente enthält, von denen mindestens ein Entscheidungselement nicht mit anderen Elementen verknüpft ist.

**Vorgehensweise**

- 1 Führen Sie den Mauszeiger auf ein Entscheidungselement, um es mit zwei anderen Elementen zu verknüpfen, die zwei mögliche Verzweigungen im Workflow definieren.

Ein blauer Pfeil und ein roter Pfeil erscheinen rechts neben dem Element.

- 2 Führen Sie den Mauszeiger auf den blauen Pfeil und verschieben Sie den Mauszeiger mit gedrückter linker Maustaste auf das Zielement.

Ein grüner Pfeil erscheint zwischen den beiden Elementen, und das Zielement wird grün. Der grüne Pfeil stellt den `true`-Pfad dar, den der Workflow nimmt, wenn der Eingabeparameter oder das Attribut, der bzw. das vom Entscheidungselement kommt, mit dem Entscheidungselement übereinstimmt.

- 3 Lassen Sie die linke Maustaste los.

Der grüne Pfeil bleibt zwischen den beiden Elementen bestehen. Sie haben den Pfad definiert, den der Workflow nimmt, wenn das Entscheidungselement den erwarteten Wert empfängt.



- 4 Führen Sie den Mauszeiger auf das Entscheidungselement, halten Sie die linke Maustaste gedrückt und bewegen Sie den Mauszeiger auf das Zielelement.

Ein gepunkteter roter Pfeil erscheint zwischen den beiden Elementen, und das Zielelement wird grün. Der rote Pfeil stellt den `false`-Pfad dar, den der Workflow nimmt, wenn der Eingabeparameter oder das Attribut, der bzw. das vom Entscheidungselement kommt, mit dem Entscheidungselement nicht übereinstimmt.

- 5 Lassen Sie die linke Maustaste los.

Der grüne Pfeil bleibt zwischen den beiden Elementen bestehen. Sie haben den Pfad definiert, den der Workflow nimmt, wenn das Entscheidungselement einen nicht erwarteten Eingabewert empfängt.

Sie haben die möglichen `true`- oder `false`-Pfade definiert, die der Workflow in Abhängigkeit von dem Eingabeparameter oder Attribut nimmt, der bzw. das vom Entscheidungselement empfangen wird.

### Weiter

Definieren Sie die Entscheidungsanweisung. Weitere Informationen finden Sie unter „[Erstellen von Workflow-Verzweigungen mithilfe von Entscheidungen](#)“, auf Seite 42.

## Löschen eines verknüpften Entscheidungselements

Wenn Sie ein verknüpftes Entscheidungselement aus einem Workflowschema löschen, müssen Sie angeben, welche Workflowpfade gelöscht werden sollen.

### Voraussetzungen

Vergewissern Sie sich, dass die Registerkarte **Schema** des Workfloweditors Elemente enthält, von denen mindestens eines ein Entscheidungselement mit Pfaden für `wahr` und `falsch` ist.

### Vorgehensweise

- 1 Wählen Sie das Entscheidungselement aus und klicken Sie auf „Löschen“.

Ein Dialogfeld mit verfügbaren Optionen wird angezeigt.

- 2 Wählen Sie, welcher Entscheidungsweig gelöscht werden soll.

Option	Beschreibung
<b>Erfolgszweig</b>	Das Entscheidungselement und alle Elemente, die dem Entscheidungspfad <code>wahr</code> folgen, werden aus dem Workflowschema gelöscht.
<b>Fehlerzweig</b>	Das Entscheidungselement und alle Elemente, die dem Entscheidungspfad <code>falsch</code> folgen, werden aus dem Workflowschema gelöscht.
<b>Beide Zweige</b>	Das Entscheidungselement und alle Elemente, die beiden Entscheidungspfaden folgen, werden aus dem Workflowschema gelöscht.
<b>Keine</b>	Nur das Entscheidungselement und seine Links werden aus dem Workflowschema gelöscht. Alle Elemente, die beiden Entscheidungspfaden folgen, bleiben im Workflowschema erhalten.

- 3 Klicken Sie auf **OK**.


## Erstellen von Workflow-Verzweigungen mithilfe von Entscheidungen

Entscheidungselemente sind einfache boolesche Funktionen, die Sie verwenden, um Verzweigungen in Workflows zu erstellen. Entscheidungselemente legen fest, ob die Eingabe mit der von Ihnen festgelegten Entscheidungsanweisung übereinstimmt. Je nach dem Ausgang der Entscheidung fährt der Workflow mit einem von zwei möglichen Pfaden fort.

### Voraussetzungen

Überprüfen Sie, ob ein Entscheidungselement mit zwei anderen Elementen im Schema im Workfloweditor verknüpft ist, bevor Sie die Entscheidung definieren.

### Vorgehensweise

- 1 Klicken Sie auf das Symbol **Bearbeiten** () des Entscheidungselements.  
Ein Dialogfeld wird geöffnet, das die Eigenschaften des Entscheidungselements aufführt.
- 2 Klicken Sie auf die Registerkarte **Entscheidung** im Dialogfeld.
- 3 Klicken Sie auf den Link **Nicht festgelegt (NULL)**, um den Quelleingangsparameter für diese Entscheidung auszuwählen.  
Ein Dialogfeld mit einer Liste aller Attribute und Eingabeparameter, die in diesem Workflow definiert sind, wird angezeigt.
- 4 Wählen Sie einen Eingabeparameter aus der Liste, in dem Sie darauf doppelklicken.
- 5 Wenn der Quellparameter, an den gebunden werden soll, noch nicht definiert ist, können Sie ihn erstellen, indem Sie auf den Link **Parameter/Attribut in Workflow erstellen** im Dialogfeld zur Parameterauswahl klicken.
- 6 Wählen Sie eine Entscheidungsanweisung aus dem Dropdown-Menü.  
Die vom Menü angebotenen Anweisungen richten sich nach dem Kontext und sind je nach dem ausgewählten Eingabeparameter verschieden.
- 7 Fügen Sie einen Wert hinzu, mit dem die Entscheidungsanweisung übereinstimmen soll.  
Je nach dem Eingabetyp und der von Ihnen ausgewählten Anweisung sehen Sie möglicherweise einen Link **Nicht festgelegt (NULL)** im Textfeld für den Wert. Wenn Sie auf diesen Link klicken, sehen Sie eine vordefinierte Auswahl von Werten. Sonst, beispielsweise bei Zeichenfolgen, erscheint ein Textfeld für die Eingabe eines Werts.

Sie haben eine Anweisung für das Entscheidungselement erstellt. Wenn das Entscheidungselement den Eingabeparameter erhält, vergleicht es den Wert des Eingabeparameter mit dem Wert in der Anweisung und ermittelt, ob die Anweisung wahr oder falsch ist.

### Weiter

Sie müssen festlegen, wie der Workflow mit Ausnahmen umgeht.

## Ausnahmebehandlung

Ausnahmebehandlung erfasst alle Fehler, die beim Ausführen eines Schemaelements auftreten. Sie definiert, wie sich das Schemaelement beim Auftreten des Fehlers verhält.

Alle Elemente in einem Workflow, außer Entscheidungs-, Start- und Endelemente, enthalten einen bestimmten Ausgabeparametertyp, der nur der Behandlung von Ausnahmen dient. Wenn für ein Element während der Ausführung ein Fehler auftritt, kann es ein Fehlersignal an einen Ausnahmehandler senden. Ausnahmehandler erfassen den Fehler und reagieren dementsprechend. Wenn die definierten Ausnahmehandler einen bestimmten Fehler nicht verarbeiten, können Sie den Ausnahme-Ausgabeparameter eines Element an ein Ausnahmeelement bindet, wodurch die Workflowausführung mit dem Status Fehlgeschlagen beendet wird.

Ausnahmen agieren als try- und catch-Sequenz innerhalb eines Flowelemente. Wenn eine bestimmte Ausnahme in einem Element nicht verarbeitet werden muss, ist eine Bindung des Ausnahme-Ausgabeparameter des Elements nicht erforderlich.

Der Ausgabeparametertyp für Ausnahmen ist immer ein `errorCode`-Objekt.



## Erstellen von Ausnahmebindungen


Elemente können Bindungen festlegen, die definieren, wie sich der Workflow verhält, wenn ein Fehler in diesem Element auftritt.

### Voraussetzungen

Vergewissern Sie sich, dass die Registerkarte **Schema** des Workfloweditors Elemente enthält.

### Vorgehensweise

- 1 Führen Sie den Mauszeiger auf das Element, für das Sie eine Ausnahmebindung definieren möchten.  
Ein roter Pfeil erscheint rechts von dem Element.
- 2 Führen Sie den Mauszeiger auf den roten Pfeil, bis er größer wird, halten Sie die linke Maustaste gedrückt und ziehen Sie den roten Pfeil auf das Zielement.  
Ein roter gestrichelter Pfeil verbindet die beiden Elemente. Das Zielement definiert das Verhalten des Workflows, wenn bei dem Element, das damit verknüpft ist, ein Fehler auftritt.
- 3 Klicken Sie auf das Symbol **Bearbeiten** () des Elements, das die Verknüpfung mit dem Element für die Ausnahmebehandlung herstellt.
- 4 Klicken Sie auf die Registerkarte **Ausnahme** auf den Eigenschaftenregisterkarten für das Schemaelement.
- 5 Zum Einrichten des Werts für die **Ausnahmebindung der Ausgabe** klicken Sie auf **Nicht festgelegt**.
  - Wählen Sie einen Parameter zum Binden an den Ausnahme-Ausgabeparameter aus dem Dialogfeld für das Binden des Ausnahmeattributs und klicken Sie auf **Auswählen**.
  - Klicken Sie auf **Parameter/Attribut in Workflow erstellen**, um einen Ausnahme-Ausgabeparameter zu erstellen.
- 6 Klicken Sie auf das Zielement, das das Verhalten für die Ausnahmebehandlung definiert.
- 7 Klicken Sie auf die Registerkarte **EIN** auf den Eigenschaftenregisterkarten für das Schemaelement.
- 8 Klicken Sie auf das Symbol **An Workflowparameter/-attribut binden** ()  
Das Dialogfeld zum Auswählen des Eingabeparameters wird angezeigt.
- 9 Wählen Sie den Ausnahme-Ausgabeparameter und klicken Sie auf **Auswählen**.

- 10 Klicken Sie auf den Eigenschaftenregisterkarten für das Schemaelement auf die Registerkarte **AUS** für das Ausnahmebehandlungs-Element.
- 11 Definieren Sie das Verhalten des Ausnahmebehandlungs-Element.
  - Klicken Sie auf das Symbol **An Workflowparameter/-attribut binden** () , um einen Ausgabeparameter für das zu generierende Ausnahmebehandlungs-Element auszuwählen.
  - Klicken Sie auf die Registerkarte **Skripterstellung** und verwenden Sie JavaScript, um das Verhalten des Ausnahmebehandlungs-Elements zu definieren.

Sie haben definiert, wie das Element Ausnahmen behandelt.

### Weiter

Sie müssen definieren, wie Eingabeparameter von Benutzern bezogen werden, wenn diese den Workflow ausführen.

## Verwenden von Fehlerhandlern

Sie können einen standardmäßigen Fehlerhandler verwenden, um das Verhalten bei Fehlern bei bestimmten Workflowschema-Elementen festzulegen. Sie können einen globalen Fehlerhandler verwenden, um das Verhalten bei Fehlern festzulegen, die nicht durch den Standard-Fehlerhandler erfasst werden.

### Hinzufügen eines Fehlerhandlers zu einem Workflow

Sie können definieren, wie Fehler in einem bestimmten Workflowelement während der Workflowausführung gehandhabt werden, indem Sie dem Workflowelement einen Fehlerhandlers hinzufügen. Sie können einen Fehlerhandler nur Workflowelementen hinzufügen, die keinen bestimmten Fehlerpfad haben.

---

**WICHTIG** Workflows, die ein **Handlerfehler**-Element enthalten, sind nicht mit Orchestrator 5.5.x oder früher kompatibel.

---

### Voraussetzungen

- Erstellen Sie einen Workflow.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.
- Fügen Sie dem Workflowschema einige Elemente hinzu.

### Vorgehensweise

- 1 Ziehen Sie ein **Handlerfehler**-Element in das entsprechende Element im Workflowschema.  
Ein Dialogfeld wird angezeigt.
- 2 Wählen Sie im Dropdown-Menü im Dialogfeld, wie Fehler behandelt werden sollen.

Option	Beschreibung
<b>Ausnahme auslösen</b>	Wenn ein Fehler auftritt, wird eine Ausnahme ausgelöst. Sie können die Ausnahmebindung ändern.
<b>Workflow aufrufen</b>	Wenn ein Fehler auftritt, wird ein ausgewählter Workflow ausgeführt.
<b>Benutzerdefiniertes Skript</b>	Wenn ein Fehler auftritt, wird ein benutzerdefiniertes Skript ausgeführt.

- 3 Klicken Sie auf **Auswählen**.

Sie haben einem Workflow einen Fehlerhandler hinzugefügt. Wenn der Workflow dieses Element erreicht, führt er die ausgewählte Aktion durch, bevor seine Ausführung beendet wird.

## Hinzufügen eines globalen Fehlerhandlers zu einem Workflow

Sie können durch Hinzufügen eines globalen Fehlerhandlers zum Workflowschema definieren, wie Fehler während einer Workflowausführung, die nicht von den Standardfehlerhandlern entdeckt werden, gehandhabt werden. Sie können einem Workflowschema einen globalen Fehlerhandler hinzufügen.

---

**WICHTIG** Workflows, die ein **Standardfehlerhandler**-Element enthalten, sind nicht mit Orchestrator 5.5.x oder niedriger kompatibel.

---

### Voraussetzungen

- Erstellen Sie einen Workflow.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.
- Fügen Sie dem Workflowschema einige Elemente hinzu.

### Vorgehensweise

- 1 Ziehen Sie ein **Standardfehlerhandler**-Element in das Workflowschema.
- 2 (Optional) Fügen Sie Schemaelemente zwischen den Elementen **Standardfehlerhandler** und **Ausnahme auslösen** ein, um anzugeben, wie globale Workflowfehler gehandhabt werden sollen.

Sie haben einem Workflow einen globalen Fehlerhandler hinzugefügt. Wenn in dem Workflow ein Fehler auftritt, der von den Standard-Fehlerhandlern nicht erkannt wird, führt der globale Fehlerhandler die festgelegten Aktionen durch, bevor die Workflowausführung beendet wird.

## Foreach-Elemente und zusammengesetzte Typen

Sie können ein Foreach-Element in den Workflow einfügen, sodass ein untergeordneter Workflow ausgeführt wird, der Arrays von Parametern oder Attributen durchläuft. Um die Verständlichkeit und Lesbarkeit eines Workflows zu verbessern, können Sie mehrere Workflowparameter verschiedenen Typs, die logisch verbunden sind, in einem zusammengesetzten Typ zusammenfassen.

### Verwenden von Foreach-Elementen

Jedes Foreach-Element durchläuft einen untergeordneten Workflow für ein Array von Eingabeparametern oder -attributen. Sie können die Arrays auswählen, für die der untergeordnete Workflow ausgeführt werden soll, und die Werte für die Elemente des Arrays beim Ausführen des Workflows übergeben. Der untergeordnete Workflow wird so oft ausgeführt, wie Elemente in dem Array festgelegt sind.

Wenn ein Konfigurationselement vorhanden ist, das ein Attributarray enthält, können Sie einen Workflow ausführen, der die Attribute eines Foreach-Elements durchläuft.

Beispiel: Angenommen, Sie haben 10 virtuelle Maschinen in einem Ordner, die Sie umbenennen möchten. Dazu fügen Sie ein Foreach-Element in einen Workflow ein und legen in dem Element den Workflow zum Umbenennen der virtuellen Maschinen als untergeordneten Workflow fest. Der Workflow zum Umbenennen der virtuellen Maschinen erhält zwei Eingabeparameter: den Namen einer virtuellen Maschine und ihren neuen Namen. Sie können diese Parameter als Eingabeparameter auf den aktuellen Workflow anwenden. Dadurch werden sie zu Arrays, für die der Workflow zum Umbenennen der virtuellen Maschinen durchlaufen wird. Beim Ausführen des Workflows können Sie die 10 virtuellen Maschinen im Ordner sowie ihre neuen Namen angeben. Bei jeder Ausführung verwendet der Workflow ein Element aus dem Array der virtuellen Maschinen und ein Element aus dem Array der neuen Namen für die virtuellen Maschinen.

## Verwenden von zusammengesetzten Typen

Ein zusammengesetzter Typ ist eine Gruppe mehrerer Eingabeparameter oder -attribute, die logisch verbunden sind, aber unterschiedliche Typen besitzen. In einem Foreach-Element können Sie eine Gruppe von Parametern als Composite-Wert zusammenfassen. Dadurch verwendet das Foreach-Element die Werte der gruppierten Parameter gleichzeitig bei jeder nachfolgenden Ausführung des Workflows.

Beispiel: Angenommen, Sie möchten eine virtuelle Maschine umbenennen. Sie benötigen das Objekt der virtuellen Maschine und den neuen Namen. Wenn Sie mehrere virtuelle Maschinen umbenennen, benötigen Sie zwei Arrays: eines für die virtuellen Maschinen und eines für ihre Namen. Diese beiden Arrays sind nicht explizit miteinander verbunden. Bei einem zusammengesetzten Typ können Sie ein Array verwenden, bei dem jedes Element die virtuelle Maschine sowie ihren neuen Namen enthält. Auf diese Weise wird die Verbindung zwischen den beiden Parametern im Fall mehrerer Werte explizit festgelegt und nicht durch das Workflowschema impliziert.

---

**HINWEIS** Workflows, die zusammengesetzte Typen enthalten, können nicht auf dem vSphere Web Client ausgeführt werden.

---

## Definieren eines Foreach-Elements

Wenn Sie einen untergeordneten Workflow mehrfach ausführen und bei jeder Ausführung unterschiedliche Parameter oder Attribute verwenden möchten, können Sie ein Foreach-Element in den übergeordneten Workflow einfügen.

Wenn Sie ein Foreach-Element einfügen, müssen Sie mindestens ein Array auswählen, über das das Foreach-Element eine Iteration durchführt. Ein Array-Element kann verschiedene Werte für jede der aufeinander folgenden Ausführungen des Workflows aufweisen.

Wenn der untergeordnete Workflow Ausgabeparameter besitzt, sollten Sie die Ausgabeparameter des Foreach-Elements auswählen, in denen die Ausgabe des Workflows erfasst werden soll, damit der untergeordnete Workflow auch diese iterativ durchlaufen kann.

### Voraussetzungen

Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.

### Vorgehensweise

- 1 Wählen Sie im Workfloweditor die Registerkarte **Schema** aus.
- 2 Ziehen Sie aus dem Menü **Generisch** ein Foreach-Element in das Workflowschema.
- 3 Wählen Sie aus dem Dialogfeld für die Auswahl einen Workflow aus.

Die folgende Benachrichtigung wird im Schema-Fensterbereich oben angezeigt.

Möchten Sie die Parameter der Aktivität dem aktuellen Workflow als Eingabe/Ausgabe hinzufügen?

- 4 Klicken Sie in der Benachrichtigung auf **Einrichten**.

Ein Popup-Fenster mit den verfügbaren Optionen wird angezeigt.

- 5 Wählen Sie den Zuordnungstyp für jeden einzelnen Eingabeparameter aus.

Option	Beschreibung
<b>Eingabe</b>	Das Argument wird einem Workflow-Eingabeparameter zugeordnet.
<b>Überspringen</b>	Das Argument wird einem NULL-Wert zugeordnet.
<b>Wert</b>	Das Argument wird einem Attribut zugeordnet, dessen Wert Sie über die Spalte „Wert“ festlegen können.

- 6 Wählen Sie den Zuordnungstyp für jeden einzelnen Ausgabeparameter aus.

Option	Beschreibung
<b>Output</b>	Das Argument wird einem Workflow-Ausgabeparameter zugeordnet.
<b>Überspringen</b>	Das Argument wird einem NULL-Wert zugeordnet.
<b>Lokale Variable</b>	Das Argument wird einem Attribut zugeordnet.

- 7 Klicken Sie auf **Heraufstufen**.
- 8 Klicken Sie mit der rechten Maustaste auf das Foreach-Element und wählen Sie **Synchronisieren > Präsentation synchronisieren**.  
Ein Bestätigungsdialogfeld wird angezeigt.
- 9 Klicken Sie auf **OK**, um die Präsentation des Foreach-Elements an den aktuellen Workflow weiterzugeben.  
Ein Dialogfeld zeigt Informationen zum Ergebnis des Vorgangs.
- 10 Überprüfen Sie auf der Registerkarte **Eingaben**, ob die Parameter des untergeordneten Workflows als Elemente vom Typ Array hinzugefügt wurden.
- 11 Überprüfen Sie auf der Registerkarte **Ausgaben**, ob die Parameter des untergeordneten Workflows als Elemente vom Typ Array hinzugefügt wurden.

Sie haben ein Foreach-Element in Ihrem Workflow erstellt. Das Foreach-Element führt einen Workflow aus, der als Parameter die Elemente des Arrays aus Parametern oder Attributen verwendet, das Sie definiert haben.

Bei Parametern oder Attributen, die nicht als Arrays definiert sind, verwendet der Workflow bei jeder der aufeinander folgenden Ausführungen denselben Wert.

#### Beispiel: Umbenennen virtueller Maschinen mithilfe eines Foreach-Elements

Sie können mit einem Foreach-Element mehrere virtuelle Maschinen gleichzeitig umbenennen. Sie müssen ein Foreach-Element in einen Workflow einfügen und die Parameter `vm` und `newName` als Eingabe für den aktuellen Workflow weitergeben. Auf diese Art geben Sie bei Ausführung des Workflows die umzubenennenden virtuellen Maschinen sowie die neuen Namen an. Die virtuellen Maschinen sind als Elemente in dem Array enthalten, das Sie für den Parameter `vm` erstellt haben. Die neuen Namen für die virtuellen Maschinen sind in dem Array enthalten, das Sie für den Parameter `newName` erstellt haben.

#### Definieren eines zusammengesetzten Typs in einem Foreach-Element

Sie können mehrere Workflowparameter gruppieren, die dann in einem neuen Typ logisch verbunden werden. Dieser Typ wird zusammengesetzter Typ genannt. Sie können ein Foreach-Element zum Binden einer Parametergruppe als zusammengesetzten Wert verwenden, um mehrere Parameter-Arrays in einem einzelnen Array zu verbinden.

#### Voraussetzungen

- Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.
- Vergewissern Sie sich, dass sich ein Foreach-Element in Ihrem Workflow befindet.

#### Vorgehensweise

- 1 Wählen Sie die Registerkarte **EINGABE** oder **AUSGABE** des Foreach-Elements.
- 2 Wählen Sie einen lokalen Parameter aus, den Sie mit anderen lokalen Parametern in einem zusammengesetzten Typ gruppieren möchten.

- 3 Klicken Sie auf **Parametergruppe als zusammengesetzten Wert binden** oben auf der Registerkarte **EINGABE** oder **AUSGABE**.
- 4 Wählen Sie im Bereich „Bindung“ die Parameter aus, die Sie als zusammengesetzten Typ gruppieren möchten.
- 5 Wählen Sie **Als Iterator binden**.  
Sie haben das Foreach-Element so eingestellt, dass es eine Iteration über das Array eines zusammengesetzten Typs ausführt.
- 6 Klicken Sie auf **Akzeptieren**.

Sie haben einen zusammengesetzten Typ definiert und sichergestellt, dass der Workflow eine Iteration über diesen zusammengesetzten Typ ausführt. Die als zusammengesetzter Typ gruppierten Parameter werden mit *Composite\_Type\_Name* benannt. *Parameter\_Name*. Wenn Sie beispielsweise einen zusammengesetzten Typ `snapshots` erstellen, können die in diesem Typ gruppierten Parameter `snapshots.vm[in-parameter]` oder `snapshots.name[in-parameter]` sein. Jedes Element des Arrays des zusammengesetzten Typs enthält eine einzelne Instanz jedes Parameters, den Sie im zusammengesetzten Typ gruppiert haben.

### Beispiel: Umbenennen virtueller Maschinen

Angenommen, Sie möchten 10 virtuelle Maschinen gleichzeitig umbenennen. Dazu fügen Sie in einen Workflow ein Foreach-Element ein und wählen im Element den Workflow zum Umbenennen der virtuellen Maschinen aus. Sie erstellen einen zusammengesetzten Typ, um die Parameter `vm` und `newName` explizit zu verbinden. Sie binden den zusammengesetzten Typ als Iterator und erstellen ein einzelnes Array, das sowohl den Parameter `vm` als auch den Parameter `newName` enthält.

## Hinzufügen einer Switch-Aktivität zu einem Workflow

Sie können einem Workflowschema eine einfache Switch-Aktivität hinzufügen, die die Switch-Fälle auf Basis von Workflowattributen oder -parametern definiert.

Jede Switch-Aktivität kann mehrere Switch-Fälle haben. Jeder Switch-Fall wird durch eine Bedingung definiert, die sich auf ein Attribut oder einen Parameter bezieht. Wenn die Bedingung erfüllt wurde, wechselt die Workflowausführung zu einem entsprechenden Workflowelement, das Sie definiert haben. Wenn keine der spezifischen Bedingungen erfüllt wurde, wechselt die Workflowausführung zu einem Standard-Workflowelement, das Sie definiert haben.

---

**WICHTIG** Workflows, die ein **Switch**-Element enthalten, sind nicht mit Orchestrator 5.5.x oder früher kompatibel.

---

### Voraussetzungen

Vergewissern Sie sich, dass die Registerkarte **Schema** des Workfloweditors Elemente enthält.

### Vorgehensweise

- 1 Ziehen Sie ein **Switch**-Element in das entsprechende Element im Workflowschema.
- 2 Klicken Sie auf das Symbol **Bearbeiten** (✎) des **Switch**-Elements.
- 3 Auf der Registerkarte **Fälle** können Sie Switch-Fälle hinzufügen oder löschen.  
Sie können die Priorität der Switch-Fälle ändern.
- 4 Definieren Sie die Bedingung für jeden Switch-Fall.
- 5 Wählen Sie das entsprechende Workflowelement für jeden Switch-Fall.
- 6 Wählen Sie das Standard-Workflowelement, zu dem Sie wechseln wollen.
- 7 Klicken Sie auf **Schließen**.



8 Klicken Sie auf **Speichern**.

Sie haben die Bedingungen des Switch-Falls und die Workflowpfade definiert.

## Entwickeln von Plug-Ins

Orchestrator ermöglicht dank seiner offenen Plug-In-Architektur die Integration in Management- und Verwaltungslösungen. Sie nutzen den Orchestrator-Client zum Ausführen und Erstellen von Plug-In-Workflows und zum Zugriff auf die Plug-In-API.

### Überblick über Plug-Ins

Orchestrator-Plug-Ins müssen einen Standardsatz von Komponenten enthalten und eine Standardarchitektur aufweisen. Diese Verfahren unterstützen Sie beim Erstellen von Plug-Ins für die weitest mögliche Varianz an externen Technologien.

- [Struktur eines Orchestrator-Plug-Ins](#) auf Seite 50  
Orchestrator-Plug-Ins haben eine grundlegende Struktur, die aus verschiedenen Typen von Ebenen besteht, die spezielle Funktionen implementieren.
- [Verfügbarmachen einer externen API in Orchestrator](#) auf Seite 51  
Die API eines externen Produkts können Sie durch Erstellen eines Orchestrator-Plug-Ins auf der Orchestrator-Plattform verfügbar machen. Ein solches Plug-In kann für jede Anwendung erstellt werden, deren API durch Zuordnung zu JavaScript-Objekten, die von Orchestrator verwendet werden können, verfügbar gemacht werden kann.
- [Komponenten eines Plug-Ins](#) auf Seite 51  
Plug-Ins bestehen aus einem Standardsatz von Komponenten, die die Objekte in der Plug-In-Technologie für die Orchestrator-Plattform verfügbar machen.
- [Rolle der Datei vso.xml](#) auf Seite 53  
Sie verwenden die Datei `vso.xml`, um die Objekte, Klassen, Methoden und Attribute der integrierten Technologie zu Bestandslistenobjekten, Skripterstellungstypen, Skripterstellungsklassen, Skripterstellungsmethoden und Attributen von Orchestrator zuzuordnen. Außerdem wird in der Datei `vso.xml` das Start- und Konfigurationsverhalten des Plug-Ins definiert.
- [Rollen des Plug-In-Adapters](#) auf Seite 53  
Der Plug-In-Adapter ist der Einstiegspunkt des Plug-Ins in den Orchestrator-Server. Der Plug-In-Adapter dient als Datenspeicher für die Plug-In-Technologie im Orchestrator-Server, erstellt eine Plug-In-Factory und verwaltet Ereignisse, die in der Plug-In-Technologie auftreten.
- [Rollen der Plug-In-Factory](#) auf Seite 54  
Die Plug-In-Factory definiert, wie Orchestrator Objekte in der Plug-In-Technologie findet und damit arbeitet.
- [Rolle von Finder-Objekten](#) auf Seite 55  
Finder-Objekte erkennen und finden bestimmte Instanzen von verwalteten Objekttypen in der integrierten Technologie. Orchestrator kann Objekte, die in der integrierten Technologie gefunden werden, ändern und mit ihnen interagieren, indem es Workflows für die Finder-Objekte ausführt.
- [Rolle von Skriptobjekten](#) auf Seite 55  
Skriptobjekte sind JavaScript-Darstellungen von Objekten aus der integrierten Technologie. Skriptobjekte aus Plug-Ins werden in der Orchestrator-JavaScript-API angezeigt. Sie können sie in Elementen mit Skript in Workflows und Aktionen verwenden.

■ [Rolle von Ereignishandlern](#) auf Seite 56

Ereignisse sind Änderungen an Zuständen oder Attributen der Objekte, die Orchestrator in der Plug-In-Technologie findet. Orchestrator überwacht Ereignisse durch Implementieren von Ereignishandlern.

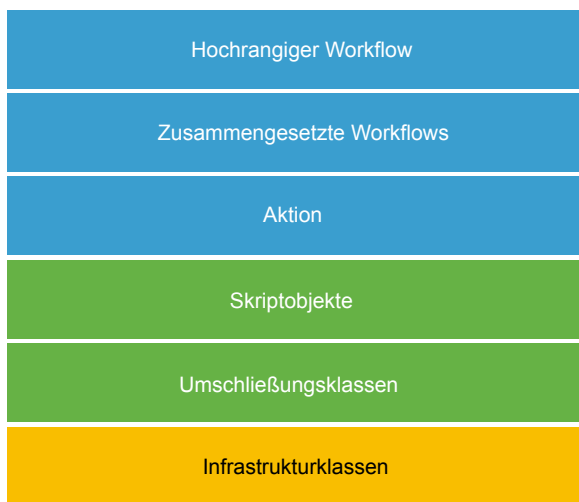
## Struktur eines Orchestrator-Plug-Ins

Orchestrator-Plug-Ins haben eine grundlegende Struktur, die aus verschiedenen Typen von Ebenen besteht, die spezielle Funktionen implementieren.

Die unteren drei Ebenen eines Orchestrator-Plug-Ins, also die Infrastrukturklassen, Umschließungsklassen und Skriptobjekte, implementieren die Verbindung zwischen Plug-In-Technologie und Orchestrator.

Die für den Benutzer sichtbaren Ebenen eines Orchestrator-Plug-Ins sind die oberen drei Ebenen: Aktionen, Bausteine und hochrangige Workflows.

**Abbildung 1-3.** Struktur eines Orchestrator-Plug-Ins



### Infrastrukturklassen

Ein Satz an Klassen, die die Verbindung zwischen Plug-In-Technologie und Orchestrator herstellen. Die Infrastrukturklassen umfassen die Klassen, die entsprechend der Plug-In-Definition implementiert werden (wie Plug-In-Factory, Plug-In-Adaptor usw.). Die Infrastrukturklassen umfassen auch die Klassen, die Funktionen für häufige Aufgaben und Objekte wie Hilfsprogramme, Zwischenspeichern, Bestandslisten usw. bereitstellen.

### Umschließungsklassen

Ein Satz an Klassen, die das Objektmodell der Plug-In-Technologie an das Objektmodell anpassen, das Sie im Orchestrator darstellen möchten.

### Skriptobjekte

JavaScript-Objekttypen, die Zugriff auf Umschließungsklassen, Methoden und Attribute in der Plug-In-Technologie bieten. In der Datei `vso.xml` definieren Sie Umschließungsklassen, Attribute und Methoden aus der Plug-In-Technologie, die im Orchestrator dargelegt wird.

### Aktionen

Ein Satz aus JavaScript-Funktionen, die Sie direkt in Workflows und Skriptaufgaben verwenden können. Aktionen können mehrere Eingabeparameter besitzen und haben einen einzelnen Rückgabewert.

**Zusammengesetzte Workflows**

Ein Satz aus Workflows, die alle generischen Funktionen abdecken, die Sie mit dem Plug-In bereitstellen möchten. Normalerweise stellt ein zusammengesetzter Workflow einen Vorgang in der Benutzerschnittstelle der Orchestrator-gesteuerten Technologie dar. Die zusammengesetzten Workflows können direkt verwendet oder in hochrangige Workflows einbezogen werden.

**Hochrangige Workflows**

Ein Satz aus Workflows, der spezielle Funktionen des Plug-Ins abdeckt. Sie können hochrangige Workflows bereitstellen, um konkrete Anforderungen zu erfüllen, oder um komplexe Beispiele der Plug-In-Nutzung zu zeigen.

**Verfügbarmachen einer externen API in Orchestrator**

Die API eines externen Produkts können Sie durch Erstellen eines Orchestrator-Plug-Ins auf der Orchestrator-Plattform verfügbar machen. Ein solches Plug-In kann für jede Anwendung erstellt werden, deren API durch Zuordnung zu JavaScript-Objekten, die von Orchestrator verwendet werden können, verfügbar gemacht werden kann.

Die Plug-Ins ordnen Java-Objekte und -Methoden JavaScript-Objekten zu, die sie dann der Orchestrator-Skript-API hinzufügen. Wird von einer externen Anwendung eine Java-API verfügbar gemacht, können Sie diese unmittelbar JavaScript-Objekten zuordnen, die Orchestrator dann für Workflows und Aktionen verwenden kann.

Für Anwendungen, deren API in einer anderen Sprache als Java verfügbar gemacht wird, können Sie Plug-Ins mithilfe von WSDL (Web Service Definition Language), REST (Representational State Transfer) oder eines Messaging-Diensts erstellen. Die Plug-Ins integrieren die verfügbar gemachte API dann in Java-Objekte. Diese integrierten Java-Objekte können Sie anschließend JavaScript-Objekten zuordnen, die mit Orchestrator kompatibel sind.

Die Plug-In-Technologie ist unabhängig von Orchestrator. Sie können Orchestrator-Plug-Ins für externe Produkte selbst dann erstellen, wenn Sie statt auf den Quellcode nur auf den binären Code zugreifen können. Dies gilt beispielsweise für Java-Archive (JAR-Dateien).

**Komponenten eines Plug-Ins**

Plug-Ins bestehen aus einem Standardsatz von Komponenten, die die Objekte in der Plug-In-Technologie für die Orchestrator-Plattform verfügbar machen.

Die wichtigsten Komponenten eines Plug-Ins sind der Plug-In-Adapter, die Factory und Ereignisimplementierungen. Sie ordnen die Objekte und Vorgänge, die im Adapter, in der Factory und in den Ereignisimplementierungen definiert sind, Orchestrator-Objekten in einer XML-Definitionsdatei namens `vso.xml` zu. Die `vso.xml`-Datei ordnet Objekte und Funktionen aus der Plug-In-Technologie den JavaScript-Skriptobjekten zu, die in der Orchestrator JavaScript-API erscheinen. Die `vso.xml`-Datei ordnet auch Objekttypen aus der Plug-In-Technologie Findern zu, die auf der Orchestrator-Registerkarte **Bestand** erscheinen.

Plug-Ins bestehen aus den folgenden Komponenten.

**Plug-In-Modul**

Das Plug-In selbst, das als eine Gruppe von Java-Klassen definiert wird, eine `vso.xml`-Datei und Pakete der Workflows und Aktionen, die mit den Objekten interagieren, auf die Sie über das Plug-In zugreifen. Das Plug-In-Modul ist unbedingt erforderlich.

**Plug-In-Adapter**

Definiert die Schnittstelle zwischen der Plug-In-Technologie und dem Orchestrator-Server. Der Adapter ist der Einstiegspunkt des Plug-Ins in die Orchestrator-Plattform. Der Adapter erstellt die Plug-In-Factory, verwaltet das Laden und Entladen des Plug-Ins sowie die Ereignisse, die im Zusammenhang mit den Objekten in der Plug-In-Technologie auftreten. Der Plug-In-Adapter ist unbedingt erforderlich.

<b>Plug-In-Factory</b>	Definiert, wie Orchestrator Objekte in der Plug-In-Technologie findet und damit Vorgänge vornimmt. Der Adapter erstellt eine Factory für die Client-sitzung, die sich zwischen Orchestrator und einer Plug-In-Technologie öffnet. Die Factory ermöglicht es Ihnen einerseits, eine Sitzung zwischen allen Clientverbindungen gemeinsam zu nutzen, oder andererseits eine Sitzung pro Clientverbindung zu öffnen. Die Plug-In-Factory ist unbedingt erforderlich.
<b>Konfiguration</b>	Orchestrator definiert keine Standardart, wie das Plug-In seine Konfiguration speichert. Sie können Konfigurationsinformationen mithilfe von Windows-Registern, statischen Konfigurationsdateien, einer Datenbank oder in XML-Dateien speichern. Orchestrator-Plug-Ins können durch das Ausführen von Konfigurationsworkflows im Orchestrator-Client konfiguriert werden.
<b>Finder</b>	Interaktionsregeln, die definieren, wie Orchestrator Objekte in der Plug-In-Technologie ermittelt und darstellt. Finder rufen Objekte aus der Gruppe von Objekten ab, die die Plug-In-Technologie für Orchestrator bereitstellt. Sie definieren in der <code>vso.xml</code> -Datei die Beziehungen zwischen Objekten, um die Navigation durch das Netzwerk von Objekten zu ermöglichen. Orchestrator stellt das Objektmodell der Plug-In-Technologie auf der Registerkarte <b>Bestand</b> dar. Finder sind unbedingt erforderlich, wenn Sie Objekte in der Plug-In-Technologie für Orchestrator verfügbar machen möchten.
<b>Skriptobjekte</b>	JavaScript-Objekttypen, die Zugriff auf Objekte, Vorgänge und Attribute in der Plug-In-Technologie bieten. Skriptobjekte definieren, wie Orchestrator auf das Objektmodell der Plug-In-Technologie über JavaScript zugreift. Sie ordnen die Klassen und Methoden der Plug-In-Technologie den JavaScript-Objekten in der <code>vso.xml</code> -Datei zu. Sie können auf die JavaScript-Objekte in der Orchestrator-Skripterstellung-API zugreifen und sie in Orchestrator-Skript Aufgaben, Aktionen und Workflows integrieren. Skriptobjekte sind unbedingt erforderlich, wenn Sie Skripttypen, Klassen und Methoden der Orchestrator-JavaScript-API hinzufügen möchten.
<b>Bestandsliste</b>	Instanzen von Objekten in der Plug-In-Technologie, die Orchestrator mithilfe von Findern erkennt, erscheinen in der Ansicht <b>Bestand</b> im Orchestrator-Client. Sie können mit den Objekten im Bestand arbeiten, indem Sie an ihnen Workflows ausführen. Der Bestand ist optional. Sie können ein Plug-In erstellen, das nur Skripttypen und Klassen der Orchestrator-JavaScript-API hinzufügt und keine Instanzen von Objekten im Bestand verfügbar macht.
<b>Ereignisse</b>	Änderungen des Status eines Objekts in der Plug-In-Technologie. Orchestrator kann passiv auf Ereignisse warten, die in der Plug-In-Technologie auftreten. Orchestrator kann auch aktiv Ereignisse in der Plug-In-Technologie auslösen. Ereignisse sind optional.

## Rolle der Datei vso.xml

Sie verwenden die Datei `vso.xml`, um die Objekte, Klassen, Methoden und Attribute der integrierten Technologie zu Bestandslistenobjekten, Skripterstellungstypen, Skripterstellungsklassen, Skripterstellungsmethoden und Attributen von Orchestrator zuzuordnen. Außerdem wird in der Datei `vso.xml` das Start- und Konfigurationsverhalten des Plug-Ins definiert.

Die Datei `vso.xml` erfüllt hauptsächlich die folgenden Rollen.

<b>Start- und Konfigurationsverhalten</b>	Definiert, wie das Plug-In gestartet wird, und findet durch das Plug-In definierte Konfigurationsimplementierungen. Lädt den Plug-In-Adapter.
<b>Bestandslistenobjekte</b>	Definiert die Objekttypen, auf die das Plug-In in der integrierten Technologie zugreift. Mit den Finder-Methoden der Plug-In-Factory-Implementierung werden Instanzen dieser Objekte gefunden und in der Orchestrator-Bestandsliste angezeigt.
<b>Skripterstellungstypen</b>	Fügt der Orchestrator-JavaScript-API Skripterstellungstypen hinzu, um die verschiedenen Objekttypen in der Bestandsliste darzustellen. Sie können diese Skripterstellungstypen als Eingabeparameter in Workflows verwenden.
<b>Skripterstellungsklassen</b>	Fügt der Orchestrator-JavaScript-API Klassen hinzu, die Sie in Elementen mit Skript in Workflows, Aktionen, Richtlinien usw. verwenden können.
<b>Skripterstellungsmethoden</b>	Fügt der Orchestrator-JavaScript-API Methoden hinzu, die Sie in Elementen mit Skript in Workflows, Aktionen, Richtlinien usw. verwenden können.
<b>Skriptstellungsattribute</b>	Fügt der Orchestrator-JavaScript-API die Attribute der Objekte aus der integrierten Technologie hinzu, die Sie in Elementen mit Skript in Workflows, Aktionen, Richtlinien usw. verwenden können.

## Rollen des Plug-In-Adapters

Der Plug-In-Adapter ist der Einstiegspunkt des Plug-Ins in den Orchestrator-Server. Der Plug-In-Adapter dient als Datenspeicher für die Plug-In-Technologie im Orchestrator-Server, erstellt eine Plug-In-Factory und verwaltet Ereignisse, die in der Plug-In-Technologie auftreten.

Zum Erstellen eines Plug-In-Adapters erstellen Sie eine Java-Klasse, die die Schnittstelle `IPluginAdaptor` implementiert.

Die Plug-In-Adapter-Klasse, die Sie erstellen, verwaltet die Plug-In-Factory, Ereignisse und Auslöser in der Plug-In-Technologie. Die Schnittstelle `IPluginAdaptor` bietet Methoden, die Sie zum Ausführen dieser Aufgaben verwenden.

Der Plug-In-Adapter erfüllt hauptsächlich die folgenden Rollen.

<b>Erstellt eine Factory</b>	Die wichtigste Rolle für den Plug-In-Adapter besteht darin, eine Plug-In-Factory-Instanz für jede Verbindung vom Orchestrator in die Plug-In-Technologie zu laden und zu entladen. Die Plug-In-Adapter-Klasse ruft die Methode <code>IPluginAdaptor.createPluginFactory()</code> auf, um eine Instanz einer Klasse zu erstellen, die die Schnittstelle <code>IPluginFactory</code> implementiert.
<b>Verwalten von Ereignissen</b>	Der Plug-In-Adapter ist die Schnittstelle zwischen dem Orchestrator-Server und der Plug-In-Technologie. Der Plug-In-Adapter verwaltet die Ereignisse, die der Orchestrator für die Objekte in einer Plug-In-Technologie ausführt oder überwacht. Der Adapter verwaltet Ereignisse durch Ereignisherausgeber. Ereignisherausgeber sind Instanzen der Schnittstelle <code>IPluginEventPublisher</code> , die der Adapter durch den Aufruf der Methode <code>IPluginAdaptor.registerEventPublisher()</code> erstellt. Ereignisherausgeber setzen Auslöser und

Kontrollen auf Objekte in der Plug-In-Technologie, damit Orchestrator definierte Aktionen starten kann, wenn bestimmte Ereignisse mit dem Objekt eintreten oder die Werte des Objekts bestimmte Schwellenwerte überschreiten. Sie können ferner `PluginTrigger`- und `PluginWatcher`-Instanzen definieren, die Ereignisse festlegen, auf die „Warteereignis“-Elemente in Workflows mit langen Ausführungszeiten warten.

**Legt den Plug-In-Namen fest**

Sie übergeben einen Namen für das Plug-In in der `vso.xml`-Datei. Der Plug-In-Adapter bezieht diesen Namen aus der `vso.xml`-Datei und veröffentlicht ihn in der Ansicht **Bestand** im Orchestrator-Client.

**Installiert Lizenzen**

Sie können Methoden aufrufen, um Lizenzdateien zu installieren, die die Plug-In-Technologie in der Adapterimplementierung erfordert.

Vollständige Details zur `IPluginAdaptor`-Schnittstelle, zu allen ihren Methoden und allen anderen Klassen der Plug-In-API finden Sie unter „[Orchestrator Plug-In-API-Referenz](#)“, auf Seite 61.

## Rollen der Plug-In-Factory

Die Plug-In-Factory definiert, wie Orchestrator Objekte in der Plug-In-Technologie findet und damit arbeitet.

Zum Erstellen einer Plug-In-Factory müssen Sie die Schnittstelle `IPluginFactory` aus der Orchestrator-Plug-In-API implementieren und erweitern. Die Plug-In-Factory-Klasse, die Sie erstellen, definiert die Finderfunktionen, die Orchestrator verwendet, um auf Objekte in der Plug-In-Technologie zuzugreifen. Die Factory ermöglicht es dem Orchestrator-Server, Objekte nach ihrem Bezeichner, ihrer Beziehung mit anderen Objekten oder durch eine Suche nach einer Abfragezeichenfolge zu finden.

Die Plug-In-Factory übernimmt hauptsächlich folgende Aufgaben.

**Suche nach Objekten**

Sie können Funktionen erstellen, die Objekte nach Namen und Typen finden. Sie finden Objekte nach Namen und Typen mithilfe der Methode `IPluginFactory.find()`.

**Suche nach Objekten mit Beziehungen zu anderen Objekten**

Sie können Funktionen erstellen, um Objekte zu finden, die mit einem bestimmten Objekt über einen bestimmten Beziehungstyp verknüpft sind. Beziehungen definieren Sie in der `vso.xml`-Datei. Sie können auch Finder erstellen, die abhängige untergeordnete Objekte finden, deren Beziehung mit allen übergeordneten Objekten durch einen bestimmten Beziehungstyp bestimmt wird. Sie implementieren die Methode `IPluginFactory.findRelation()`, um Objekte zu finden, die mit einem bestimmten übergeordneten Objekt durch einen bestimmten Beziehungstyp verknüpft sind. Sie implementieren die Methode `IPluginFactory.hasChildrenInRelation()`, um zu ermitteln, ob mindestens ein untergeordnetes Objekt für eine übergeordnete Instanz existiert.

**Definieren von Abfragen, um Objekte nach Ihren Kriterien zu finden**

Sie können Objektfinder erstellen, die von Ihnen definierte Abfragerregeln implementieren. Sie implementieren die Methode `IPluginFactory.findAll()`, um alle Objekte zu finden, die bestimmten, von Ihnen definierten Abfragerregeln entsprechen, wenn die Factory diese Methode aufruft. Sie erhalten die Ergebnisse der Methode `findAll()` in einem `QueryResult`-Objekt, das eine Liste aller gefundenen Objekte enthält, die mit den von Ihnen definierten Abfragerregeln übereinstimmen.

Weitere Informationen zur Schnittstelle `IPluginFactory`, zu allen ihren Methoden und allen anderen Klassen der Plug-In-API finden Sie unter „[Orchestrator Plug-In-API-Referenz](#)“, auf Seite 61.

## Rolle von Finder-Objekten

Finder-Objekte erkennen und finden bestimmte Instanzen von verwalteten Objekttypen in der integrierten Technologie. Orchestrator kann Objekte, die in der integrierten Technologie gefunden werden, ändern und mit ihnen interagieren, indem es Workflows für die Finder-Objekte ausführt.

Jede Instanz eines bestimmten verwalteten Objekttyps in der integrierten Technologie muss einen eindeutigen Bezeichner aufweisen, damit Orchestrator-Finder-Objekte sie finden können. Die integrierte Technologie stellt die eindeutigen Bezeichner für die Objektinstanzen als Zeichenfolgen bereit. Wenn ein Workflow ausgeführt wird, legt Orchestrator die eindeutigen Bezeichner der gefundenen Objekte als Workflow-Attributwerte fest. Workflows, die ein Objekt eines bestimmten Typs als Eingabeparameter benötigen, werden mit einer bestimmten Instanz dieses Objekttyps ausgeführt.

Finder-Objekte, die von Plug-Ins zur Orchestrator-JavaScript-API hinzugefügt werden, enthalten den Plug-In-Namen als Präfix. Der verwaltete Objekttyp `VirtualMachine` aus der vCenter Server-API wird beispielsweise in Orchestrator als JavaScript-Typ `VC:VirtualMachine` angezeigt.

Um beispielsweise über das vCenter Server-Plug-In auf eine bestimmte `VC:VirtualMachine`-Instanz zuzugreifen, implementiert Orchestrator ein Finder-Objekt, bei dem das Attribut `id` der virtuellen Maschine als deren eindeutiger Bezeichner verwendet wird. Sie können diese Objektinstanz als Attributwerte an Workflowelemente übergeben.

Ein Orchestrator-Plug-In ordnet die Objekte aus der integrierten Technologie äquivalenten Orchestrator-Finder-Objekten zu, die in den `<finder>`-Elementen in der Datei `vso.xml` enthalten sind. Die `<finder>`-Elemente identifizieren die Methode oder Funktion aus der integrierten Technologie, die den eindeutigen Bezeichner für eine bestimmte Instanz eines Objekts abrufen. Außerdem definieren die `<finder>`-Elemente Beziehungen zwischen Objekten, um Objekte anhand der Art ihrer Beziehungen zu anderen Objekten zu finden.

Finder-Objekte werden in der Orchestrator-Registerkarte **Bestandsliste** unter dem Plug-In angezeigt, das sie enthält.

## Rolle von Skriptobjekten

Skriptobjekte sind JavaScript-Darstellungen von Objekten aus der integrierten Technologie. Skriptobjekte aus Plug-Ins werden in der Orchestrator-JavaScript-API angezeigt. Sie können sie in Elementen mit Skript in Workflows und Aktionen verwenden.

Skriptobjekte aus Plug-Ins werden in der Orchestrator-JavaScript-API als JavaScript-Module, -Typen und -Klassen angezeigt. Die meisten Finder-Objekte verfügen über eine Skriptobjekt-Darstellung. Die JavaScript-Klassen können der Orchestrator-JavaScript-API Methoden und Attribute hinzufügen, die die Methoden und Attribute für Objekte aus der API der integrierten Technologie darstellen. Die integrierte Technologie stellt die Implementierungen der Objekte, Typen, Klassen, Attribute und Methoden unabhängig von Orchestrator bereit. Das vCenter Server-Plug-In stellt beispielsweise alle Objekte aus der vCenter Server-API als JavaScript-Objekte in der Orchestrator-JavaScript-API dar, mit JavaScript-Darstellungen aller durch die vCenter Server-API definierten Klassen, Methoden und Attribute. Sie können die Skripterstellungsklassen von vCenter Server und die durch sie definierten Methoden und Attribute in Orchestrator-Funktionen mit Skript verwenden.

Der verwaltete Objekttyp `VirtualMachine` aus der vCenter Server-API wird beispielsweise vom Finder `VC:VirtualMachine` gefunden und in der Orchestrator-JavaScript-API als JavaScript-Klasse `VcVirtualMachine` angezeigt. Die JavaScript-Klasse `VcVirtualMachine` in der Orchestrator-JavaScript-API definiert genau die gleichen Methoden und Attribute wie das verwaltete Objekt `VirtualMachine` aus der vCenter Server-API.

Ein Orchestrator-Plug-In ordnet die Objekte, Typen, Klassen, Attribute und Methoden aus der integrierten Technologie äquivalenten Orchestrator-JavaScript-Objekten, -Typen, -Klassen, -Attributen und -Methoden zu, die im Element `<scripting-objects>` in der Datei `vso.xml` enthalten sind.

## Rolle von Ereignishandlern

Ereignisse sind Änderungen an Zuständen oder Attributen der Objekte, die Orchestrator in der Plug-In-Technologie findet. Orchestrator überwacht Ereignisse durch Implementieren von Ereignishandlern.

Mithilfe von Orchestrator-Plug-Ins können Sie Ereignisse in einer Plug-In-Technologie auf verschiedene Arten überwachen. Mithilfe der Orchestrator-Plug-In-API können Sie Ereignishandler der folgenden Typen erstellen, um Ereignisse in einer integrierten Technologie zu überwachen.

### Listener

Überwachen Objekte in der integrierten Technologie passiv auf Änderungen an ihrem Zustand. Durch die integrierte Technologie oder die Plug-In-Implementierung werden die Ereignisse definiert, die Listener überwachen. Listener initiieren keine Ereignisse, sondern benachrichtigen Orchestrator, wenn die Ereignisse auftreten. Listener erkennen Ereignisse entweder durch Abfragen der integrierten Technologie oder durch Benachrichtigungen von der integrierten Technologie. Wenn Ereignisse auftreten, können Orchestrator-Richtlinien oder -Workflows, die auf das betreffende Ereignis warten, darauf reagieren, indem sie Vorgänge im Orchestrator-Server starten. Listener-Komponenten sind optional.

### Richtlinien

Überwachen bestimmte Ereignisse in der integrierten Technologie und starten Vorgänge im Orchestrator-Server, wenn die Ereignisse auftreten. Richtlinien können Richtlinienauslöser und Richtlinienkontrollen überwachen. Richtlinienauslöser definieren ein Ereignis in der integrierten Technologie, dessen Auftreten dazu führt, dass eine laufende Richtlinie einen Vorgang im Orchestrator-Server startet, z. B. die Ausführung eines Workflows. Richtlinienkontrollen definieren Wertebereiche für die Attribute eines Objekts in der integrierten Technologie, deren Überschreitung dazu führt, dass Orchestrator einen Vorgang startet. Richtlinien sind optional.

### Workflowauslöser

Wenn ein laufender Workflow ein Warteereignis-Element enthält, wird die Ausführung bei Erreichen dieses Elements angehalten und der Workflow wartet darauf, dass ein Ereignis in der integrierten Technologie auftritt. Workflowauslöser definieren die Ereignisse in der integrierten Technologie, auf die bei Warteereignis-Elementen in Workflows gewartet wird. Sie registrieren Workflowauslöser bei Watchern. Workflowauslöser sind optional.

### Watcher

Überwachen Workflowauslöser im Auftrag eines Warteereignis-Elements in einem Workflow auf ein bestimmtes Ereignis in der integrierten Technologie. Bei Auftreten des Ereignisses benachrichtigen die Watcher alle Workflows, die auf dieses Ereignis warten. Watcher sind optional.

## Inhalt und Struktur von Plug-Ins

Orchestrator-Plug-Ins müssen einen Standardsatz von Komponenten enthalten und eine Standarddateistruktur aufweisen. Damit ein Plug-In der Standarddateistruktur entspricht, muss es bestimmte Ordner und Dateien umfassen.

Zum Erstellen eines Orchestrator-Plug-Ins definieren Sie, wie Orchestrator auf Objekte der entsprechenden Plug-In-Technologie zugreift und mit diesen interagiert. Und Sie ordnen diese Objekte und Funktionen der Plug-In-Technologie in der Datei `vso.xml` den entsprechenden Objekten und Funktionen von Orchestrator zu.

Die Datei `vso.xml` muss Verweise auf alle Objekttypen und Vorgänge enthalten, die in Orchestrator genutzt werden. Jedes Objekt, das vom Plug-In in der Plug-In-Technologie gefunden wird, muss eine eindeutige, von Ihnen zugewiesene Kennung besitzen. Sie definieren die Objektnamen in den `finder`-Elementen und in den Objektelementen in der Datei `vso.xml`.



Ein Plug-In kann als Standard-Java-Archivdatei (JAR) oder als ZIP-Datei bereitgestellt werden, muss jedoch stets unter Verwendung der Dateierweiterung `.dar` umbenannt werden.

---

**HINWEIS** Über das Orchestrator Control Center können Sie DAR-Dateien auf den Orchestrator Server importieren.

---

- [Definieren der Anwendungszuweisung in der Datei vso.xml](#) auf Seite 57  
Objekte in der Datei `vso.xml` erscheinen als Skriptobjekte in der Orchestrator-Skript-API oder als Finder-Objekte auf der Orchestrator-Registerkarte **Bestandsliste**.
- [Format der Plug-In-Definitionsdatei vso.xml](#) auf Seite 58  
Die Datei `vso.xml` definiert, wie der Orchestrator-Server mit der integrierten Technologie interagiert. Sie müssen in die Datei `vso.xml` einen Verweis auf alle Objekttypen und Vorgänge für Orchestrator einschließen.
- [Benennen von Plug-In-Objekten](#) auf Seite 59  
Jedem Objekt, das vom Plug-In in der Plug-In-Technologie gefunden wird, muss von Ihnen eine eindeutige Kennung zugewiesen werden. Sie definieren die Objektnamen in den `<finder>`-Elementen und in den `<object>`-Elementen in der Datei `vso.xml`.
- [Benennungskonventionen für Plug-In-Objekte](#) auf Seite 59  
Sie müssen die Benennungskonventionen der Java-Klasse beachten, wenn Sie Objekte in Plug-Ins benennen.
- [Dateistruktur des Plug-Ins](#) auf Seite 60  
Ein Plug-In muss mit der Standarddateistruktur kompatibel sein und bestimmte spezifische Ordner und Dateien enthalten. Sie stellen ein Plug-In als Standard-Java-Archiv (JAR) oder als ZIP-Datei bereit, die Sie auf die Dateierweiterung `.dar` umbenennen müssen.

## Definieren der Anwendungszuweisung in der Datei vso.xml

Objekte in der Datei `vso.xml` erscheinen als Skriptobjekte in der Orchestrator-Skript-API oder als Finder-Objekte auf der Orchestrator-Registerkarte **Bestandsliste**.

Die `vso.xml`-Datei gibt dem Orchestrator-Server die folgenden Informationen an:

- Version, Name und Beschreibung für das Plug-In
- Verweist auf die Klassen der integrierten Technologie und den verknüpften Plug-In-Adapter
- Initialisiert das Plug-In beim Start des Orchestrator-Servers
- Skripterstellungstypen, die die verschiedenen Objekttypen in der integrierten Technologie darstellen
- Beziehungen zwischen Objekttypen, um zu definieren, wie die Objekte in der Orchestrator-Bestandsliste angezeigt werden
- Skripterstellungsklassen, die die Objekte und Vorgänge in der integrierten Technologie Funktionen und Objekttypen in der Orchestrator-JavaScript-API zuweisen
- Enumerationen, um eine Liste konstanter Werte zu definieren, die für alle Objekte eines bestimmten Typs gelten
- Ereignisse, die Orchestrator in der integrierten Technologie überwacht

Die Datei `vso.xml` muss der XML-Schemadefinition von Orchestrator-Plug-Ins entsprechen. Diese können Sie auf der VMware-Support-Website aufrufen.

<http://www.vmware.com/support/orchestrator/plugin-4-1.xsd>

Beschreibungen aller Elemente der Datei `vso.xml` finden Sie unter „[Elemente der Plug-In-Definitionsdatei vso.xml](#)“, auf Seite 72.

## Format der Plug-In-Definitionsdatei vso.xml

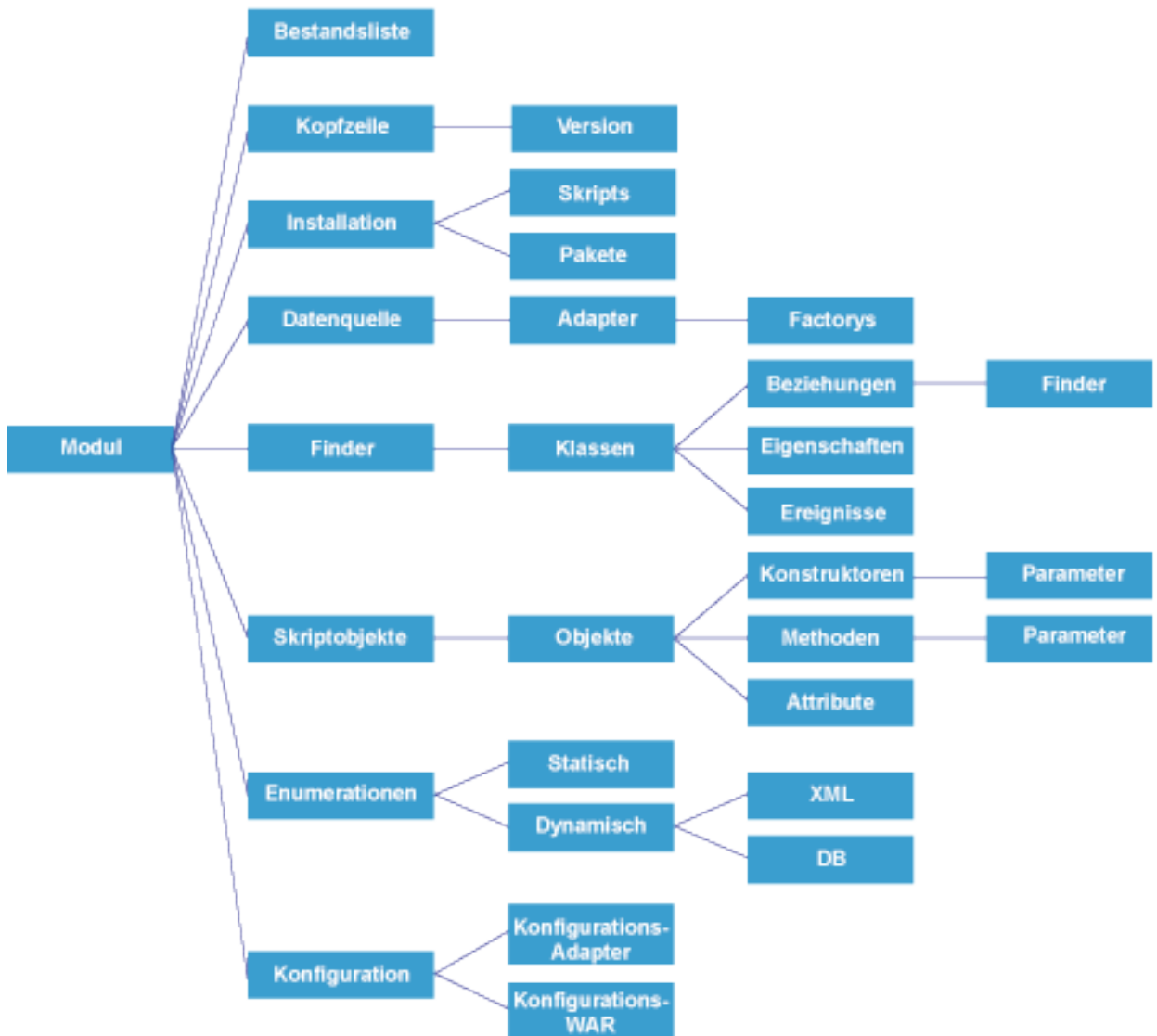
Die Datei vso.xml definiert, wie der Orchestrator-Server mit der integrierten Technologie interagiert. Sie müssen in die Datei vso.xml einen Verweis auf alle Objekttypen und Vorgänge für Orchestrator einschließen.

Objekte in der Datei vso.xml erscheinen als Skriptobjekte in der Orchestrator-Skript-API oder als Finder-Objekte auf der Orchestrator-Registerkarte **Bestandsliste**.

Als Teil der offenen Architektur und standardisierten Implementierung von Plug-Ins muss die Datei vso.xml einem Standardformat entsprechen.

Das folgende Diagramm zeigt das Format der Plug-In-Definitionsdatei vso.xml und wie die Elemente ineinander verschachtelt sind.

**Abbildung 1-4.** Format der Plug-In-Definitionsdatei vso.xml



## Benennen von Plug-In-Objekten

Jedem Objekt, das vom Plug-In in der Plug-In-Technologie gefunden wird, muss von Ihnen eine eindeutige Kennung zugewiesen werden. Sie definieren die Objektnamen in den <finder>-Elementen und in den <object>-Elementen in der Datei vso.xml.

Die Finder-Vorgänge, die Sie in der Factory-Implementierung definieren, finden Objekte in der Plug-In-Technologie. Wenn das Plug-In Objekte findet, können Sie sie in Orchestrator-Workflows verwenden und von einem Workflowelement zu einem anderen weiterreichen. Die eindeutigen Bezeichner, die Sie für die Objekte bereitstellen, ermöglichen es Ihnen, zu den Elementen in einem Workflow zu wandern.

Der Orchestrator-Server speichert nur den Typ und den Bezeichner jedes von ihm verarbeiteten Objekts. Informationen über die Herkunft oder die Art, wie Orchestrator das Objekt erhalten hat, werden nicht gespeichert. Sie müssen Objekte in der Plug-In-Implementierung konsistent benennen, damit Sie die Objekte verfolgen können, die Sie aus den Plug-Ins erhalten.

Wenn der Orchestrator-Server stoppt, während Workflows ausgeführt werden, werden die Workflows beim Neustart des Servers an dem Workflowelement wieder aufgenommen, das beim Stoppen des Servers aktiv war. Der Workflow nutzt die Bezeichner, um Objekte abzurufen, die das Element verarbeitet hat, als der Server gestoppt wurde.

## Benennungskonventionen für Plug-In-Objekte

Sie müssen die Benennungskonventionen der Java-Klasse beachten, wenn Sie Objekte in Plug-Ins benennen.

---

**WICHTIG** Aufgrund der Art und Weise, wie die Workflow-Engine die Daten serialisiert, dürfen Sie nicht die folgenden Zeichenfolgen in Objektnamen verwenden. Das Verwenden dieser Zeichenfolgen in Objektnamen führt dazu, dass die Workflow-Engine Workflows falsch analysiert, wodurch unerwartetes Verhalten bei der Workflowausführung auftreten kann.

- # ; #
  - # , #
  - # = #
- 

Verwenden Sie diese Richtlinien, wenn Sie Objekte in Plug-Ins benennen.

- Verwenden Sie einen Anfangsgroßbuchstaben für jedes Wort im Namen.
- Verwenden Sie keine Leerzeichen zum Trennen von Wörtern.
- Verwenden Sie bei Buchstaben nur die Standardzeichen A bis Z und a bis z.
- Verwenden Sie keine Sonderzeichen wie Akzente.
- Verwenden Sie keine Zahl als erstes Zeichen eines Namen.
- Verwenden Sie nach Möglichkeit weniger als 10 Zeichen.

In [Tabelle 1-5](#) sind die Regeln aufgeführt, die für die einzelnen Objekttypen gelten.

**Tabelle 1-5.** Benennungsregeln für Plug-In-Objekte

Objekttyp	Benennungsregeln
Plug-In	<ul style="list-style-type: none"> <li>■ Definiert im Element <code>&lt;module&gt;</code> in der Datei <code>vso.xml</code>.</li> <li>■ Muss den Java-Klassen-Benennungskonventionen entsprechen.</li> <li>■ Muss eindeutig sein. Sie können nicht zwei Plug-Ins mit demselben Namen im Orchestrator-Server ausführen.</li> </ul>
Finder-Objekt	<ul style="list-style-type: none"> <li>■ Definiert in den Elementen <code>&lt;finder&gt;</code> in der Datei <code>vso.xml</code>.</li> <li>■ Muss den Java-Klassen-Benennungskonventionen entsprechen.</li> <li>■ Muss im Plug-In eindeutig sein.</li> </ul> <p>Orchestrator fügt Finder-Objektnamen in den Finder-Objekttypen in der Orchestrator-Skript-API den Plug-In-Namen und einen Doppelpunkt hinzu. Beispiel: Der Objekttyp <code>VirtualMachine</code> aus dem vCenter Server-Plug-In wird in der Orchestrator-Skript-API als <code>VC:VirtualMachine</code> angezeigt.</p>
Skriptobjekt	<ul style="list-style-type: none"> <li>■ Definiert in den Elementen <code>&lt;scripting-object&gt;</code> in der Datei <code>vso.xml</code>.</li> <li>■ Muss den Java-Klassen-Benennungskonventionen entsprechen.</li> <li>■ Muss im Orchestrator-Server eindeutig sein.</li> <li>■ Fügen Sie dem Skriptobjektnamen immer ein Präfix mit dem Namen des Plug-Ins hinzu, um eine Verwechslung von Skriptobjekten mit Finder-Objekten mit demselben Namen oder mit Skriptobjekten aus anderen Plug-Ins zu vermeiden. Fügen Sie aber keinen Doppelpunkt hinzu. Beispiel: Die Klasse <code>VirtualMachine</code> aus dem vCenter Server-Plug-In wird in der Orchestrator-Skript-API als Klasse <code>VcVirtualMachine</code> angezeigt.</li> </ul>

## Dateistruktur des Plug-Ins

Ein Plug-In muss mit der Standarddateistruktur kompatibel sein und bestimmte spezifische Ordner und Dateien enthalten. Sie stellen ein Plug-In als Standard-Java-Archiv (JAR) oder als ZIP-Datei bereit, die Sie auf die Dateierweiterung `.dar` umbenennen müssen.

Die Inhalte des DAR-Archivs müssen die folgende Ordnerstruktur und die folgenden Namenskonventionen verwenden.

**Tabelle 1-6.** Struktur des DAR-Archivs

Ordner	Beschreibung
<code>plug-in_name\VS0-INF\</code>	Enthält die <code>vso.xml</code> -Datei, die die Zuordnung der Objekte in der Plug-In-Technologie zu Orchestrator-Objekten definiert. Der <code>VS0-INF</code> -Ordner und die <code>vso.xml</code> -Datei sind vorgeschrieben.
<code>plug-in_name\lib\</code>	Enthält die JAR-Dateien, in denen die Binärdateien der Plug-In-Technologie zusammengefasst sind. Der Ordner enthält auch JAR-Dateien, die die Implementierungen von Adapter, Factory, Benachrichtigungshandles und anderen Schnittstellen im Plug-In umfassen. Der <code>lib</code> -Ordner und die JAR-Dateien sind vorgeschrieben.
<code>plug-in_name\resources\</code>	Enthält Ressourcendateien, die das Plug-In erfordert. Der Ordner <code>resources</code> kann folgende Elementtypen enthalten: <ul style="list-style-type: none"> <li>■ Bilddateien zur Darstellung der Objekte des Plug-Ins auf der Registerkarte <b>Bestand</b> in Orchestrator</li> <li>■ Skripte zur Definition des Initialisierungsverhaltens beim Start des Plug-Ins</li> <li>■ Orchestrator-Pakete, die benutzerdefinierte Workflows, Aktionen und andere Ressourcen enthalten können, die mit den Objekten interagieren, auf die Sie über die Plug-Ins zugreifen</li> </ul> Sie können Ressourcen in Unterordnern organisieren. Beispiele dafür: <code>resources\images\</code> , <code>resources\scripts\</code> oder <code>resources\packages\</code> . Der Ordner <code>resources</code> ist optional.

Über das Orchestrator Control Center können Sie DAR-Dateien auf den Orchestrator Server importieren.

## Orchestrator Plug-In-API-Referenz

Die Orchestrator-Plug-In-API definiert Java-Schnittstellen und -Klassen, um sie zu implementieren und zu erweitern, wenn Sie die `IPluginAdaptor`- und `IPluginFactory`-Implementierungen entwickeln, um ein Plug-In zu erstellen.

Alle Klassen sind im Paket `ch.dunes.vso.sdk.api` enthalten, sofern nicht etwas anderes festgelegt ist.

### IAop-Schnittstelle

Die `IAop`-Schnittstelle bietet Methoden zum Abrufen und Einrichten von Eigenschaften von Objekten in Plug-In-Anwendungen.

```
public interface IAop
```

Die `IAop`-Schnittstelle definiert die folgenden Methoden:

Methode	Gibt Folgendes zurück	Beschreibung
<code>get(java.lang.String propertyName, java.lang.Object object, java.lang.Object sdkObject)</code>	<code>java.lang.Object</code>	Ruft eine Eigenschaft eines bestimmten Objekts des Plug-Ins ab.
<code>set(java.lang.String propertyName, java.lang.String propertyValue, java.lang.Object object)</code>	<code>Void</code>	Richtet eine Eigenschaft für ein bestimmtes Objekt des Plug-Ins ein.

## IDynamicFinder-Schnittstelle

Die IDynamicFinder-Schnittstelle gibt die ID und die Eigenschaften eines Finders programmgesteuert zurück, anstatt die ID und die Eigenschaften in der Datei `vso.xml` zu definieren.

Die IDynamicFinder-Schnittstelle definiert die folgenden Methoden.

Methode	Gibt Folgendes zurück	Beschreibung
<code>getIdAccessor(java.lang.String type)</code>	<code>java.lang.String</code>	Stellt einen OGNL-Ausdruck bereit, um programmgesteuert eine Objekt-ID zu erhalten.
<code>getProperties(java.lang.String type)</code>	<code>java.util.List&lt;SDKFinderProperty&gt;</code>	Stellt programmgesteuert eine Liste von Objekteigenschaften bereit.

## IPluginAdaptor-Schnittstelle

Sie implementieren die Schnittstelle IPluginAdaptor, um Plug-In-Factorys, Ereignisse und Watcher zu verwalten. Die Schnittstelle IPluginAdaptor definiert einen Adapter zwischen einem Plug-In und dem Orchestrator-Server.

IPluginAdaptor-Instanzen sind für die Sitzungsverwaltung zuständig. Die Schnittstelle IPluginAdaptor definiert die folgenden Methoden.

Methode	Gibt Folgendes zurück	Beschreibung
<code>addWatcher(PluginWatcher watcher)</code>	<code>Void</code>	Fügt einen Watcher zur Überwachung eines bestimmten Ereignisses hinzu
<code>createPluginFactory(java.lang.String sessionId, java.lang.String username, java.lang.String password, IPluginNotificationHandler notificationHandler)</code>	<code>IPluginFactory</code>	Erstellt eine IPluginFactory-Instanz. Der Orchestrator-Server verwendet die Factory, um Objekte von der durch Plug-In-Technologie über ihre ID, über ihre Beziehung zu anderen Objekten usw. anzufordern.  Die Sitzungs-ID ermöglicht es Ihnen, eine laufende Sitzung zu identifizieren. Beispiel: Ein Benutzer kann sich bei zwei verschiedenen Orchestrator-Clients anmelden und gleichzeitig zwei Sitzungen ausführen.  Durch den Start eines Workflows wird eine Sitzung erstellt, die von dem Client unabhängig ist, auf dem der Workflow gestartet wurde. Ein Workflow wird weiter ausgeführt, auch wenn Sie den Orchestrator-Client schließen.
<code>installLicenses(PluginLicense[] licenses)</code>	<code>Void</code>	Installiert die Lizenzinformationen für Standard-Plug-Ins, die von VMware bereitgestellt werden
<code>registerEventPublisher(java.lang.String type, java.lang.String id, IPluginEventPublisher publisher)</code>	<code>Void</code>	Setzt Auslöser und Kontrollen für ein Element im Bestand
<code>removeWatcher(java.lang.String watcherId)</code>	<code>Void</code>	Entfernt einen Watcher
<code>setPluginName(java.lang.String pluginName)</code>	<code>Void</code>	Ruft den Plug-In-Namen aus der <code>vso.xml</code> -Datei ab
<code>setPluginPublisher(IPluginPublisher pluginPublisher)</code>	<code>Void</code>	Legt den Herausgeber des Plug-Ins fest

Methode	Gibt Folgendes zurück	Beschreibung
<code>uninstallPluginFactory(IPluginFactory plugin)</code>	Void	Deinstalliert eine Plug-In-Factory
<code>unregisterEventPublisher(java.lang.String type, java.lang.String id, IPluginEventPublisher publisher)</code>	Void	Entfernt Auslöser und Kontrollen von einem Element im Bestand

## IPluginEventPublisher-Schnittstelle

Die Schnittstelle `IPluginEventPublisher` veröffentlicht Kontrollen und Auslöser auf einem Ereignisbenachrichtigungsbuss zur Überwachung von Orchestrator-Richtlinien.

Sie können `IPluginEventPublisher`-Instanzen direkt in der Plug-In-Adapterimplementierung oder in getrennten Klassen zur Ereignisgenerierung erstellen.

Sie können die `IPluginEventPublisher`-Schnittstelle verwenden, um Ereignisse in der integrierten Technologie für die Orchestrator-Richtlinien-Engine zu veröffentlichen. Sie erstellen Methoden, um Richtlinienauslöser und -kontrollen für Objekte in integrierter Technologie und Ereignislistener festzulegen, um Ereignisse in diesen Objekten zu überwachen.

Richtlinien können entweder Anzeigen oder Auslöser implementieren, um Objekte in der integrierten Technologie zu überwachen. Richtlinienkontrollen überwachen die Attribute von Objekten und übertragen mithilfe von Push ein Ereignis auf den Orchestrator-Server, wenn die Werte der Objekte bestimmte Grenzwerte überschreiten. Richtlinienauslöser überwachen Objekte und übertragen mithilfe von Push ein Ereignis auf den Orchestrator-Server, wenn ein definiertes Ereignis für das Objekt eintritt. Sie registrieren Richtlinienkontrollen und -auslöser bei `IPluginEventPublisher`-Instanzen, damit Orchestrator-Richtlinien diese überwachen können.

Die Schnittstelle `IPluginEventPublisher` definiert die folgenden Methoden.

Type	Gibt Folgendes zurück	Beschreibung
<code>pushGauge(java.lang.String type, java.lang.String id, java.lang.String gaugeName, java.lang.String deviceName, java.lang.Double gaugeValue)</code>	Void	Veröffentlicht eine Kontrolle für zu überwachende Richtlinien. Übernimmt die folgenden Parameter: <ul style="list-style-type: none"> <li>■ <code>type</code>: Typ des zu überwachenden Objekts</li> <li>■ <code>id</code>: Bezeichner des zu überwachenden Objekts</li> <li>■ <code>gaugeName</code>: Name für diese Kontrolle</li> <li>■ <code>deviceName</code>: Name für den Attributtyp, der von der Kontrolle überwacht wird</li> <li>■ <code>gaugeValue</code>: Wert, auf den die Kontrolle das Objekt überwacht</li> </ul>
<code>pushTrigger(java.lang.String type, java.lang.String id, java.lang.String triggerName, java.util.Properties additionalProperties)</code>	Void	Veröffentlicht einen Auslöser für zu überwachende Richtlinien. Übernimmt die folgenden Parameter: <ul style="list-style-type: none"> <li>■ <code>type</code>: Typ des zu überwachenden Objekts</li> <li>■ <code>id</code>: Bezeichner des zu überwachenden Objekts</li> <li>■ <code>triggerName</code>: Name für diesen Auslöser</li> <li>■ <code>additionalProperties</code>: Gegebenenfalls zusätzliche Eigenschaften für den Auslöser zur Überwachung</li> </ul>

## IPluginFactory-Schnittstelle

IPluginAdaptor gibt IPluginFactory-Instanzen zurück. IPluginFactory-Instanzen führen Befehle in der als Plug-In integrierten Anwendung aus und finden Objekte, mit denen Orchestrator-Vorgänge ausgeführt werden sollen.

Die Schnittstelle IPluginFactory definiert das folgende Feld:

```
static final java.lang.String RELATION_CHILDREN
```

Die Schnittstelle IPluginFactory definiert die folgenden Methoden.

Methode	Gibt Folgendes zurück	Beschreibung
<code>executePluginCommand(java.lang.String cmd)</code>	Void	Plug-In verwenden, um einen Befehl auszuführen. VMware empfiehlt, diese Methode nicht zu verwenden.
<code>find(java.lang.String type, java.lang.String id)</code>	java.lang.Object	Plug-In verwenden, um ein Objekt zu finden. Das Objekt wird nach ID und Typ identifiziert.
<code>findAll(java.lang.String type, java.lang.String query)</code>	QueryResult	Plug-In verwenden, um Objekte eines bestimmten Typs zu finden, die mit einer Abfragezeichenfolge übereinstimmen. Sie definieren die Syntax der Abfrage in der IPluginFactory-Implementierung des Plug-Ins. Wenn Sie die Abfragesyntax nicht definieren, gibt <code>findAll()</code> alle Objekte des angegebenen Typs zurück.



Methode	Gibt Folgendes zurück	Beschreibung
findRelation(java.lang.String parentType, java.lang.String parentId, java.lang.String relationName)	java.util.List	Ermittelt, ob ein Objekt untergeordnete Objekte hat.
hasChildrenInRelation(java.lang.String parentType, java.lang.String parentId, java.lang.String relationName)	HasChildrenResult	Findet alle untergeordneten Objekte, die eine bestimmte Beziehung mit einem bestimmten übergeordneten Objekt haben.
invalidate(java.lang.String type, java.lang.String id)	Void	Objekte nach Typ und ID als nicht gültig ermitteln.
void invalidateAll()	Void	Alle Objekte im Cache als nicht gültig ermitteln.

## IPluginNotificationHandler-Schnittstelle

IPluginNotificationHandler definiert Methoden, um Orchestrator über verschiedene Arten von Ereignissen zu benachrichtigen, die bei den Objekten vorkommen, auf die Orchestrator über das Plug-In zugreift.

Die Schnittstelle IPluginNotificationHandler definiert die folgenden Methoden.

Methode	Gibt Folgendes zurück	Beschreibung
getSessionID()	java.lang.String	Gibt die aktuelle Sitzungs-ID zurück.
notifyElementDeleted(java.lang.String type, java.lang.String id)	Void	Benachrichtigt das System, dass ein Objekt mit dem angegebenen Typ und der angegebenen ID gelöscht wurde
notifyElementInvalidate(java.lang.String type, java.lang.String id)	Void	Benachrichtigt das System, dass sich die Beziehungen eines Objekts geändert haben. Sie können die notifyElementInvalidate()-Methode verwenden, um Orchestrator über alle Veränderungen in Beziehungen zwischen Objekten zu benachrichtigen, nicht nur über Änderungen von Beziehungen, die ein Objekt ungültig machen. Das Hinzufügen eines untergeordneten Objekts zu einem übergeordneten Objekt stellt beispielsweise eine Änderung in der Beziehung zwischen den beiden Objekten dar.
notifyElementUpdated(java.lang.String type, java.lang.String id)	Void	Benachrichtigt das System, dass die Attribute eines Objekts geändert wurden
notifyMessage(ch.dunnes.vso.sdk.api.ErrorLevel severity, java.lang.String type, java.lang.String id, java.lang.String message)	Void	Veröffentlicht eine Fehlermeldung in Bezug auf das aktuelle Modul

## IPluginPublisher-Schnittstelle

Die Schnittstelle `IPluginPublisher` veröffentlicht ein `Watcher`-Ereignis auf einem Ereignisbenachrichtigungsbus für „Warteereignis“-Elemente bei länger ausführenden Workflows, die überwacht werden sollen.

Wenn ein Workflowauslöser ein Ereignis in der integrierten Technologie startet, benachrichtigt ein Plug-In-Watcher, der diesen Auslöser überwacht und der bei einer `IPluginPublisher`-Instanz registriert ist, alle wartenden Workflows darüber, dass das Ereignis eingetreten ist.

Die Schnittstelle `IPluginPublisher` definiert die folgende Methode.

Typ	Wert	Beschreibung
<code>pushWatcherEvent(java.lang.String id, java.util.Properties properties)</code>	<code>Void</code>	Veröffentlicht ein <code>Watcher</code> -Ereignis auf dem Ereignisbenachrichtigungsbus

## WebConfigurationAdaptor-Schnittstelle

Die `WebConfigurationAdaptor`-Schnittstelle implementiert `IConfigurationAdaptor` und definiert Methoden zum Finden und Installieren einer Webanwendung auf der Konfigurationsregisterkarte für ein Plug-In.

**HINWEIS** Die `WebConfigurationAdaptor`-Schnittstelle ist seit Orchestrator 4.1 veraltet. Um der Konfiguration eine Webanwendung hinzuzufügen, implementieren Sie `IConfigurationAdaptor` und verwenden Sie das Attribut `configuration-war` in der Datei `vso.xml` zum Identifizieren der Webanwendung.

Die `WebConfigurationAdaptor`-Schnittstelle definiert die folgenden Methoden.

Methode	Gibt Folgendes zurück	Beschreibung
<code>getWebApplicationContext()</code>	<code>String</code>	Findet die WAR-Datei der Webanwendung für die Konfigurationsregisterkarte. Geben Sie den Namen und Pfad zur WAR-Datei aus dem <code>/webapps</code> -Verzeichnis in der DAR-Datei als Zeichenfolge ein.
<code>setWebConfiguration(boolean webConfiguration)</code>	<code>Boolean</code>	Legen Sie fest, ob die Inhalte der Konfigurationsregisterkarte durch eine Webanwendung definiert werden.

## PluginTrigger-Klasse

Die Klasse `PluginTrigger` erstellt ein Auslösermodul, das Informationen über Objekte und Ereignisse erhält, die in der Plug-In-Technologie für ein „Warteereignis“-Element in einem Workflow zu überwachen sind.

Die `PluginTrigger`-Klasse definiert Methoden zum Abrufen oder Festlegen des Typs und Namens des zu überwachenden Objekts, der Art des Ereignisses und einer Zeitüberschreitungsdauer.

Sie erstellen Implementierungen der `PluginTrigger`-Klasse für die ausschließliche Verwendung von Warteereigniselementen in Workflows. Sie definieren Richtlinienauslöser für Orchestrator-Richtlinien in Klassen, die Ereignisse definieren und die `IPluginEventPublisher.pushTrigger()`-Methode implementieren.

```
public class PluginTrigger
extends java.lang.Object
implements java.io.Serializable
```

Die Klasse `PluginTrigger` definiert die folgenden Methoden.

Methode	Gibt Folgendes zurück	Beschreibung
<code>getModuleName()</code>	<code>java.lang.String</code>	Bezieht den Namen des Auslösermoduls.
<code>getProperties()</code>	<code>java.util.Properties</code>	Bezieht eine Liste von Eigenschaften für den Auslöser.
<code>getSdkId()</code>	<code>java.lang.String</code>	Bezieht den Bezeichner des Objekts, das in der Plug-In-Technologie überwacht werden soll.
<code>getSdkType()</code>	<code>java.lang.String</code>	Bezieht den Typ des Objekts, das in der Plug-In-Technologie überwacht werden soll.
<code>getTimeout()</code>	<code>Long</code>	Bezieht die Zeitüberschreitungsperiode für den Auslöser.
<code>setModuleName(java.lang.String moduleName)</code>	<code>Void</code>	Legt den Namen des Triggermoduls fest
<code>setProperties(java.util.Properties properties)</code>	<code>Void</code>	Legt eine Liste von Eigenschaften für den Auslöser fest.
<code>setSdkId(java.lang.String sdkId)</code>	<code>Void</code>	Legt den Bezeichner des Objekts fest, das in der Plug-In-Technologie überwacht werden soll.
<code>setSdkType(java.lang.String sdkType)</code>	<code>Void</code>	Legt den Typ des Objekts fest, das in der Plug-In-Technologie überwacht werden soll.
<code>setTimeout(long timeout)</code>	<code>Void</code>	Legt eine Zeitüberschreitungsperiode in Sekunden fest. Ein negativer Wert deaktiviert die Zeitüberschreitung.

## Konstrukturen

- `PluginTrigger()`
- `PluginTrigger(java.lang.String moduleName, long timeout, java.lang.String sdkType, java.lang.String sdkId)`

## PluginWatcher-Klasse

Die Klasse `PluginWatcher` überwacht ein Auslösermodul auf ein definiertes Ereignis in der Plug-In-Technologie für ein Warteereignis-Element eines Workflows mit langer Ausführungszeit.

Die `PluginWatcher`-Klasse definiert einen Konstruktor, den Sie verwenden können, um Plug-In-Watcher-Instanzen zu erstellen. Die `PluginWatcher`-Klasse definiert Methoden zum Abrufen oder Festlegen von Namen des zu überwachenden Workflowauslösers sowie eine Zeitüberschreitung.

```
public class PluginWatcher
extends java.lang.Object
implements java.io.Serializable
```

Die Klasse `PluginWatcher` definiert die folgenden Methoden:

Methode	Gibt Folgendes zurück	Beschreibung
<code>getId()</code>	<code>java.lang.String</code>	Bezieht die ID des Auslösers
<code>getModuleName()</code>	<code>java.lang.String</code>	Bezieht den Namen des Auslösermoduls.
<code>getTimeoutDate()</code>	<code>Long</code>	Bezieht das Zeitüberschreitungsdatum für den Auslöser

Methoden	Gibt Folgendes zurück	Beschreibung
<code>getTrigger()</code>	Void	Bezieht einen Auslöser
<code>setId(java.lang.String id)</code>	Void	Legt den Bezeichner des Auslösers fest
<code>setTimeoutDate()</code>	Void	Legt das Zeitüberschreitungsdatum für den Auslöser fest

### Konstruktor

`PluginWatcher(PluginTrigger trigger)`

### QueryResult-Klasse

Die Klasse `QueryResult` enthält die Ergebnisse einer `find`-Abfrage auf Objekten, auf die Orchestrator über das Plug-In zugreift.

```
public class QueryResult
extends java.lang.Object
implements java.io.Serializable
```

Der Wert `totalCount` kann größer als die Anzahl von Elementen sein, die von `QueryResult` zurückgegeben werden, wenn die Gesamtanzahl der gefundenen Ergebnisse die Anzahl von Ergebnissen überschreitet, die die Abfrage zurückgibt. Die Anzahl von Ergebnissen, die von der Abfrage zurückgegeben werden, wird in der Abfragesyntax der `vso.xml`-Datei definiert.

Die Klasse `QueryResult` definiert die folgenden Methoden:

Methoden	Gibt Folgendes zurück	Beschreibung
<code>addElement(java.lang.Object element)</code>	Void	Fügt <code>QueryResult</code> ein Element hinzu
<code>addElements(java.util.List elements)</code>	Void	Fügt <code>QueryResult</code> eine Elementliste hinzu
<code>getElements()</code>	<code>java.util.List</code>	Bezieht Elemente aus der Plug-In-Anwendung
<code>getTotalCount()</code>	Lang	Bezieht eine Zählung aller in der Plug-In-Technologie verfügbaren Elemente
<code>isPartialResult()</code>	Boolean	Ermittelt, ob das bezogene Ergebnis vollständig ist
<code>removeElement(java.lang.Object element)</code>	Void	Entfernt ein Element aus der Plug-In-Technologie
<code>setElements(java.util.List elements)</code>	Void	Setzt Elemente in der Plug-In-Technologie
<code>setTotalCount(long totalCount)</code>	Void	Setzt die Gesamtanzahl von verfügbaren Elementen in der Plug-In-Technologie

### Konstruktoren

- `QueryResult()`
- `QueryResult(java.util.List ret)`
- `QueryResult(java.util.List elements, long totalCount)`

## SDKFinderProperty-Klasse

Die Klasse SDKFinderProperty definiert Methoden, um Eigenschaften in den Objekten zu beziehen und festzulegen, die in der Plug-In-Technologie von Orchestrator-Finderobjekten gefunden werden. Die Methode IDynamicFinder.getProperties gibt SDKFinderProperty-Objekte zurück.

```
public class SDKFinderProperty
extends java.lang.Object
```

Die Klasse SDKFinderProperty definiert die folgenden Methoden:

Methoden	Gibt Folgendes zurück	Beschreibung
getAttributeName()	java.lang.String	Bezieht einen Objektattributnamen
getBeanProperty()	java.lang.String	Bezieht Eigenschaften aus einer Java Bean
getDescription()	java.lang.String	Bezieht eine Objektbeschreibung
getDisplayName()	java.lang.String	Bezieht einen Objektanzeigenamen
getPossibleResultType()	java.lang.String	Bezieht mögliche Typen von Ergebnissen, die ein Finder zurückgibt
getPropertyAccessor()	java.lang.String	Bezieht eine Zugriffsroutine auf Objekteigenschaften
getPropertyAccessorTree()	java.lang.Object	Bezieht einen Routinenbaum für den Zugriff auf Objekteigenschaften
isHidden()	Boolean	Zeigt oder verbirgt das Objekt
isShowInColumn()	Boolean	Zeigt oder verbirgt das Objekt in der Datenbankspalte
isShowInDescription()	Boolean	Zeigt oder verbirgt die Objektbeschreibung
setAttributeName(java.lang.String attributeName)	Void	Setzt einen Objektattributnamen
setBeanProperty(java.lang.String beanProperty)	Void	Setzt Eigenschaften in einer Java Bean
setDescription(java.lang.String description)	Void	Setzt eine Objektbeschreibung
setDisplayName(java.lang.String displayName)	Void	Ersetzt einen Objektanzeigenamen
setHidden(boolean hidden)	Void	Zeigt oder verbirgt das Objekt
setPossibleResultType(java.lang.String possibleResultType)	Void	Setzt mögliche Typen von Ergebnissen, die ein Finder zurückgibt
setPropertyAccessor(java.lang.String propertyAccessor)	Void	Setzt eine Zugriffsroutine auf Objekteigenschaften
setPropertyAccessorTree(java.lang.Object propertyAccessorTree)	Void	Setzt einen Routinenbaum für den Zugriff auf Objekteigenschaften
setShowInColumn(boolean showInTable)	Void	Zeigt oder verbirgt das Objekt in der Datenbankspalte
setShowInDescription(boolean showInDescription)	Void	Zeigt oder verbirgt die Objektbeschreibung

**Konstruktor**

```
SDKFinderProperty(java.lang.String attributeName, java.lang.String displayName, java.lang.String
beanProperty, java.lang.String propertyAccessor)
```

**PluginExecutionException-Klasse**

Die Klasse `PluginExecutionException` gibt eine Fehlermeldung zurück, wenn das Plug-In beim Ausführen eines Vorgangs auf eine Ausnahme trifft.

```
public class PluginExecutionException
extends java.lang.Exception
implements java.io.Serializable
```

Die Klasse `PluginExecutionException` erbt die folgenden Methoden von class `java.lang.Throwable`:

```
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace,
printStackTrace, printStackTrace, setStackTrace, toString, fillInStackTrace, getCause, getLocalizedMessage,
sage, getMessage, getStackTrace, initCause, printStackTrace
```

**Konstruktor**

```
PluginExecutionException(java.lang.String message)
```

**PluginOperationException-Klasse**

Die Klasse `PluginOperationException` verarbeitet Fehler, die bei einem Plug-In-Vorgang aufgetreten sind.

```
public class PluginOperationException
extends java.lang.RuntimeException
implements java.io.Serializable
```

Die Klasse `PluginOperationException` erbt die folgenden Methoden von class `java.lang.Throwable`:

```
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace,
printStackTrace, printStackTrace, setStackTrace, toString
```

**Konstruktor**

```
PluginOperationException(java.lang.String message)
```

**Enumeration „HasChildrenResult“**

Die Enumeration `HasChildrenResult` gibt an, ob ein übergeordnetes Element untergeordnete Elemente hat. Die Methode `IPluginFactory.hasChildrenInRelation` gibt `HasChildrenResult`-Objekte zurück.

```
public enum HasChildrenResult
extends java.lang.Enum<HasChildrenResult>
implements java.io.Serializable
```

Die Enumeration `HasChildrenResult` definiert die folgenden Konstanten:

- `public static final HasChildrenResult Yes`
- `public static final HasChildrenResult No`
- `public static final HasChildrenResult Unknown`

Die Enumeration `HasChildrenResult` definiert die folgenden Methoden:

Methode	Gibt Folgendes zurück	Beschreibung
<code>getValue()</code>	<code>int</code>	Gibt einen der folgenden Werte zurück:  <b>1</b> Übergeordnetes Element hat untergeordnete Elemente  <b>-1</b> Übergeordnetes Element hat keine untergeordneten Elemente  <b>0</b> Unbekannter oder ungültiger Parameter
<code>valueOf(java.lang.String name)</code>	<code>static HasChildrenResult</code>	Gibt einen Enumerationskonstante dieses Typs mit dem angegebenen Namen zurück. Die Zeichenfolge muss genau mit einem Bezeichner übereinstimmen, der zum Deklarieren einer Enumerationskonstante dieses Typs verwendet wird. Verwenden Sie keine Leerzeichen im Enumerationsnamen.
<code>values()</code>	<code>static HasChildrenResult[]</code>	Gibt einen Array mit den Konstanten dieses Enumerationstyps in der Reihenfolge der Deklaration zurück. Diese Methode kann Konstanten iterativ durchlaufen:  <pre>for (HasChildrenResult c : HasChildrenResult.values()) System.out.println(c);</pre>

Die Enumeration `HasChildrenResult` erbt die folgenden Methoden von class `java.lang.Enum`:

`clone`, `compareTo`, `equals`, `finalize`, `getDeclaringClass`, `hashCode`, `name`, `ordinal`, `toString`, `valueOf`

## ScriptingAttribute-Anmerkungstyp

Der Anmerkungstyp `ScriptingAttribute` versieht ein Attribut von einem Objekt in der Plug-In-Technologie mit Anmerkungen, um es bei der Skripterstellung als Eigenschaft wenden zu können.

```
@Retention(value=RUNTIME)
@Target(value={METHOD, FIELD})
public @interface ScriptingAttribute
```

Der Anmerkungstyp `ScriptingAttribute` hat folgenden Wert:

```
public abstract java.lang.String value
```

## ScriptingFunction-Anmerkungstyp

Der Anmerkungstyp `ScriptingFunction` versieht eine Methode mit einer Anmerkung, die als Eigenschaft bei der Skripterstellung verwendet wird.

```
@Retention(value=RUNTIME)
@Target(value={METHOD, CONSTRUCTOR})
public @interface ScriptingFunction
```

Der Anmerkungstyp `ScriptingFunction` hat folgenden Wert:

```
public abstract java.lang.String value
```

## ScriptingParameter-Anmerkungstyp

Der Anmerkungstyp `ScriptingParameter` versieht einen Parameter mit einer Anmerkung, die als Eigenschaft bei der Skripterstellung verwendet wird.

```
@Retention(value=RUNTIME)
@Target(value=PARAMETER)
public @interface ScriptingParameter
```

Der Anmerkungstyp `ScriptingParameter` hat folgenden Wert:

```
public abstract java.lang.String value
```

## Elemente der Plug-In-Definitionsdatei vso.xml

Die Datei `vso.xml` umfasst einen Satz an Standardelementen. Einige Elemente sind obligatorisch, andere optional. Jedes Element hat Attribute, die Werte für die Objekte und Vorgänge definieren, die Sie Orchestrator-Objekten und -Vorgängen zuordnen.

Außerdem können Elemente null oder mehr untergeordnete Elemente haben. Eine untergeordnete Element definiert das übergeordnete Element noch weiter. Das gleiche untergeordnete Element kann in mehreren übergeordneten Elementen auftreten. Das Element `description` hat beispielsweise keine untergeordneten Elemente, wird aber als untergeordnetes Element für viele übergeordneten Elemente angezeigt: `module`, `example`, `trigger`, `gauge`, `finder`, `constructor`, `method`, `object` und `enumeration`.

Jede der folgenden Elementdefinitionen listet seine über- und untergeordneten Attribute auf.

### Modulelement

Ein Modul beschreibt eine Gruppe von Plug-In-Objekten, die für Orchestrator verfügbar gemacht werden.

Das Modul enthält Informationen darüber, wie Daten aus der Plug-In-Technologie den Java-Klassen zugeordnet sind, über die Versionierung, die Bereitstellung des Moduls und die Art, wie das Plug-In im Orchestrator-Bestand erscheint.

Das Element `<module>` ist optional. Das Element `<module>` besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
<code>name</code>	Zeichenfolge	Definiert den Typ aller <code>&lt;finder&gt;</code> -Elemente im Plug-In. Erforderliches Attribut.
<code>version</code>	Zahl	Die Plug-In-Versionsnummer, die beim erneuten Laden von Paketen in einer neuen Version des Plug-Ins verwendet wird. Erforderliches Attribut.
<code>build-number</code>	Zahl	Die Plug-In-Build-Nummer, die beim erneuten Laden von Paketen in einer neuen Version des Plug-Ins verwendet wird. Erforderliches Attribut.
<code>image</code>	Bilddatei	Das Symbol, das im Orchestrator-Bestand angezeigt wird. Erforderliches Attribut.
<code>display-name</code>	Zeichenfolge	Der Name, der im Orchestrator-Bestand angezeigt wird. Optionales Attribut.
<code>interface-mapping-allowed</code>	True oder False	VMware rät von der Schnittstellenzuordnung ab. Optionales Attribut.



**Tabelle 1-7.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
Keine	<ul style="list-style-type: none"> <li>■ &lt;description&gt;</li> <li>■ &lt;installation&gt;</li> <li>■ &lt;configuration&gt;</li> <li>■ &lt;finder-datasources&gt;</li> <li>■ &lt;inventory&gt;</li> <li>■ &lt;finders&gt;</li> <li>■ &lt;scripting-objects&gt;</li> <li>■ &lt;enumerations&gt;</li> </ul>

## Element „description“

Die Elemente <description> bieten Beschreibungen der Elemente des Plug-Ins, das in der API-Explorer-Dokumentation angezeigt wird.

Der Text, der in der API-Explorer-Dokumentation erscheint, wird zwischen den Tags <description> und </description> eingefügt.

Das Element <description> ist optional. Das Element <description> besitzt keine Attribute.

**Tabelle 1-8.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<ul style="list-style-type: none"> <li>■ &lt;module&gt;</li> <li>■ &lt;example&gt;</li> <li>■ &lt;trigger&gt;</li> <li>■ &lt;gauge&gt;</li> <li>■ &lt;finder&gt;</li> <li>■ &lt;constructor&gt;</li> <li>■ &lt;method&gt;</li> <li>■ &lt;object&gt;</li> <li>■ &lt;enumeration&gt;</li> </ul>	Keine

## Element „deprecated“

Mit dem Element <deprecated> werden veraltete Objekte und Methoden in der API-Explorer-Dokumentation markiert.

Der Text, der in der API-Explorer-Dokumentation erscheint, wird zwischen den Tags <deprecated> und </deprecated> eingefügt.

Das Element <deprecated> ist optional. Das Element <deprecated> besitzt keine Attribute.

**Tabelle 1-9.** Elementhierarchie

Übergeordnete Elemente	Untergeordnete Elemente
<ul style="list-style-type: none"> <li>■ &lt;method&gt;</li> <li>■ &lt;object&gt;</li> </ul>	Keine

## URL-Element

Das Element <url> stellt eine URL bereit, die auf eine externe Dokumentation über ein Objekt oder eine Auflistung verweist.

Sie geben eine URL zwischen den Tags <url> und </url> an.

Das Element <url> ist optional. Das Element <url> besitzt keine Attribute.

**Tabelle 1-10.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
■ <enumeration>	Keine
■ <object>	

## installation-Element

Das Element <installation> ermöglicht Ihnen die Installation eines Pakets oder das Ausführen eines Skripts, wenn der Server startet.

Das Element <installation> ist optional. Das Element <installation> besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
mode	always, never oder version	Wenn der mode-Wert aktiv ist, wird das folgende Verhalten bewirkt, sobald der Orchestrator-Server startet: <ul style="list-style-type: none"> <li>■ Die Aktion always wird ausgeführt</li> <li>■ Die Aktion never wird ausgeführt</li> <li>■ Die Aktion wird ausgeführt, wenn der Server eine neuere Version des Plug-Ins entdeckt</li> </ul> Erforderliches Attribut.

**Tabelle 1-11.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<module>	<action>

## Element „action“

Das Element <action> legt die Aktion fest, die beim Start des Orchestrator-Servers ausgeführt wird.

Die Attribute für das Element <action> geben den Pfad zum Orchestrator-Paket oder -Skript an, das das Verhalten des Plug-Ins beim Start definiert.

Das Element <action> ist optional. Ein Plug-In kann über beliebig viele <action>-Elemente verfügen. Das Element <action> besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
resource	Zeichenfolge	Der Pfad zum Java-Paket oder -Skript vom Stammordner der DAR-Datei. Erforderliches Attribut.
type	install-package oder execute-script	Entweder wird das angegebene Orchestrator-Paket auf dem Orchestrator-Server installiert oder das angegebene Skript ausgeführt. Erforderliches Attribut.

**Tabelle 1-12.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<installation>	Keine

## Element „finder-datasources“

Das Element `<finder-datasources>` ist der Container für die Elemente `<finder-datasource>`.

Das Element `<finder-datasources>` ist optional. Das Element `<finder-datasources>` besitzt keine Attribute.

**Tabelle 1-13.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;module&gt;</code>	<code>&lt;finder-datasource&gt;</code>

## Element „finder-datasource“

Das Element `<finder-datasource>` verweist auf die Java-Klassendatei der `IPluginAdaptor`-Implementierung, die Sie für das Plug-In erstellen.

Sie legen im `<finder-datasource>`-Element fest, wie Orchestrator auf die Objekte der integrierten Technologie zugreift. Das Element `<finder-datasource>` gibt die Java-Klasse des erstellten Plug-In-Adapters an. Die Adapterklasse des Plug-Ins instanziiert die erstellte Plug-In-Factory. Die Plug-In-Factory legt die Methoden fest, mit denen Objekte in der integrierten Technologie gefunden werden. Sie können die Zeitüberschreitungs-werte für die Finder-Methodenaufrufe, die die Factory durchführt, im Element `<finder-datasource>` festlegen. Für die unterschiedlichen Finder-Methoden von der `IPluginFactory`-Schnittstelle gelten unterschiedliche Zeitüberschreitungs-werte.

Das Element `<finder-datasource>` ist optional. Ein Plug-In kann über beliebig viele `<finder-datasources>`-Elemente verfügen. Das Element `<finder-datasource>` besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
<code>name</code>	Zeichenfolge	Kennzeichnet die Datenquelle in den <code>datasource</code> -Attributen des <code>&lt;finder&gt;</code> -Elements. Entspricht einer XML-id. Erforderliches Attribut.
<code>adaptor-class</code>	Java-Klasse	Verweist auf die <code>IPluginAdaptor</code> -Implementierung, die Sie für das Erstellen des Plug-In-Adapters festlegen. Beispiel: <code>com.vmware.plugins.sample.Adaptor</code> . Erforderliches Attribut.
<code>concurrent-call</code>	<code>true</code> (Standard) oder <code>false</code>	Ermöglicht mehreren Benutzern den gleichzeitigen Zugriff auf den Adapter. Sie müssen <code>concurrent-call</code> auf <code>false</code> setzen, wenn das gleichzeitige Aufrufen vom Plug-In nicht unterstützt wird. Optionales Attribut.
<code>invoker-mode</code>	<code>direct</code> (Standard) oder <code>timeout</code>	Legt einen Zeitüberschreitungs-wert für die Finder-Funktion fest. Wenn die Einstellung <code>direct</code> lautet, erfolgt keine Zeitüberschreitung. Lautet die Einstellung <code>timeout</code> , wendet der Orchestrator-Server die Zeitüberschreitungs-dauer an, die der Finder-Methode entspricht. Optionales Attribut.
<code>anonymous-login-mode</code>	<code>never</code> (Standard) oder <code>always</code>	Hiermit wird festgelegt, ob der Benutzername und das Kennwort des Benutzers an das Plug-In übergeben werden. Optionales Attribut.
<code>timeout-fetch-relation</code>	Zahl; Standard 30 Sekunden	Gilt für alle Aufrufe von <code>findRelation()</code> . Optionales Attribut.

Attribute	Wert	Beschreibung
timeout-find-all	Zahl; Standard 60 Sekunden	Gilt für alle Aufrufe von <code>findAll()</code> . Optionales Attribut.
timeout-find	Zahl; Standard 60 Sekunden	Gilt für alle Aufrufe von <code>find()</code> . Optionales Attribut.
timeout-has-children-in-relation	Zahl; Standard 2 Sekunden	Gilt für alle Aufrufe von <code>findChildrenInRelation()</code> . Optionales Attribut.
timeout-execute-plugin-command	Zahl; Standard 30 Sekunden	Gilt für alle Aufrufe von <code>executePluginCommand()</code> . Optionales Attribut.

**Tabelle 1-14.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<finder-datasources>	Keine

## inventory-Element

Das Element <inventory> definiert die Stammebene der hierarchischen Liste für das Plug-In, die in der Ansicht **Bestand** und in Dialogfeldern für die Objektauswahl im Orchestrator-Client angezeigt wird.

Das Element <inventory> stellt kein Objekt in der als Plug-In integrierten Anwendung dar, sondern repräsentiert das Plug-In selbst als Objekt in der Skripterstellung-API von Orchestrator.

Das Element <inventory> ist optional. Das Element <inventory> besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
type	Ein Orchestrator-Objekttyp	Der Typ des <finder>-Elements, der die Stammebene der Hierarchie von Objekten darstellt. Erforderliches Attribut.

**Tabelle 1-15.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<module>	Keine

## Element „finders“

Das Element <finders> ist der Container für die Elemente <finder>.

Das Element <finders> ist optional. Das Element <finders> besitzt keine Attribute.

**Tabelle 1-16.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<module>	<finder>

## Element „finder“

Das Element <finder> steht im Orchestrator-Client für einen Objekttyp, der über das Plug-In gefunden wurde.

Das Element <finder> identifiziert die Java-Klasse, die das vom Objekt-Finder dargestellte Objekt definiert. Das Element <finder> definiert, wie das Objekt in der Orchestrator-Client-Schnittstelle angezeigt wird. Es identifiziert auch das Skriptobjekt, das die Orchestrator-Skript-API zum Darstellen dieses Objekts definiert.

Finder dienen als Schnittstelle zwischen Objektformaten, die von verschiedenen Typen integrierter Technologien verwendet werden.

Das Element `<finder>` ist optional. Ein Plug-In kann über beliebig viele `<finder>`-Elemente verfügen. Das Element `<finder>` definiert die folgenden Attribute:

Attribute	Wert	Beschreibung
<code>type</code>	Ein Orchestrator-Objekttyp	Vom Finder dargestellter Objekttyp. Erforderliches Attribut.
<code>datasource</code>	Attribut <code>&lt;finder-datasource name&gt;</code>	Identifiziert die Java-Klasse, die das Objekt definiert, anhand der Datenquelle <code>refId</code> . Erforderliches Attribut.
<code>dynamic-finder</code>	Java-Methode	Definieren Sie eine benutzerdefinierte Finder-Methode, die Sie in einer <code>IDynamicFinder</code> -Instanz implementieren, um die ID und Eigenschaften eines Finders programmatisch zurückzugeben, statt sie in der Datei <code>vso.xml</code> zu definieren. Optionales Attribut.
<code>hidden</code>	<code>true</code> oder <code>false</code> (Standard)	Bei <code>true</code> wird der Finder im Orchestrator-Client ausgeblendet. Optionales Attribut.
<code>image</code>	Pfad zu einer Grafikdatei	Ein 16x16-Symbol, das den Finder in hierarchischen Listen im Orchestrator-Client darstellt. Optionales Attribut.
<code>java-class</code>	Name einer Java-Klasse	Die Java-Klasse, die das vom Finder gefundene Objekt definiert und einem Skriptobjekt zuordnet. Optionales Attribut.
<code>script-object</code>	<code>&lt;scripting-object type&gt;</code> -Attribut	Der <code>&lt;scripting-object&gt;</code> -Typ, dem dieser Finder ggf. zuzuordnen ist. Optionales Attribut.

**Tabelle 1-17.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;finders&gt;</code>	<ul style="list-style-type: none"> <li>■ <code>&lt;id&gt;</code></li> <li>■ <code>&lt;description&gt;</code></li> <li>■ <code>&lt;properties&gt;</code></li> <li>■ <code>&lt;default-sorting&gt;</code></li> <li>■ <code>&lt;inventory-children&gt;</code></li> <li>■ <code>&lt;relations&gt;</code></li> <li>■ <code>&lt;inventory-tabs&gt;</code></li> <li>■ <code>&lt;events&gt;</code></li> </ul>

## Eigenschaftenelement

Das Element `<properties>` ist der Container für die `<finder>``<property>`-Elemente.

Das Element `<properties>` ist optional. Das Element `<properties>` besitzt keine Attribute.

**Tabelle 1-18.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<code>&lt;finder&gt;</code>	<code>&lt;property&gt;</code>

## Eigenschaftselement

Das Element `<property>` ordnet die Eigenschaften des gefundenen Objekts den Java-Eigenschaften oder Methoden aufrufen zu.

Sie können die Methoden der Klasse `SDKFinderProperty` aufrufen, wenn Sie die Plug-In-Factory implementieren, um Eigenschaften für die Verarbeitung durch die Plug-In-Factory zu erhalten.

Sie können Objekteigenschaften in den Ansichten im Orchestrator-Client anzeigen oder ausblenden. Sie können auch die Enumeration verwenden, um Objekteigenschaften zu definieren.

Das Element `<property>` ist optional. Ein Plug-In kann über beliebig viele `<property>`-Elemente verfügen. Das Element `<property>` besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
<code>name</code>	Findername	Der Name, den <code>FinderResult</code> verwendet, um das Element zu speichern. Erforderliches Attribut.
<code>display-name</code>	Findername	Der angezeigte Eigenschaftensname. Optionales Attribut.
<code>bean-property</code>	Eigenschaftensname	Sie verwenden das Attribut <code>bean-property</code> , um eine Eigenschaft zu kennzeichnen, die Sie mit den Vorgängen <code>get</code> und <code>set</code> beziehen möchten. Wenn Sie eine Eigenschaft mit dem Namen <code>MyProperty</code> kennzeichnen, definiert das Plug-In die Vorgänge <code>getMyProperty</code> und <code>setMyProperty</code> . Sie können entweder <code>bean-property</code> oder <code>property-accessor</code> , aber nicht beide festlegen. Optionales Attribut.
<code>property-accessor</code>	Die Methode, die einen Eigenschaftswert von einem Objekt erhält	Das Attribut <code>property-accessor</code> ermöglicht Ihnen die Definition eines OGNL-Ausdrucks, um die Eigenschaften eines Objekts zu validieren. Sie können entweder <code>bean-property</code> oder <code>property-accessor</code> , aber nicht beide. Optionales Attribut.
<code>show-in-column</code>	<code>true</code> (Standard) oder <code>false</code>	Wenn <code>true</code> , wird diese Eigenschaft in der Ergebnistabelle des Orchestrator-Clients angezeigt. Optionales Attribut.
<code>show-in-description</code>	<code>true</code> (Standard) oder <code>false</code>	Wenn <code>true</code> , wird die Eigenschaft in der Objektbeschreibung angezeigt. Optionales Attribut.
<code>hidden</code>	<code>true</code> oder <code>false</code> (Standard)	Wenn <code>true</code> , wird diese Eigenschaft in allen Fällen angezeigt. Optionales Attribut.
<code>linked-enumeration</code>	Name der Enumeration	Verknüpft eine Findereigenschaft mit einer Enumeration. Optionales Attribut.

**Tabelle 1-19.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;properties&gt;</code>	Untergeordnete Elemente

## relations-Element

Das Element `<relations>` ist der Container für die `<finder>``<relation>`-Elemente.

Das Element `<relations>` ist optional. Das Element `<relations>` besitzt keine Attribute.

**Tabelle 1-20.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<code>&lt;finder&gt;</code>	<code>&lt;relation&gt;</code>

## relation-Element

Das Element `<relation>` definiert, wie die Beziehungen von Objekten zu anderen Objekten aussehen.

Sie definieren den Relationsnamen im Element `<relation>`.

Das Element `<relation>` ist optional. Ein Plug-In kann über beliebig viele `<relation>`-Elemente verfügen. Das Element `<relation>` besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
name	Beziehungsname	Ein Name für diese Beziehung. Erforderliches Attribut.
type	Orchestrator-Objektyp	Der Typ des Objekts, der mit einem anderen Objekt über diese Beziehung verbunden ist. Erforderliches Attribut.
cardinality	to-one oder to-many	Definiert die Beziehung zwischen den Objekten als 1:1 oder 1:N. Optionales Attribut.

**Tabelle 1-21.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;relations&gt;</code>	Keine

## ID-Element

Das `<id>`-Element definiert eine Methode, um eine eindeutige ID für das Objekt zu erhalten, das der Finder identifiziert.

Das Element `<id>` ist optional. Das Element `<id>` besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
accessor	Methodenname	Das <code>accessor</code> -Attribut ermöglicht Ihnen die Definition eines OGNL-Ausdrucks, um die Eigenschaften eines Objekts zu validieren. Erforderliches Attribut.

**Tabelle 1-22.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;finder&gt;</code>	Keine

## inventory-children-Element

Das Element `<inventory-children>` definiert die Hierarchie der Listen, die in der Ansicht **Bestand** und in Dialogfeldern für die Objektauswahl im Orchestrator-Client angezeigt werden.

Das Element `<inventory-children>` ist optional. Das Element `<inventory-children>` besitzt keine Attribute.

**Tabelle 1-23.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<code>&lt;finder&gt;</code>	<code>&lt;relation-link&gt;</code>

## relation-link-Element

Das Element `<relation-link>` definiert die Hierarchien zwischen übergeordneten und untergeordneten Objekten auf der Registerkarte **Bestand**.

Das Element `<relation-link>` ist optional. Ein Plug-In kann über beliebig viele `<relation-link>`-Elemente verfügen. Das Element `<relation-link>` besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
name	Beziehungsname	Ein refid zu einem Beziehungsnamen. Erforderliches Attribut.

**Tabelle 1-24.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;inventory-children&gt;</code>	Keine

## Element „events“

Das Element `<events>` ist der Container für die Elemente `<trigger>` und `<gauge>`.

Das Element `<events>` kann eine unbegrenzte Anzahl an Auslösern oder Kontrollen enthalten.

Das Element `<events>` ist optional. Das Element `<events>` besitzt keine Attribute.

**Tabelle 1-25.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;finder&gt;</code>	<ul style="list-style-type: none"> <li>■ <code>&lt;trigger&gt;</code></li> <li>■ <code>&lt;gauge&gt;</code></li> </ul>

## Element „trigger“

Das Element `<trigger>` kennzeichnet die Auslöser, die Sie für diesen Finder verwenden können. Zum Festlegen von Auslösern müssen Sie die Methoden `registerEventPublisher()` und `unregisterEventPublisher()` von `IPluginAdaptor` implementieren.

Das Element `<trigger>` ist optional. Das Element `<trigger>` besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
name	Auslösername	Der Name des Auslösers. Erforderliches Attribut.



**Tabelle 1-26.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<events>	<ul style="list-style-type: none"> <li>■ &lt;description&gt;</li> <li>■ &lt;trigger-properties&gt;</li> </ul>

## trigger-properties-Element

Das Element <trigger-properties> ist der Container für die <trigger-property>-Elemente.

Das Element <trigger-properties> ist optional. Das Element <trigger-properties> besitzt keine Attribute.

**Tabelle 1-27.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<trigger>	<trigger-property>

## trigger-property-Element

Das Element <trigger-property> definiert die Eigenschaften, die ein Triggerobjekt definieren.

Das Element <trigger-property> ist optional. Ein Plug-In kann über beliebig viele <trigger-property>-Elemente verfügen. Das Element <trigger-property> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
name	Auslösername	Ein Name für den Auslöser. Optionales Attribut.
display-name	Auslösername	Der Name, der im Orchestrator-Client angezeigt wird. Optionales Attribut.
type	Auslösertyp	Der Objekttyp, der den Auslöser definiert. Erforderliches Attribut.

**Tabelle 1-28.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<Auslösereigenschaften>	Keine

## Element „gauge“

Das Element <gauge> definiert die Kontrollen, die Sie für diesen Finder verwenden können. Zum Festlegen von Kontrollen müssen Sie die Methoden `registerEventPublisher()` und `unregisterEventPublisher()` von `IPluginAdaptor` implementieren.

Das Element <gauge> ist optional. Ein Plug-In kann über beliebig viele <gauge>-Elemente verfügen. Das Element <gauge> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
name	Name der Kontrolle	Ein Name für die Kontrolle. Erforderliches Attribut.
min-value	Zahl	Mindestschwellenwert. Optionales Attribut.
max-value	Zahl	Maximalschwellenwert. Optionales Attribut.

Typ	Wert	Beschreibung
unit	Objektyp	Objektyp, der die Kontrolle definiert. Erforderliches Attribut.
format	String	Das Format des überwachten Werts. Optionales Attribut.

**Tabelle 1-29.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<events>	<description>

## scripting-objects-Element

Das Element <scripting-objects> ist der Container für die <object>-Elemente.

Das Element <scripting-objects> ist optional. Das Element <scripting-objects> besitzt keine Attribute.

**Tabelle 1-30.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<module>	<object>

## Objektelement

Das Element <object> ordnet die Konstruktoren der Plug-In-Technologie, Attribute und Methoden den JavaScript-Objektypen zu, die durch die Skripterstellung-API von Orchestrator bereitgestellt werden.

Unter „[Benennen von Plug-In-Objekten](#)“, auf Seite 59 finden Sie Informationen über Namenskonventionen für Objekte.

Das Element <object> ist optional. Ein Plug-In kann über beliebig viele <object>-Elemente verfügen. Das Element <object> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
script-name	JavaScript-Name	Klassenname für die Skripterstellung. Muss global eindeutig sein. Erforderliches Attribut.
java-class	Java-Klasse	Die Java-Klasse, die in dieser JavaScript-Klasse enthalten ist. Erforderliches Attribut.
create	true (Standard) oder false	Wenn true, können Sie eine neue Instanz dieser Klasse erstellen. Optionales Attribut.
strict	true oder false (Standard)	Wenn true, können Sie nur Methoden aufrufen, die Sie in der vso.xml-Datei anmerken oder deklarieren. Optionales Attribut.
is-deprecated	true oder false (Standard)	Wenn true, ordnet das Objekt eine veraltete Java-Klasse zu. Optionales Attribut.
since-version	String	Version, seit der die Java-Klasse veraltet ist. Optionales Attribut.

**Tabelle 1-31.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<scripting-objects>	<ul style="list-style-type: none"> <li>■ &lt;description&gt;</li> <li>■ &lt;deprecated&gt;</li> <li>■ &lt;url&gt;</li> <li>■ &lt;constructors&gt;</li> <li>■ &lt;attributes&gt;</li> <li>■ &lt;methods&gt;</li> <li>■ &lt;singleton&gt;</li> </ul>

**Element „constructors“**

Das Element <constructors> ist der Container für die <object><constructor>-Elemente.

Das Element <constructors> ist optional. Das Element <constructors> besitzt keine Attribute.

**Tabelle 1-32.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<object>	<constructor>

**Element „constructor“**

Das Element <constructor> definiert eine Konstruktormethode. Mithilfe des Elements <constructor> wird Dokumentation im API-Explorer generiert.

Das Element <constructor> ist optional. Ein Plug-In kann über beliebig viele <constructor>-Elemente verfügen. Das Element <constructor> besitzt keine Attribute.

**Tabelle 1-33.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<constructors>	<ul style="list-style-type: none"> <li>■ &lt;description&gt;</li> <li>■ &lt;parameters&gt;</li> </ul>

**Element „parameters“ des Konstruktors**

Das Element <parameters> ist der Container für die <constructor><parameter>-Elemente.

Das Element <parameters> ist optional. Das Element <parameters> besitzt keine Attribute.

**Tabelle 1-34.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<constructor>	<parameter>

**Konstruktor-Parameterelement**

Das <parameter>-Element definiert die Parameter des Konstruktors.

Das Element <parameter> ist optional. Ein Plug-In kann über beliebig viele <parameter>-Elemente verfügen. Das Element <parameter> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
name	String	In API-Dokumentation zu verwendender Parametername. Erforderliches Attribut.
type	Orchestrator-Parametertyp	In API-Dokumentation zu verwendender Parametertyp. Erforderliches Attribut.
is-optional	true oder false	Bei true kann der Wert null sein. Optionales Attribut.
since-version	String	Methodenversion. Optionales Attribut.

**Tabelle 1-35.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<parameters>	Keine

## Element „attributes“

Das Element <attributes> ist der Container für die <object><attribute>-Elemente.

Das Element <attributes> ist optional. Das Element <attributes> besitzt keine Attribute.

**Tabelle 1-36.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<object>	<attribute>

## Element „attribute“

Das Element <attribute> ordnet die Attribute einer Java-Klasse von der Plug-In-Technologie zu JavaScript-Attributen zu, die das Orchestrator-JavaScript-Modul zur Verfügung stellt.

Das Element <attribute> ist optional. Ein Plug-In kann über beliebig viele <attribute>-Elemente verfügen. Das Element <attribute> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
java-name	Java-Attribut	Name des Java-Attributs. Erforderliches Attribut.
script-name	JavaScript-Objekt	Name des entsprechenden JavaScript-Objekts. Erforderliches Attribut.
return-type	Zeichenfolge	Der Objekttyp, der durch dieses Attribut zurückgegeben wird. Wird in der API-Explorer-Dokumentation angezeigt. Optionales Attribut. <b>HINWEIS</b> Wenn der JavaScript-Rückgabebetyp <code>Properties</code> ist, sind die unterstützten, zugrunde liegenden Java-Implementierungen <code>java.util.HashMap</code> und <code>java.util.Hashtable</code> .
read-only	true oder false	Wenn true, können Sie dieses Attribut nicht ändern. Optionales Attribut.
is-optional	true oder false	Wenn true, kann dieses Feld einen Nullwert aufweisen. Optionales Attribut.

Typ	Wert	Beschreibung
show-in-api	true oder false	Wenn false, wird dieses Attribut nicht in der API-Dokumentation angezeigt. Optionales Attribut.
is-deprecated	true oder false	Wenn true, ordnet das Objekt ein veraltetes Attribut zu. Optionales Attribut.
since-version	Zahl	Die Version, ab der das Attribut veraltet war. Optionales Attribut.

**Tabelle 1-37.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<attributes>	Keine

## Methodenelement

Das Element <methods> ist der Container für die <object><method>-Elemente.

Das Element <methods> ist optional. Das Element <methods> besitzt keine Attribute.

**Tabelle 1-38.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<object>	<method>

## Methodenelement

Das Element <method> ordnet eine Java-Methode von der Plug-In-Technologie einer Methode zu, die die Orchestrator-JavaScript-Engine zur Verfügung stellt.

Das Element <method> ist optional. Ein Plug-In kann über beliebig viele <method>-Elemente verfügen. Das Element <method> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
java-name	Java-Methode	Name der Java-Methodensignatur mit Argumenttypen in Klammern, beispielsweise <code>getVms(DataStore)</code> . Erforderliches Attribut.
script-name	JavaScript-Methode	Name der entsprechenden JavaScript-Methode. Erforderliches Attribut.
return-type	Java-Objektyp	Der Typ, den diese Methode erhält. Optionales Attribut. <b>HINWEIS</b> Wenn der JavaScript-Rückgabebetyp <code>Properties</code> ist, sind die unterstützten, zugrunde liegenden Java-Implementierungen <code>java.util.HashMap</code> und <code>java.util.Hashtable</code> .
static	true oder false	Wenn true, ist diese Methode statisch. Optionales Attribut.
show-in-api	true oder false	Wenn false, wird diese Methode nicht in der API-Dokumentation angezeigt. Optionales Attribut.

Typ	Wert	Beschreibung
is-deprecated	true oder false	Wenn true, ordnet das Objekt eine veraltete Methode zu. Optionales Attribut.
since-version	Zahl	Die Version, ab der die Methode veraltet war. Optionales Attribut.

**Tabelle 1-39.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<methods>	<ul style="list-style-type: none"> <li>■ &lt;deprecated&gt;</li> <li>■ &lt;description&gt;</li> <li>■ &lt;example&gt;</li> <li>■ &lt;parameters&gt;</li> </ul>

### Element „example“

Das Element <example> ermöglicht das Hinzufügen von Beispielcode für JavaScript-Methoden, die in der API-Explorer-Dokumentation vorkommen.

Das Element <example> ist optional. Das Element <example> besitzt keine Attribute.

**Tabelle 1-40.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<method>	<ul style="list-style-type: none"> <li>■ &lt;code&gt;</li> <li>■ &lt;description&gt;</li> </ul>

### Element „code“

Das Element <code> stellt Beispielcode bereit, der in der API-Explorer-Dokumentation angezeigt wird.

Sie stellen das Codebeispiel zwischen den Tags <code> und </code> bereit. Das Element <code> ist optional. Das Element <code> besitzt keine Attribute.

**Tabelle 1-41.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<example>	Keine

### Methodenparameterelement

Das Element <parameters> ist der Container für die <method><parameter>-Elemente.

Das Element <parameters> ist optional. Das Element <parameters> besitzt keine Attribute.

**Tabelle 1-42.**

Übergeordnetes Element	Untergeordnetes Element
<method>	<parameter>

### Methodenparameterelement

Das Element <parameter> definiert die Eingabeparameter der Methode.

Das Element <parameter> ist optional. Ein Plug-In kann über beliebig viele <parameter>-Elemente verfügen. Das Element <parameter> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
name	String	Parametername. Erforderliches Attribut.
type	Orchestrator-Parametertyp	Parametertyp. Erforderliches Attribut.
is-optional	true oder false	Bei true kann der Wert null sein. Optionales Attribut.
since-version	String	Methodenversion. Optionales Attribut.

**Tabelle 1-43.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<parameters>	Keine

## Singleton-Element

Das Element <singleton> erstellt ein JavaScript-Skriptobjekt als Singleton-Instanz.

Ein Singleton-Objekt verhält sich wie eine statische Java-Klasse. Singleton-Objekte definieren generische Objekte zur Verwendung durch das Plug-In. Sie definieren nicht eine bestimmte Instanz eines Objekts, auf das Orchestrator in der integrierten Technologie zugreift. Beispiel: Sie können ein Singleton-Objekt verwenden, um eine Verbindung zu einer integrierten Technologie herzustellen.

Das Element <singleton> ist optional. Das Element <singleton> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
script-name	JavaScript-Objekt	Name des entsprechenden JavaScript-Objekts. Erforderliches Attribut.
datasource	Java-Objekt	Das Java-Quellobjekt für das JavaScript-Objekt. Erforderliches Attribut.

**Tabelle 1-44.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<object>	Keine

## Element „enumerations“

Das Element <enumerations> ist der Container für die Elemente <enumeration>.

Das Element <enumerations> ist optional. Das Element <enumerations> besitzt keine Attribute.

**Tabelle 1-45.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<module>	<enumeration>

## Element „enumeration“

Das Element `<enumeration>` definiert häufige Werte, die für alle Objekte eines bestimmten Typs gelten.

Wenn alle Objekte eines bestimmten Typs ein bestimmtes Attribut erfordern und wenn der Wertebereich für dieses Attribut limitiert ist, können Sie die verschiedenen Werte als Enumerationseinträge definieren. Wenn beispielsweise ein Objekttyp das Attribut `color` erfordert und die einzig verfügbaren Farben rot, blau und grün sind, können Sie drei Enumerationseinträge zum Definieren dieser drei Farbwerte erstellen. Sie definieren Einträge als untergeordnete Elemente des Elements „enumeration“.

Das Element `<enumeration>` ist optional. Ein Plug-In kann über beliebig viele `<enumeration>`-Elemente verfügen. Das Element `<enumeration>` besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
type	Orchestrator-Objekttyp	Enumerationstyp. Erforderliches Attribut.

**Tabelle 1-46.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;enumerations&gt;</code>	<ul style="list-style-type: none"> <li>■ <code>&lt;url&gt;</code></li> <li>■ <code>&lt;description&gt;</code></li> <li>■ <code>&lt;entries&gt;</code></li> </ul>

## Element „entries“

Das Element `<entries>` ist der Container für die Elemente `<enumeration>``<entry>`.

Das Element `<entries>` ist optional. Das Element `<entries>` besitzt keine Attribute.

**Tabelle 1-47.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<code>&lt;enumeration&gt;</code>	<code>&lt;entry&gt;</code>

## Element „entry“

Das Element `<entry>` bietet einen Wert für ein Enumerationsattribut.

Das Element `<entry>` ist optional. Ein Plug-In kann über beliebig viele `<entry>`-Elemente verfügen. Das Element `<entry>` besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
id	Text	Der von Objekten verwendete Bezeichner, um den Enumerationseintrag als Attribut festzulegen. Erforderliches Attribut.
name	Text	Der Name des Eintrags. Erforderliches Attribut.

**Tabelle 1-48.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;entries&gt;</code>	Keine



## Best Practices zur Entwicklung von Orchestrator-Plug-Ins

Sie können bestimmte Aspekte der entwickelten Orchestrator Plug-Ins verbessern durch Verständnis der Struktur und des Inhalts der Plug-Ins sowie durch Verständnis darüber, wie man bestimmte Probleme vermeidet.

- [Methoden zum Erstellen eines Orchestrator-Plug-Ins](#) auf Seite 89  
Zum Erstellen von Orchestrator-Plug-Ins können Sie verschiedene Methoden anwenden. Sie können entweder die Ebenen des Plug-Ins einzeln erstellen oder Sie erstellen alle Ebenen gleichzeitig.
- [Typen von Orchestrator-Plug-Ins](#) auf Seite 91  
Mithilfe von Plug-Ins können Sie Bibliotheken und Dienstprogramme für allgemeine Zwecke (etwa XML oder SSH) in Orchestrator integrieren oder auch ganze Systeme (etwa vCloud Director). Je nach Technologie, die Sie in Orchestrator integrieren, fallen Plug-Ins in verschiedene Kategorien: Plug-Ins für Dienste, für allgemeine Zwecke oder für Systeme.
- [Plug-In-Implementierung](#) auf Seite 94  
Sie können beim Strukturieren Ihrer Plug-Ins bestimmte hilfreiche Methoden und Techniken verwenden, die erforderlichen Java-Klassen und JavaScript-Objekte implementieren, Plug-In-Workflows und -Aktionen entwickeln und die Workflowdarstellung bereitstellen.
- [Empfehlungen zur Entwicklung von Orchestrator-Plug-Ins](#) auf Seite 98  
Indem Sie sich beim Entwickeln der verschiedenen Komponenten für Ihre Orchestrator-Plug-Ins an bestimmte Grundsätze halten, können Sie die Qualität der Plug-Ins verbessern.
- [Dokumentierung von Plug-In-Benutzeroberflächen-Zeichenfolgen und -APIs](#) auf Seite 101  
Wenn Sie Benutzeroberflächen-Zeichenfolgen für Orchestrator-Plug-Ins und die zugehörige API-Dokumentation schreiben, folgen Sie den allgemein anerkannten Stil- und Formatregeln.

## Methoden zum Erstellen eines Orchestrator-Plug-Ins

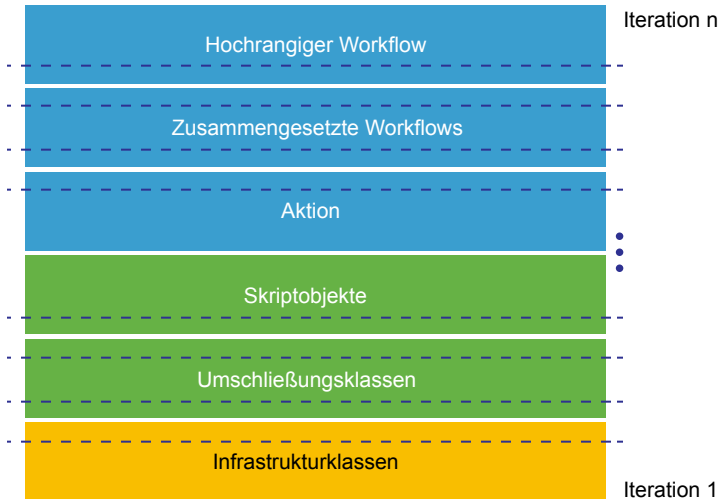
Zum Erstellen von Orchestrator-Plug-Ins können Sie verschiedene Methoden anwenden. Sie können entweder die Ebenen des Plug-Ins einzeln erstellen oder Sie erstellen alle Ebenen gleichzeitig.

Weitere Informationen zu Plug-In-Ebenen finden Sie unter „[Struktur eines Orchestrator-Plug-Ins](#)“, auf Seite 50.

### Bottom-Up-Plug-In-Entwicklung

Ein Plug-In kann Schicht für Schicht mit dem Bottom-Up-Entwicklungsansatz erstellt werden.

Der Bottom-Up-Entwicklungsansatz erstellt das Plug-In Schicht für Schicht, beginnend mit den Schichten der untersten Ebenen und weiter mit den Schichten der höheren Ebenen. Wird dieser Ansatz mit einem interaktiven und iterativen Entwicklungsansatz kombiniert, wird bei jeder Iteration entweder ein Teil oder eine ganze Schicht fertiggestellt. Nach dem Abschluss von Iteration N ist das Plug-In vollständig.

**Abbildung 1-5. Bottom-Up-Plug-In-Entwicklung**

Ein Vorteil des Bottom-Up-Plug-In-Entwicklungsansatzes ist, dass die Entwicklung sich auf eine Schicht nach der anderen konzentriert.

Berücksichtigen Sie auch die folgenden Nachteile des Bottom-Up-Plug-In-Entwicklungsansatzes.

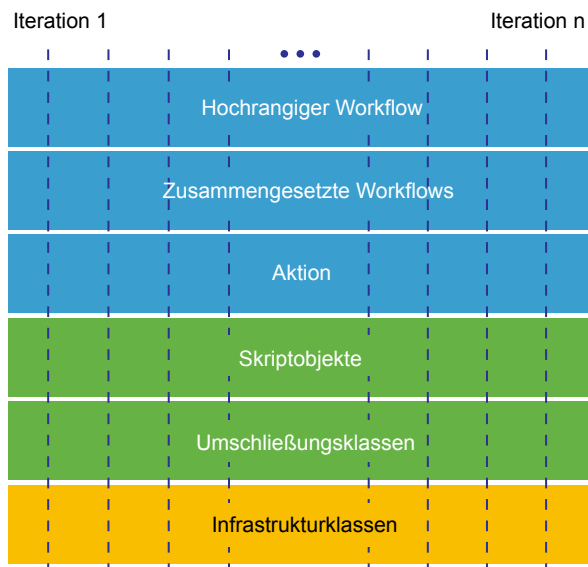
- Es ist schwierig den Fortschritt der Plug-In-Entwicklung zu präsentieren, bis einige Elemente eingefügt wurden.
- Das passt nicht gut in einen agilen Entwicklungsprozess.

Der Bottom-Up-Entwicklungsprozess wird für einige kleine Plug-Ins mit wenigen oder keinen Umschließungsklassen, Skriptobjekten, Aktionen oder Workflows als ausreichend betrachtet.

### Top-down-Plug-In-Entwicklung

Ein Plug-In kann mithilfe eines Top-down-Ansatzes konstruiert werden, wobei eine Unterteilung der Funktionalität von der obersten bis hin zur untersten Ebene erfolgt.

Wird der Top-down-Ansatz mit einem agilen Entwicklungsprozess kombiniert, wird bei jeder Iteration neue Funktionalität bereitgestellt. Nach dem Abschluss von Iteration N ist das Plug-In vollständig implementiert.

**Abbildung 1-6. Top-down-Plug-In-Entwicklung**

Der Top-down-Ansatz bei der Entwicklung von Plug-Ins bietet die folgenden Vorteile:

- Der Fortschritt bei der Entwicklung des Plug-Ins ab der ersten Iteration ist einfach zu sehen, da bei jeder Iteration neue Funktionalität fertiggestellt wird und das Plug-In nach jeder Iteration veröffentlicht und benutzt werden kann.
- Durch die vertikale Unterteilung der Funktionalität können die Erfolgskriterien und Fortschritte eindeutig festgelegt und die Kommunikation zwischen Entwicklern, Produktmanagement und Quality Assurance (QA)-Technikern verbessert werden.
- Die QA-Techniker können von Beginn des Entwicklungsprozesses an testen und automatisieren. Dieser Ansatz ermöglicht wertvolle Rückmeldungen und verkürzt die Gesamtbereitstellungsdauer des Projekts.

Ein Vorteil der Top-down-Entwicklung von Plug-Ins ist, dass die Entwicklung auf verschiedenen Ebenen gleichzeitig erfolgt.

Die Top-down-Entwicklung wird für die meisten Plug-Ins empfohlen. Sie eignet sich für Plug-Ins mit dynamischen Anforderungen.

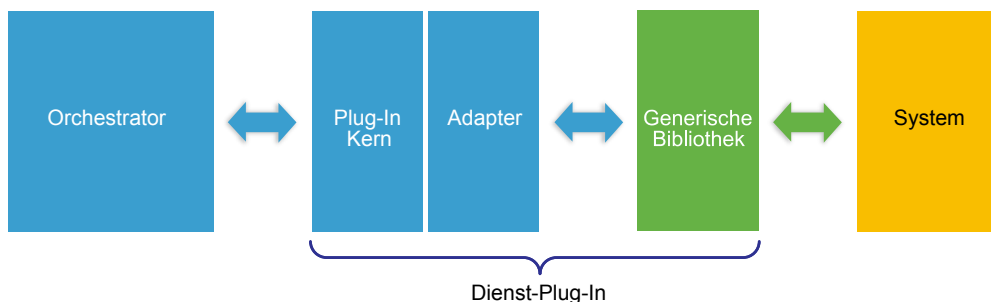
## Typen von Orchestrator-Plug-Ins

Mithilfe von Plug-Ins können Sie Bibliotheken und Dienstprogramme für allgemeine Zwecke (etwa XML oder SSH) in Orchestrator integrieren oder auch ganze Systeme (etwa vCloud Director). Je nach Technologie, die Sie in Orchestrator integrieren, fallen Plug-Ins in verschiedene Kategorien: Plug-Ins für Dienste, für allgemeine Zwecke oder für Systeme.

### Plug-Ins für Dienste

Plug-Ins für Dienste oder Plug-Ins für allgemeine Zwecke bieten Funktionen, die als Dienst innerhalb von Orchestrator angesehen werden können.

**Abbildung 1-7.** Architektur der Plug-Ins für Dienste



Plug-Ins für Dienste stellen generische Bibliotheken oder Dienstprogramme für Orchestrator bereit, wie XML, SSH oder SOAP. Die folgenden in Orchestrator verfügbaren Plug-Ins sind Beispiele für Dienst-Plug-Ins.

<b>JDBC-Plug-In</b>	Damit können Sie jede Datenbank in einem Workflow verwenden.
<b>Mail-Plug-In</b>	Damit können Sie E-Mails in einem Workflow senden.
<b>SSH-Plug-In</b>	Damit können Sie SSH-Verbindungen öffnen und Befehle in einem Workflow ausführen.
<b>XML-Plug-In</b>	Damit können Sie XML-Dokumente in einem Workflow verwalten.

Plug-Ins für Dienste haben die folgenden Merkmale:

<b>Komplexität</b>	Plug-Ins für Dienste weisen einen Komplexitätsgrad von niedrig bis mittel auf. Plug-Ins für Dienste machen eine spezifische Bibliothek oder Teile einer Bibliothek innerhalb von Orchestrator zugänglich, um konkrete Funktionen bereitzustellen. Das XML-Plug-In fügt beispielsweise der Orchestrator-JavaScript-API eine Implementierung eines DOM (Document Object Model)-XML-Parsers hinzu.
<b>Größe</b>	Plug-Ins für Dienste weisen eine relativ kleine Größe auf. Sie erfordern denselben grundlegenden Satz an Klassen wie alle Plug-Ins sowie weitere Klassen, die neue Skriptobjekte für neue Funktionen bieten.
<b>Bestandsliste</b>	Plug-Ins für Dienste benötigen eine kleine Bestandsliste von Objekten, um zu funktionieren, oder sie benötigen überhaupt keine Bestandsliste. Plug-Ins für Dienste haben ein generisches und ein kleines Objektmodell. Daher müssen Sie dieses Modell in der Orchestrator-Bestandsliste überhaupt nicht zeigen.

### Plug-Ins für Systeme

Plug-Ins für Systeme verbinden die Engine für Orchestrator-Workflows mit einem externen System, sodass Sie das externe System steuern können.

Es folgen einige Beispiele für Plug-Ins für Systeme.

<b>vCenter Server-Plug-In</b>	Damit können Sie vCenter Server-Instanzen mithilfe von Workflows verwalten.
<b>vCloud Director-Plug-In</b>	Damit können Sie mit einer vCloud Director-Installation in einem Workflow interagieren.
<b>Cisco UCSM Plug-In</b>	Damit können Sie mit Cisco-Einheiten in einem Workflow interagieren.

Es folgen die Hauptmerkmale von Plug-Ins für Systeme.

<b>Komplexität</b>	Plug-Ins für Systeme weisen höhere Komplexität als herkömmliche Plug-Ins auf, da die von ihnen eingesetzten Technologien relativ komplex sind. Plug-Ins für Systeme stellen alle Elemente des externen Systems im Orchestrator dar, die mit dem externen System interagieren und ihre Funktionalität in Orchestrator bereitstellen. Wenn das externe System einen Integrationsmechanismus bietet, können Sie diesen verwenden, um die Funktionalität des Systems leichter zugänglich zu machen. Zusätzlich zum Darstellen der Elemente des externen Systems in Orchestrator müssen Plug-Ins für Systeme ggf. auch hohe Skalierbarkeit bieten, einen Cache-Mechanismus bereitstellen, Ereignisse und Benachrichtigungen handhaben können usw.
<b>Größe</b>	Die Größe von Plug-Ins für Systeme reicht von mittel bis groß. Plug-Ins für Systeme benötigen zusätzlich zum grundlegenden Satz an Klassen zahlreiche weitere Klassen, da sie normalerweise eine große Anzahl an Skriptobjekten bieten. Plug-Ins für Systeme benötigen ggf. ein anderes Hilfsprogramm und Hilfsklassen, die mit ihnen interagieren.
<b>Bestandsliste</b>	In der Regel verfügen Plug-Ins für Systeme über eine große Anzahl an Objekten. Sie müssen diese Objekte in der Bestandsliste richtig darstellen, damit Sie sie lokalisieren und in Orchestrator mühelos mit ihnen arbeiten können. Aufgrund der großen Anzahl an Objekten, die Plug-Ins für Systeme darstellen müssen, sollten Sie ein Hilfstool oder einen Prozess erstellen, um für das Plug-In möglichst viel Code automatisch zu generieren. Das vCenter Server-Plug-In bietet ein solches Tool.



Beim Entwickeln von Plug-Ins für ressourcenorientierte Systeme können Sie Folgendes berücksichtigen.

- Wenn Sie REST oder nur HTTP in Verbindung mit XML verwenden, erhalten Sie eine oder mehrere XML-Schemadateien, um Nachrichten lesen und schreiben zu können. Über diese Schemas können Sie einen Satz an Klassen generieren, die das Objektmodell definieren. Mit diesem Satz an Klassen wird nur der Status der Objekte definiert, da die Vorgänge implizit mit den HTTP-Methoden definiert werden (beispielsweise beim vCloud Director-Plug-In), oder explizit mit einigen speziellen XML-Nachrichten (beispielsweise beim Cisco UCSM Plug-In).
- Sie müssen den Kommunikationsmechanismus in einem anderen Satz an Klassen implementieren. Mit diesem Satz an Klassen wird ein neues Objektmodell definiert, das mit dem ursprünglichen Objektmodell interagiert. Das Objektmodell für die Kommunikationsmechanismen besteht nur aus Objekten und Methoden.
- Sie können sowohl das ursprüngliche Objektmodell als auch das Objektmodell für den Kommunikationsmechanismus im Orchestrator zeigen. Auf diese Weise kann die Komplexität etwas erhöht werden, abhängig davon, auf welche Weise beide Objektmodelle angezeigt werden, und davon, ob Sie verwandte Objekte von beiden Seiten (zum Simulieren eines objektorientierten Systems) zusammenführen oder sie getrennt halten.

## Plug-In-Implementierung

Sie können beim Strukturieren Ihrer Plug-Ins bestimmte hilfreiche Methoden und Techniken verwenden, die erforderlichen Java-Klassen und JavaScript-Objekte implementieren, Plug-In-Workflows und -Aktionen entwickeln und die Workflowdarstellung bereitstellen.

- [Projektstruktur](#) auf Seite 94  
Sie können eine Standardstruktur für die Projekte der Orchestrator-Plug-Ins anwenden.
- [Projektinterne Ansätze](#) auf Seite 95  
Sie können bestimmte Ansätze beim Implementieren Ihres Plug-Ins anwenden, z. B. Objekte zwischenspeichern, Objekte in den Hintergrund setzen, Objekte klonen usw. Durch solche Ansätze können Sie die Leistung Ihrer Plug-Ins verbessern, Parallelitätsprobleme vermeiden und die Reaktionsfähigkeit des Orchestrator-Clients erhöhen.
- [Workflow-interne Ansätze](#) auf Seite 96  
Sie können einen Workflow implementieren, um langfristige Vorgänge zu überwachen, die das Orchestrator-Plug-In durchführt.
- [Workflows und Aktionen](#) auf Seite 97  
Sie können bestimmte bewährte Vorgehensweisen verwenden, um die Entwicklung und Verwendung von Workflows zu vereinfachen.
- [Workflowpräsentation](#) auf Seite 97  
Wenn Sie die Präsentation eines Workflows erstellen, müssen Sie bestimmte Strukturen und Regeln anwenden.

## Projektstruktur

Sie können eine Standardstruktur für die Projekte der Orchestrator-Plug-Ins anwenden.

Sie können eine Maven-Standardstruktur mit Modulen für Ihre Plug-In-Projekte verwenden, um klarer zu machen, wo sich die Funktionen befinden.

**Tabelle 1-49.** Struktur eines Plug-In-Projekts

Modul	Beschreibung
/myAwesomePlugin-plugin	Der Stamm des Plug-In-Projekts.
/o11nplugin-myAwesomePlugin	Das Modul, das die endgültige Plug-In-DAR-Datei erstellt.

**Tabelle 1-49.** Struktur eines Plug-In-Projekts (Fortsetzung)

Modul	Beschreibung
/o11nplugin-myAwesomePlugin-config	Das Modul, dass die Webanwendung für die Plug-In-Konfiguration enthält. Es wird eine standardmäßige WAR-Datei generiert.
/o11nplugin-myAwesomePlugin-core	Das Modul, das die Klassen enthält, die beliebige Orchestrator-Plug-In-Schnittstellen sowie andere Hilfsklassen, die sie verwenden, implementieren. Es wird eine standardmäßige JAR-Datei generiert.
/o11nplugin-myAwesomePlugin-model	Das Modul, dass die Klassen enthält, mit denen Sie die Technologie von Drittanbietern über das Plug-In besser in Orchestrator integrieren können. Die Klassen dürfen keine direkte Referenz auf die standardmäßigen Orchestrator-Plug-In-APIs enthalten.
/o11nplugin-myAwesomePlugin-package	Das Modul, das eine externe Orchestrator-Paketdatei mit Aktionen und Workflows importiert, die in der endgültigen Plug-In-DAR-Datei einbezogen werden soll. Das Modul ist optional.

### Projektinterne Ansätze

Sie können bestimmte Ansätze beim Implementieren Ihres Plug-Ins anwenden, z. B. Objekte zwischenspeichern, Objekte in den Hintergrund setzen, Objekte klonen usw. Durch solche Ansätze können Sie die Leistung Ihrer Plug-Ins verbessern, Parallelitätsprobleme vermeiden und die Reaktionsfähigkeit des Orchestrator-Clients erhöhen.

### Cache-Objekte

Ihr Plug-In kann mit einem Remotedienst interagieren. Diese Interaktion wird von lokalen Objekten bereitgestellt, die Remoteobjekte auf der Dienstseite darstellen. Sie können die lokalen Objekte zwischenspeichern, statt sie jedes mal vom Remoteservice abzurufen, um eine gute Leistung des Plug-Ins und eine gute Reaktionsfähigkeit der Orchestrator-Benutzeroberfläche zu gewährleisten. Überlegen Sie sich den Umfang des Cache, z. B. ein Cache für alle Plug-In-Clients, ein Cache pro Benutzer des Plug-Ins oder ein Cache pro Benutzer des Drittanbieterdienstes. Wenn der Cachemechanismus implementiert ist, wird er in die Plug-In-Schnittstelle integriert und dient zum Suchen und zur Invalidierung von Objekten.

### Objekte in den Hintergrund setzen

Wenn Sie eine umfangreiche Liste von Objekten in der Plug-In-Bestandsliste anzeigen müssen und keine schnelle Möglichkeit zum Abrufen dieser Objekte haben, können Sie diese Objekten in den Hintergrund setzen. Sie können Objekte in den Hintergrund setzen, z. B., indem Sie Objekte mit zwei Zuständen haben: *fake* und *loaded*. Angenommen, die *fake*-Objekte sind sehr einfach zu erstellen und enthalten wenig in der Bestandsliste anzuzeigende Informationen, wie z. B. Name und ID. In diesem Fall ist es möglich, immer *fake*-Objekte zurückzugeben, und wenn wirklich alle Informationen (das reale Objekt) erforderlich sind, kann die verwendende Einheit oder das Plug-In automatisch eine *load*-Methode aufrufen, um das reale Objekt abzurufen. Sie können sogar den Objektladevorgang konfigurieren, sodass dieser automatisch startet, nachdem *fake*-Objekte zurückgegeben werden. So können Sie den Aktionen der verwendenden Einheit zuvorkommen.

### Objekte klonen, um Parallelitätsprobleme zu vermeiden

Wenn Sie für Ihr Plug-In einen Cache verwenden, müssen Sie Objekte klonen. Das Verwenden eines Cache, der immer dieselbe Instanz eines Objekts an jede anfordernde Einheit zurückgibt, kann unerwünschte Folgen haben. Beispiel: Einheit A fordert Objekt O an, und die Einheit zeigt das Objekt in der Bestandsliste mit allen seinen Attributen an. Gleichzeitig fordert auch Einheit B das Objekt O an, und Einheit A führt einen Workflow aus, der mit dem Ändern der Attribute von Objekt O beginnt. Am Ende der Ausführung ruft der Workflow die *update*-Methode des Objekts auf, um das Objekt auf der Serverseite zu aktualisieren. Wenn Einheit A und Einheit B dieselbe Instanz von Objekt B erhalten, werden für Einheit A in der Bestandsliste

alle Änderungen angezeigt, die Einheit B ausführt, und das bereits vor dem Festschreiben der Änderungen auf der Serverseite. Wenn alles ordnungsgemäß abläuft, ist dies kein Problem. Wenn die Ausführung aber fehlschlägt, werden die Attribute von Objekt O nicht für Objekt A wiederhergestellt. In diesem Fall verwenden, wenn der Cache (die find-Vorgänge des Plug-Ins) einen Klon des Objekts anstelle immer wieder derselben Instanz zurückgibt, beide die Einheitsansichten und ändern ihre eigene Kopie, wodurch Parallelitätsprobleme zumindest innerhalb von Orchestrator vermieden werden.

### **Benachrichtigen anderer Benutzer über Änderungen**

Es können Probleme auftreten, wenn Sie einen Cache verwenden und gleichzeitig Objekte klonen. Das größte Problem ist, dass das Objekt, das Einheitsansichten verwendet, möglicherweise nicht die neueste Version ist, die für das Objekt verfügbar ist. Beispiel: Wenn eine Einheit die Bestandsliste anzeigt, werden die Objekte einmal geladen. Gleichzeitig zeigt aber die erste Einheit die Änderungen nicht an, wenn eine andere Einheit einige der Objekte ändert. Verwenden Sie zum Vermeiden dieses Problems die Methoden `PluginWatcher` und `IPluginPublisher` aus der Orchestrator-Plug-in-API, um andere Benutzer über die Änderungen zu benachrichtigen, damit andere Instanzen der Orchestrator-Clients die Änderungen sehen können. Dies gilt auch für eine eindeutige Instanz des Orchestrator-Clients, wenn Änderungen aus einem Objekt aus der Bestandsliste andere Objekte in der Bestandsliste beeinflussen und diese auch benachrichtigt werden müssen. Die Vorgänge, die häufig Benachrichtigungen verwenden, sind Hinzufügen, Aktualisieren und Löschen von Objekten, wenn diese Objekte, oder einige der Eigenschaften, in der Bestandsliste angezeigt werden.

### **Aktivieren der Suche nach Objekten jederzeit und überall**

Sie müssen die `find`-Methode der `IPluginFactory`-Schnittstelle implementieren, um Objekte nur über Typ und ID zu finden. Die `find`-Methode kann direkt nach dem Neustart von Orchestrator und dem Fortsetzen eines Workflows aufgerufen werden.

### **Simulieren eines Abfragedienstes, wenn Sie keinen haben**

Für den Orchestrator-Client ist es möglicherweise erforderlich, in bestimmten Fällen einige Objekte abzurufen oder sie nicht als Baumstruktur sondern beispielsweise als Liste oder Tabelle anzuzeigen. Dies bedeutet, dass Ihr Plug-In in der Lage sein muss, zu jeder Zeit Objektsätze abzufragen. Wenn die Drittanbietersoftware einen Abfragedienst anbietet, müssen Sie diesen Dienst anpassen und verwenden. Andernfalls sollte es Ihnen möglich sein, einen Abfragedienst zu simulieren, unabhängig von der höheren Komplexität oder niedrigeren Leistung der Lösung.

### **Suchmethoden dürfen keine Laufzeitausnahmen zurückgeben**

Die Methoden der `IPluginFactory`-Schnittstelle, die Suchvorgänge innerhalb des Plug-Ins implementieren, dürfen keine gesteuerten oder nicht gesteuerten Laufzeitausnahmen zurückgeben. Dies kann zu unerwarteten Fehlschlägen des Typs *Validierungsfehler* führen, wenn ein Workflow ausgeführt wird. Beispiel: Zwischen zwei Knoten eines Workflows wird die Methode `find` aufgerufen, wenn eine Ausgabe aus dem ersten Knoten eine Eingabe im zweiten Knoten ist. Zu diesem Zeitpunkt erhalten Sie, falls das Objekt aufgrund einer Laufzeitausnahme nicht gefunden wird, keine weiteren Informationen als einen *Validierungsfehler* im Orchestrator-Client. Danach hängt es davon ab, wie das Plug-In die Ausnahmen protokolliert, ob mehr oder weniger Informationen in den Protokolldateien enthalten sind.

### **Workflow-interne Ansätze**

Sie können einen Workflow implementieren, um langfristige Vorgänge zu überwachen, die das Orchestrator-Plug-In durchführt.

Sie können einen Workflow zum Überwachen von langfristig ausgeführten Vorgängen wie der Aufgabenüberwachung implementieren. Dieser Workflow kann auf Orchestrator-Auslösern und Wartereignissen basieren. Sie müssen beachten, dass ein Workflow, der beim Warten auf eine Aufgabe blockiert ist, fortgesetzt werden kann, sobald der Orchestrator-Server gestartet wird. Das Plug-In muss alle erforderlichen Informationen abrufen können, um den Überwachungsprozess ordnungsgemäß fortzusetzen.

Der Überwachungsworkflow oder die Aufgabe, die dieser intern verwenden kann, müssen einen Mechanismus bereitstellen, um die Abrufrate und eine mögliche Zeitüberschreitung anzugeben.



Der Debugging-Vorgang eines Codes zur Skripterstellung innerhalb eines Workflows ist nicht einfach, insbesondere wenn der Code keinen Java-Code abrufen. Aus diesem Grund ist es manchmal die einzige Option, die Protokollierungsmethoden zu verwenden, die von den standardmäßigen Orchestrator-Skriptobjekten bereitgestellt werden.

### **Workflows und Aktionen**

Sie können bestimmte bewährte Vorgehensweisen verwenden, um die Entwicklung und Verwendung von Workflows zu vereinfachen.

#### **Entwickeln Sie Workflows als Bausteine**

Ein Baustein kann ein einfacher Workflow sein, der wenige Eingabeparameter benötigt und eine einfache Ausgabe zurückgibt. Wenn Sie eine Vielzahl an Bausteinen haben, können Sie hochrangigere Workflows einfach erstellen und einen besseren Toolsatz zum Erstellen von komplexen Workflows bereitstellen.

#### **Erstellen von hochrangigen Workflows basierend auf kleineren Komponenten**

Wenn Sie einen komplexen Workflow mit mehreren Eingaben und internen Schritten entwickeln müssen, können Sie diesen in kleinere und einfachere zusammengesetzte Workflows und Aktionen aufteilen.

#### **Erstellen Sie Aktionen, wo möglich**

Sie können Aktionen erstellen, um zusätzliche Flexibilität beim Entwickeln von Workflows zu erhalten.

- Zum einfachen Erstellen komplexer Objekte oder Parameter für Skripterstellungsmethoden
- Zum Vermeiden von ständigen Wiederholungen in allgemeinen Codeabschnitten
- Zum Ausführen von Benutzeroberflächen-Validierungen

#### **Workflows sollten nach Möglichkeit Aktionen aufrufen**

Aktionen können direkt als Knoten innerhalb des Workflowschemas aufgerufen werden. Dadurch können Workflowschemas einfach gehalten werden, da Sie keine Skript-Codeabschnitte hinzufügen müssen, um eine einzelne Aktion aufzurufen.

#### **Füllen Sie die erwarteten Informationen aus**

Geben Sie Informationen zu allen Elementen eines Workflows oder einer Aktion an.

- Geben Sie eine Beschreibung des Workflows oder der Aktion an.
- Geben Sie eine Beschreibung der Eingabeparameter an.
- Geben Sie eine Beschreibung der Ausgaben an.
- Geben Sie eine Beschreibung der Attribute für die Workflows an.

#### **Halten Sie die Versionsinformationen auf dem neuesten Stand**

Wenn Sie Versionen von Plug-Ins erstellen, fügen Sie aussagekräftige Kommentare mit Informationen zu wichtigen Updates am Plug-In, wichtigen Implementierungsdetails usw. hinzu.

#### **Workflowpräsentation**

Wenn Sie die Präsentation eines Workflows erstellen, müssen Sie bestimmte Strukturen und Regeln anwenden.

Verwenden Sie die folgenden Eigenschaften für Workfloweingaben in der Workflowpräsentation.

**Tabelle 1-50.** Eigenschaften für Workfloweingaben

Eigenschaften	Nutzung
Show in Inventory	Verwenden Sie diese Eigenschaft, um dem Benutzer beim Ausführen eines Workflows aus der Bestandslistenansicht zu helfen.
Specify a root object to be shown in the chooser	Verwenden Sie diese Eigenschaft, um dem Benutzer beim Auswählen von Eingaben zu helfen. Wenn das Root-Objekt in der Präsentation aktualisiert werden kann, ein Attribut ist oder von einer Objektmethode abgerufen wird, müssen Sie eine angemessene Aktion erstellen oder festlegen, um das Objekt in der Präsentation zu aktualisieren.
Maximum string length	Verwenden Sie diese Eigenschaft für lange Zeichenfolgen wie Namen, Beschreibungen, Dateipfade usw.
Minimum string length	Verwenden Sie diese Eigenschaft, um leere Zeichenfolgen aus den Testtools zu vermeiden.
Custom validation	Implementieren Sie komplexe Validierungen mit Aktionen.

Organisieren Sie die Eingaben mit Schritten und Anzeigegruppe. Diese Organisation unterstützt den Benutzer dabei, alle Eingabeparameter eines Workflows zu bestimmen und zu unterscheiden.

## Empfehlungen zur Entwicklung von Orchestrator-Plug-Ins

Indem Sie sich beim Entwickeln der verschiedenen Komponenten für Ihre Orchestrator-Plug-Ins an bestimmte Grundsätze halten, können Sie die Qualität der Plug-Ins verbessern.

**Tabelle 1-51.** Nützliche Verfahren bei der Plug-In-Implementierung

Komponente	Element	Beschreibung
Allgemein	Zugriff auf die Drittanbieter-API	Nach Möglichkeit sollten Plug-Ins vereinfachte Methoden bereitstellen, um auf die Drittanbieter-API zuzugreifen.
	Schnittstelle	Plug-Ins sollten eine kohärente und standardisierte Schnittstelle für Benutzer bereitstellen, auch wenn die API dies nicht tut.
Aktion	Skriptobjekte	Erstellen Sie Aktionen für sämtliche Erstellungs-, Änderungs- und Lösch- und sonstigen Methoden, die für ein Skriptobjekt verfügbar sind.
	Beschreibung	In der Beschreibung einer Aktion sollte erklärt werden, was die Aktion bewirkt, und nicht, wie sie funktioniert.
	Skripterstellung	Wenn Sie Skripterstellung verwenden, um die Eigenschaften oder Methoden eines Objekts abzurufen, können Sie überprüfen, ob sich der Objektwert von <code>null</code> oder <code>undefined</code> unterscheidet.
	Veraltung	Wenn eine Aktion veraltet ist, sollte die <code>comment</code> - oder <code>throw</code> -Anweisung die Ersatzaktion enthalten oder die Aktion selbst sollte eine neue Ersatzaktion aufrufen, damit Lösungen, die auf der veralteten Version der Aktion basieren, nicht fehlschlagen.
Workflow	Benutzeroberflächenvorgänge in der Orchestrator-gesteuerten Technologie	Erstellen Sie einen Workflow für jeden Vorgang, der in der Benutzeroberfläche der Orchestrator-gesteuerten Technologie verfügbar ist.
	Beschreibung	In der Beschreibung eines Workflows sollte erklärt werden, was der Workflow bewirkt, und nicht, wie er funktioniert.
	Präsentationseigenschaft <code>mandatory input</code>	Für alle obligatorischen Workfloweingaben müssen Sie die Eigenschaft <code>mandatory input</code> festlegen.

**Tabelle 1-51.** Nützliche Verfahren bei der Plug-In-Implementierung (Fortsetzung)

Komponente	Element	Beschreibung
	Präsentationseigenschaft default value	Wenn Sie einen Workflow entwickeln, der eine Einheit konfiguriert, sollten die Standardkonfigurationswerte für diese Einheit in die Workflowpräsentation geladen werden. Wenn Sie beispielsweise einen Workflow mit dem Namen „Hostkonfiguration“ entwickeln, müssen die Standardwerte der Hostkonfiguration in die Präsentation des Workflows geladen werden.
	Präsentationseigenschaft Show in inventory	Sie müssen die Eigenschaft Show in inventory so einstellen, dass Kontextworkflows für Bestandslistenobjekte vorhanden sind.
	Präsentationseigenschaft specify a root parameter	Verwenden Sie diese Eigenschaft in Workflows, wenn es nicht notwendig ist, die Bestandsliste vom Stamm der Baumstruktur aus zu durchsuchen.
	Workflowvalidierung	Sie müssen Workflows validieren und sämtliche Fehler korrigieren.
	Objekterstellung	Alle Workflows, die ein neues Objekt erstellen, sollten das neue Objekt als Ausgabeparameter zurückgeben.
	Veraltung	Wenn ein Workflow veraltet ist, sollte die comment- oder throw-Anweisung den Ersatzworkflow enthalten oder der veraltete Workflow sollte einen neuen Ersatzworkflow aufrufen, damit Lösungen, die auf vorherigen Versionen des Workflows basieren, nicht fehlschlagen.
Bestandsliste	Trennen der Hostverbindung	Wenn Ihre Bestandsliste eine Verbindung zu einem Host enthält und dieser Host nicht mehr verfügbar ist, geben Sie an, dass die Hostverbindung getrennt wurde. Dazu können Sie entweder das Stammobjekt umbenennen, indem Sie – disconnected anhängen, oder die Baumstruktur der Objekt unterhalb dieses Objekts entfernen, was auch das vCloud Director-Plug-In tut.
	Eigenschaft Select value as list	Ein Bestandslistenobjekt muss als treeview oder list auswählbar sein.
	Hostmanager	Wenn das Plug-In ein host-Objekt für das Zielsystem implementiert, sollte ein übergeordnetes hostmanager-Stammobjekt mit Eigenschaften zum Hinzufügen, Entfernen oder Bearbeiten von Hosteseigenschaften vorhanden sein.
	Abrufen oder Aktualisieren von Objekten	Wenn ein Abfragedienst in der Orchestrator-gesteuerten Technologie ausgeführt wird, sollten Sie diesen zum Abrufen mehrerer Objekte verwenden.
	Erkennung untergeordneter Objekte	Wenn Sie untergeordnete Objekte separat abrufen müssen, muss der Abrufprozess mehrere Threads umfassen und darf nicht bei einem einzelnen Fehler blockieren.
	Orchestrator-Objektänderung	Alle Workflows, die den Zustand eines Elements in der Bestandsliste ändern können, müssen die Bestandsliste aktualisieren, damit die Synchronisierung der Objekte nicht verloren geht.

**Tabelle 1-51.** Nützliche Verfahren bei der Plug-In-Implementierung (Fortsetzung)

Komponente	Element	Beschreibung
Skriptobjekt	Externe Objektänderung	Sie können mithilfe eines Benachrichtigungsmechanismus auf Änderungen in der Orchestrator-gesteuerten Technologie hinweisen, die aus außerhalb von Orchestrator ausgeführten Vorgängen resultieren. Falls solche Vorgänge dazu führen, dass Objekte aus der Orchestrator-gesteuerten Technologie entfernt werden, müssen Sie die Bestandsliste entsprechend aktualisieren, um Fehler oder Datenverlust zu vermeiden. Wenn beispielsweise eine virtuelle Maschine aus vCenter Server gelöscht wird, aktualisiert das vCenter Server-Plug-In die Bestandsliste, um das Objekt der entfernten virtuellen Maschine zu entfernen.
	Finder-Objekt	Finder-Objekte sollten über Eigenschaften verfügen, mit deren Hilfe Objekte unterschieden werden können. In der Regel sind dies die Eigenschaften, die in der Benutzeroberfläche angezeigt werden.
	Implementierung	Die Methode <code>equals</code> muss implementiert werden, um sicherzustellen, dass der Vorgang <code>==</code> auf dasselbe Objekt angewendet wird, da das Objekt möglicherweise über zwei Instanzen verfügt.
	Plug-In-Objekteigenschaften	Für Objekte mit übergeordneten Objekten sollte eine <code>parent</code> -Eigenschaft implementiert werden.
	Plug-In-Objekteigenschaften	Für Objekte mit untergeordneten Objekten sollten GET-Methoden implementiert werden, die Arrays von untergeordneten Objekten zurückgeben.
	Bestandslistenobjekte	Bestandslistenobjekte sollten sich mit <code>Server.find</code> suchen lassen.  Alle Bestandslistenobjekte sollten serialisierbar sein, damit sie in einem Workflow als Eingabe- oder Ausgabeattribute verwendet werden können.
	Konstruktor und Methoden	In den meisten Fällen sollten skriptfähige Objekte entweder über einen Konstruktor verfügen oder von anderen Objektattributen bzw. Methoden zurückgegeben werden.
	Objekt-ID	Objekte mit einer von einem externen System ausgegebenen ID sollten eine interne ID verwenden, um sicherzustellen, dass keine IDs dupliziert werden, wenn Sie mehrere Server mit Orchestrator steuern.
	Suchen nach Objekten	In <code>search</code> - oder <code>find</code> -Methoden sollte ein Filter implementiert werden, damit der angegebene Name bzw. die angegebene ID gefunden werden kann, statt einfach alle Objekte zu finden. Der Orchestrator-Server verfügt beispielsweise über die Methode <code>Server.FindById</code> , mit der ein Plug-In-Objekt anhand seiner ID gefunden werden kann. Dazu muss die Methode für jedes auffindbare Objekt im Plug-In implementiert werden.
	Auslöser	Nach Möglichkeit sollten Auslöser für Objekte, die sich ändern, verfügbar sein, damit Orchestrator bei verschiedenen Ereignissen Richtlinien auslösen lassen kann. Um beispielsweise zu ermitteln, wann eine neue virtuelle Maschine hinzugefügt, eingeschaltet, ausgeschaltet usw. wird, kann Orchestrator einen Auslöser oder ein Ereignis im vCenter-Plug-In für das Datacenter-Objekt überwachen.

**Tabelle 1-51.** Nützliche Verfahren bei der Plug-In-Implementierung (Fortsetzung)

Komponente	Element	Beschreibung
	Objekteigenschaften	Objekte in anderen Plug-Ins sollten über Eigenschaften verfügen, die eine einfache Konvertierung von einem Plug-In-Objekt in ein anderes ermöglichen. Objekte von virtuellen Maschinen müssen beispielsweise über eine <code>moref</code> (Referenz auf verwaltetes Objekt) verfügen.
	Sitzungsmanager	Wenn Sie eine Verbindung zu einem Remoteserver herstellen, der eine andere Sitzung aufweisen kann, sollten im Plug-In eine gemeinsam genutzte Sitzung und eine Sitzung pro Benutzer implementiert werden.
Auslöser	Auslöser	Es sollte möglich sein, alle langen Vorgänge und blockierenden Methoden nach einer zurückgegebenen Aufgabe zu starten und bei Abschluss ein Auslöseereignis zu generieren.
Enumerationen	Enumerationen	Enumerationen für einen bestimmten Typ sollten über ein Bestandslistenobjekt verfügen, das ein Auswählen aus den verschiedenen Werten in der Enumeration ermöglicht.
Protokollierung	Protokolle	Für Methoden sollten verschiedene Protokollierungsebenen implementiert werden.
Versionierung	Plug-In-Version	Die Plug-In-Version sollte Standards entsprechen und zusammen mit dem Plug-In-Update aktualisiert werden.
API-Dokumentation	Methoden	Methoden, die in der API-Dokumentation beschrieben werden, sollten nie die Ausnahme <code>no xyz method / property</code> für ein Objekt zurückgeben. Stattdessen sollten Methoden <code>null</code> zurückgeben, wenn keine Eigenschaften verfügbar sind, und mit Details dokumentiert werden, wenn diese Eigenschaften nicht verfügbar sind.
	<code>vso.xml</code>	Alle Objekte, Methoden und Eigenschaften müssen in der Datei <code>vso.xml</code> dokumentiert werden.

## Dokumentierung von Plug-In-Benutzeroberflächen-Zeichenfolgen und -APIs

Wenn Sie Benutzeroberflächen-Zeichenfolgen für Orchestrator-Plug-Ins und die zugehörige API-Dokumentation schreiben, folgen Sie den allgemein anerkannten Stil- und Formatregeln.

### Allgemeine Empfehlungen

- Verwenden Sie die offiziellen Namen für VMware-Produkte, die im Plug-In involviert sind. Verwenden Sie beispielsweise offizielle Namen für die folgenden Produkte sowie VMware-Terminologie.

Korrektter Begriff	Nicht verwenden
vCenter Server	VC oder vCenter
vCloud Director	vCloud

- Beenden Sie alle Workflowbeschreibungen mit einem Punkt. Beispielsweise ist `Creates a new Organization.` eine Workflowbeschreibung.
- Verwenden Sie einen Texteditor mit einer Rechtschreibprüfung, um die Beschreibungen zu schreiben, und verschieben Sie sie dann in das Plug-In.
- Achten Sie darauf, dass der Name des Plug-Ins genau mit dem genehmigten Produktnamen des Drittanbieters, mit dem es verknüpft ist, übereinstimmt.

### Workflows und Aktionen

- Verfassen Sie informative Beschreibungen. Ein oder zwei Sätze genügen für die meisten Aktionen und Workflows.

- Hochrangige Workflows beinhalten möglicherweise umfangreichere Beschreibungen und Kommentare.
- Beginnen Sie Beschreibungen mit einem Verb, z. B. `Creates...`. Verwenden Sie keine auf sich selbst verweisende Sprache wie beispielsweise `This workflow creates`.
- Setzen Sie einen Punkt am Ende der Beschreibungen, die einen ganzen Satz darstellen.
- Beschreiben Sie, was ein Workflow oder eine Aktion macht, anstatt wie er/sie implementiert wird.
- Workflows und Aktionen sind normalerweise in Ordnern und Paketen enthalten. Beschreiben Sie auch diese Ordner und Pakete kurz. Beispielsweise kann ein Workflowordner eine Beschreibung wie die folgende haben: `Set of workflows related to vApp Template management`.

### Parameter von Workflows und Aktionen

- Beginnen Sie Beschreibungen für Workflows und Aktionen mit einer beschreibenden Nominalphrase, z. B. `Name of`. Verwenden Sie keine Phrasen wie `It's the name of`.
- Setzen Sie keinen Punkt am Ende von Beschreibungen für Parameter und Aktionen. Es sind keine vollständigen Sätze.
- Eingabeparameter von Workflows müssen eine Bezeichnung mit entsprechenden Namen in der Präsentationsansicht angeben. In vielen Fällen können Sie verwandte Eingaben in einer Anzeigegruppe zusammenfassen. Anstatt zwei Eingaben mit den Bezeichnungen „Name der Organisation“ und „Vollständiger Name der Organisation“ zu erstellen, können Sie beispielsweise eine Anzeigegruppe mit der Bezeichnung „Organisation“ erstellen und die Eingaben „Name“ und „Vollständiger Name“ in die Gruppe „Organisation“ platzieren.
- Fügen Sie für Schritte und Anzeigegruppen Beschreibungen oder Kommentare hinzu, die auch in der Workflowpräsentation angezeigt werden.

### Plug-In-API

- Die Dokumentation der API verweist auf die gesamte Dokumentation in der Datei `vso.xml` und in den Java-Quelldateien.
- Verwenden Sie für die Datei `vso.xml` dieselben Regeln für die Beschreibungen der Finder-Objekte und Skripterstellung-Objekte mit ihren Methoden, die Sie für Workflows und Aktionen verwenden. Beschreibungen von Objektattributen und Methodenparametern verwenden dieselben Regeln wie die Workflow- und Aktionsparameter.
- Vermeiden Sie Sonderzeichen in der Datei `vso.xml` und beziehen Sie die Beschreibungen innerhalb eines `<![CDATA[insert your description here!]]>`-Tags mit ein.
- Verwenden Sie den Javadoc-Standardstil für die Java-Quelldateien.

## Abrufen der Eingabeparameter von Benutzern beim Start eines Workflows

Wenn ein Workflow Eingabeparameter erfordert, öffnet er ein Dialogfeld, in dem Benutzer die erforderlichen Eingabeparameterwerte zum Ausführen eingeben. Sie können den Inhalt und das Layout oder die Präsentation dieses Dialogfelds auf der Registerkarte **Präsentation** im Workfloweditor organisieren.

Die Organisation von Parametern auf der Registerkarte **Präsentation** wird im Dialogfeld der Eingabeparameter dargestellt, wenn der Workflow ausgeführt wird.

Die Registerkarte **Präsentation** ermöglicht es Ihnen auch, Beschreibungen der Eingabeparameter hinzuzufügen, damit Benutzer bei der Eingabe von Parametern entsprechend informiert werden können. Sie können auch Eigenschaften und Integritätsregeln für Parameter auf der Registerkarte **Präsentation** eingeben, um die Parameter zu begrenzen, die von den Benutzern bereitgestellt werden. Wenn die Parameter, die der Benutzer eingibt, die Integritätsregeln nicht einhält, die Sie auf der Registerkarte **Präsentation** festgelegt haben, wird der Workflow nicht ausgeführt.

- [Erstellen des Dialogfelds „Eingabeparameter“ auf der Registerkarte „Präsentation“](#) auf Seite 103  
Sie definieren das Layout des Dialogfelds, in dem die Benutzer Eingabeparameter eingeben, wenn Sie einen Workflow in der Registerkarte **Präsentation** im Workfloweditor ausführen.
- [Setzen von Parametereigenschaften](#) auf Seite 104  
Mit Orchestrator können Sie die Eigenschaften definieren, die Werte von Eingabeparametern der Benutzer beim Ausführen von Workflows qualifizieren. Die von Ihnen definierten Parametereigenschaften setzen Grenzwerte für Typen und Werte der von Benutzern eingegebenen Parameter.

## Erstellen des Dialogfelds „Eingabeparameter“ auf der Registerkarte „Präsentation“

Sie definieren das Layout des Dialogfelds, in dem die Benutzer Eingabeparameter eingeben, wenn Sie einen Workflow in der Registerkarte **Präsentation** im Workfloweditor ausführen.

Die Registerkarte **Präsentation** ermöglicht es Ihnen, Eingabeparameter in Kategorien zusammenzufassen und die Reihenfolge zu definieren, in der diese Kategorien im Dialogfeld für die Eingabeparameter angezeigt werden.

### Beschreibungen der Präsentation

Sie können eine zugehörige Beschreibung für jeden Parameter und jede Parametergruppe hinzufügen, die im Dialogfeld für die Eingabeparameter angezeigt werden. Die Beschreibungen informieren die Benutzer über die richtigen Eingaben für die Eingabeparameter. Sie können das Layout des Beschreibungstexts hervorheben, indem Sie HTML-Formatierung verwenden.

### Definieren von Eingabeschritten für die Präsentation

Standardmäßig enthält das Dialogfeld für die Eingabeparameter alle erforderlichen Eingabeparameter in einer einzelnen Liste. Zur Unterstützung der Benutzer bei der Eingabe von Parametern können Sie auf der Registerkarte „Präsentation“ Knoten definieren, die als Eingabeschritte bezeichnet werden. Eingabeschritte gruppieren Eingabeparameter ähnlicher Art. Die Eingabeparameter unter einem Eingabeschritt erscheinen in einem eigenen Abschnitt im Dialogfeld für die Eingabeparameter, wenn der Workflow ausgeführt wird.

### Definieren von Anzeigegruppen für die Präsentation

Jeder Eingabeschritt kann eigene Knoten haben, die als Anzeigegruppen bezeichnet werden. Die Anzeigegruppen definieren die Reihenfolge, in der Textfelder für die Eingabe von Parametern in dem jeweiligen Abschnitt des Dialogfelds für die Eingabeparameter erscheinen. Sie können Anzeigegruppen unabhängig von Eingabeschritten definieren.

### Erstellen der Präsentation des Dialogfelds für Eingabeparameter

Sie erstellen die Präsentation des Dialogfelds, in dem die Benutzer Eingabeparameter eingeben, wenn sie einen Workflow in der Registerkarte **Präsentation** im Workfloweditor ausführen.

#### Voraussetzungen

- Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.
- Vergewissern Sie sich, dass der Workflow eine definierte Liste von Eingabeparametern hat.

#### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Präsentation**.

Standardmäßig erscheinen alle Parameter des Workflows unter dem Hauptknoten **Präsentation** in der Reihenfolge, in der Sie sie erstellen.

- 2 Klicken Sie mit der rechten Maustaste auf den Knoten **Präsentation** und wählen Sie **Neuen Schritt erstellen**.

Der Knoten **Neuer Schritt** erscheint unter dem Knoten **Präsentation**.

- 3 Geben Sie einen geeigneten Namen für den Schritt ein und drücken Sie die Eingabetaste.

Dieser Name erscheint als Abschnittsüberschrift im Dialogfeld der Eingabeparameter, wenn der Workflow ausgeführt wird.

- 4 Klicken Sie auf den Eingabeschritt und fügen Sie eine Beschreibung auf der Registerkarte **Allgemein** in der unteren Hälfte der Registerkarte **Präsentation** ein.

Diese Beschreibung erscheint im Dialogfeld der Eingabeparameter und stellt Benutzern Informationen zur Eingabe der richtigen Eingabeparameter bereit. Sie können das Layout des Beschreibungstexts hervorheben, indem Sie HTML-Formatierung verwenden.

- 5 Klicken Sie mit der rechten Maustaste auf den von Ihnen erstellten Eingabeschritt und wählen Sie **Anzeigegruppe erstellen**.

Der Knoten **Neue Gruppe** erscheint unter dem Eingabeschritt-Knoten.

- 6 Geben Sie einen geeigneten Namen für die Anzeigegruppe ein und drücken Sie die Eingabetaste.

Dieser Name erscheint als Überschrift des Unterabschnitts im Dialogfeld der Eingabeparameter, wenn der Workflow ausgeführt wird.

- 7 Klicken Sie auf die Anzeigegruppe und fügen Sie eine Beschreibung auf der Registerkarte **Allgemein** in der unteren Hälfte der Registerkarte **Präsentation** ein.

Diese Beschreibung erscheint im Dialogfeld der Eingabeparameter. Sie können das Layout des Beschreibungstexts hervorheben, indem Sie HTML-Formatierung verwenden. Sie können mithilfe einer OGNL-Anweisung einer Gruppenbeschreibung einen Parameterwert hinzufügen, beispielsweise `${#param}`.

- 8 Wiederholen Sie die vorstehenden Schritte, bis Sie alle Eingabeschritte und Anzeigegruppen erstellt haben, die im Dialogfeld der Eingabeparameter erscheinen sollen, wenn der Workflow ausgeführt wird.

- 9 Ziehen Sie Parameter unter dem Knoten **Präsentation** auf die Schritte und Gruppen Ihrer Wahl.

Sie haben das Layout des Dialogfelds für die Eingabeparameter erstellt, über die die Benutzer Eingabeparameterwerte eingeben, wenn der Workflow ausgeführt wird.

#### Weiter

Sie müssen die Parametereigenschaften festlegen.

## Setzen von Parametereigenschaften

Mit Orchestrator können Sie die Eigenschaften definieren, die Werte von Eingabeparametern der Benutzer beim Ausführen von Workflows qualifizieren. Die von Ihnen definierten Parametereigenschaften setzen Grenzwerte für Typen und Werte der von Benutzern eingegebenen Parameter.

Jeder Parameter kann mehrere Eigenschaften haben. Sie definieren die Eigenschaften eines Eingabeparameter auf der Registerkarte **Eigenschaften** für einen bestimmten Parameter auf der Registerkarte **Präsentation**.

Parametereigenschaften validieren die Eingabeparameter und modifizieren die Art, wie Textfelder im Dialogfeld für die Eingabeparameter angezeigt werden. Einige Parametereigenschaften können Abhängigkeiten zwischen Parametern schaffen.

### Eigenschaftswert für statische und dynamische Parameter

Ein Eigenschaftswert für einen Parameter kann statisch oder dynamisch sein. Statische Eigenschaftswerte bleiben konstant. Wenn Sie einen Eigenschaftswert auf „statisch“ setzen, legen Sie den Wert der Eigenschaft fest oder wählen ihn aus einer Liste aus, die der Workfloweditor gemäß dem Parametertyp erstellt.



Dynamische Eigenschaftswerte hängen vom Wert eines anderen Parameters oder eines Attributs ab. Sie definieren die Funktionen, durch die dynamische Eigenschaften Werte erhalten, indem Sie einen Ausdruck in der Object Graph Navigation Language (OGNL) verwenden. Wenn der Eigenschaftswert eines dynamischen Parameters vom Eigenschaftswert eines anderen Parameters abhängt und sich der Eigenschaftswert des anderen Parameters ändert, berechnet der OGNL-Ausdruck den dynamischen Eigenschaftswert erneut und ändert ihn.

## Setzen von Parametereigenschaften



Wenn ein Workflow startet, validiert er die Werte von Eingabeparametern von Benutzern anhand der von Ihnen eingegebenen Parametereigenschaften.

### Voraussetzungen

- Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.
- Vergewissern Sie sich, dass der Workflow eine definierte Liste von Eingabeparametern hat.

### Vorgehensweise



- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Präsentation**.
- 2 Klicken Sie auf einen Parameter auf der Registerkarte **Präsentation**.  
Die Registerkarten **Allgemein** und **Eigenschaften** erscheinen unten auf der Registerkarte **Präsentation**.
- 3 Klicken Sie auf die Registerkarte **Eigenschaften** des Parameters.
- 4 Klicken Sie mit der rechten Maustaste in die Registerkarte **Eigenschaften** und wählen Sie **Eigenschaft hinzufügen**.  
Ein Dialogfeld wird geöffnet, in dem eine Liste der möglichen Eigenschaften für einen Parameter des ausgewählten Typs angezeigt wird.
- 5 Wählen Sie eine Eigenschaft aus der im Dialogfeld angezeigten Liste und klicken Sie auf **OK**.  
Die Eigenschaft erscheint auf der Registerkarte **Eigenschaften**.
- 6 Setzen Sie unter **Wert** den Eigenschaftswert entweder auf statisch oder auf dynamisch, indem Sie das entsprechende Symbol aus dem Dropdown-Menü auswählen.

Option	Beschreibung
	Eigenschaft „statisch“
	Eigenschaft „dynamisch“

- 7 Wenn Sie den Eigenschaftswert auf statisch setzen, wählen Sie einen Eigenschaftswert entsprechend dem Parametertyp aus, für den Sie die Eigenschaften festlegen.

- 8 Wenn Sie den Eigenschaftswert auf dynamisch setzen, definieren Sie mithilfe eines OGNL-Ausdrucks die Funktion, mit der der Eigenschaftswert des Parameters ermittelt werden soll.

Der Workfloweditor bietet Unterstützung beim Schreiben des OGNL-Ausdrucks.

- a Klicken Sie auf das Symbol , um eine Liste aller Attribute und Parameter zu erhalten, die von dem Workflow definiert wurden, für den dieser Ausdruck Aufrufe durchführt.
- b Klicken Sie auf das Symbol , für den Abruf einer Liste aller Aktionen in der Orchestrator-API, die einen Ausgabeparameter des Typs zurückgeben, für den Sie die Eigenschaften definieren.

Wenn Sie auf Elemente in den angebotenen Listen von Parametern und Aktionen klicken, werden diese dem OGNL-Ausdruck hinzugefügt.

- 9 Klicken Sie unten im Workfloweditor auf **Speichern**.

Sie haben die Eigenschaften der Eingabeparameter des Workflows definiert.

### Weiter

Validieren Sie den Workflow und beheben Sie etwaige Fehler.

## Eigenschaften der Eingabeparameter des Workflows

Sie können Parametereigenschaften festlegen, um die Eingabeparameter zu beschränken, die Benutzer beim Ausführen von Workflows bereitstellen.

Verschiedene Parametertypen können verschiedene Eigenschaften haben.

**Tabelle 1-52.** Eigenschaften der Eingabeparameter des Workflows

Parametereigenschaft	Parametertyp	Beschreibung
Maximale Zeichenfolgenlänge	String	Legt die maximale Länge für den Parameter fest.
Minimale Zeichenfolgenlänge	String	Legt die Mindestlänge für den Parameter fest.
Übereinstimmung mit regulärem Ausdruck	String	Validiert die Eingabe mithilfe eines regulären Ausdrucks.
Maximaler Zahlenwert	Zahl	Legt einen maximalen Wert für den Parameter fest.
Minimaler Zahlenwert	Zahl	Legt einen Mindestwert für den Parameter fest.
Zahlenformat	Zahl	Formatiert die Eingabe für den Parameter.
Erforderliche Eingabe	Alle einfachen Typen	Macht den Parameter zu einer erforderlichen Eingabe.
Vordefinierte Antworten	Alle einfachen Typen	Erstellt eine Vordefinition einer Liste möglicher Werte für die Eigenschaft als Array von einfachen Typen. Sie definieren entweder das Array manuell oder die Eigenschaft ruft eine Aktion auf, die ein Array von Objekten des geeigneten Typs zurückgibt.
Vordefinierte Liste von Elementen	Jeder einfache oder komplexe Typ	Erstellt eine Vordefinition einer Liste möglicher Werte für die Eigenschaft als Array von einfachen oder komplexen Typen. Ruft eine Aktion auf, die ein Array von Objekten des geeigneten Typs zurückgibt.

**Tabelle 1-52.** Eigenschaften der Eingabeparameter des Workflows (Fortsetzung)

Parametereigenschaft	Parametertyp	Beschreibung
Parameterangabe anzeigen	Jeder einfache oder komplexe Typ	Zeigt je nach Wert eines vorangehenden booleschen Parameters im Dialogfeld der Präsentation ein Textfeld für Parameter an oder blendet es aus.
Parametereingabe ausblenden	Jeder einfache oder komplexe Typ	Ähnlich wie <b>Parametereingabe anzeigen</b> , übernimmt aber den negativen Wert eines vorhergehenden booleschen Parameters.
Übereinstimmender Ausdruck	Jeder Parametertyp, der von einem Plug-In kommt.	Der Eingabeparameter stimmt mit einem gegebenen Ausdruck überein.
In Bestand anzeigen	Jeder Parametertyp, der von einem Plug-In kommt.	Wenn der Parameter gesetzt ist, können Sie den vorliegenden Workflow mit jedem Objekt dieses Typs ausführen, indem Sie mit der rechten Maustaste in der Bestandsliste darauf klicken und <b>Workflow ausführen</b> wählen.
Geben Sie ein Stammobjekt an, das in der Auswahlfunktion angezeigt wird. Das Stammobjekt wird von einem Parameter oder einem Attribut geliefert.	Jeder Parametertyp, der von einem Plug-In kommt.	Gibt das Stammobjekt an, wenn die Auswahlfunktion für diesen Parameter die Auswahl aus einer hierarchischen Liste ist.
Auswählen als	Jeder Parametertyp, der von einem Plug-In kommt.	Verwenden Sie eine Liste oder eine Auswahl aus einer hierarchischen Liste für die Parameterauswahl.
Standardwert	Jeder einfache oder komplexe Typ	Standardwert für diesen Parameter.
Benutzerdefinierte Validierung	Mit OGNL skriptfähige Validierung	Wenn der OGNL-Ausdruck eine Zeichenfolge zurückgibt, zeigt die Validierung diese Zeichenfolge als Text des Fehlerergebnisses.
Datenbindung	Jeder einfache oder komplexe Typ	Bindet an eine Eigenschaft, die Sie bereits in einem anderen Parameter definiert haben.
Nur autorisierte	Jeder Parametertyp, der von einem Plug-In kommt.	Nur autorisierte Benutzer dürfen auf diesen Parameter zugreifen.
Mehrzeilige Texteingabe	Jeder einfache oder komplexe Typ	Ermöglicht Benutzern die Eingabe von mehrzeiligem Text im Dialogfeld der Eingabeparameter.

## Vordefinierte Konstantenwerte für OGNL-Ausdrücke

Sie können vordefinierte Konstanten verwenden, wenn Sie OGNL-Ausdrücke erstellen, um dynamische Werte für Parametereigenschaften zu erhalten.

Orchestrator definiert die folgenden Konstanten für die Verwendung in OGNL-Ausdrücken.

**Tabelle 1-53.** Vordefinierte OGNL-Konstantenwerte

Konstantenwert	Beschreibung
<code>\${#__current}</code>	Aktueller Wert der angepassten Validierungseigenschaft oder übereinstimmende Ausdruckseigenschaft
<code>\${#__username}</code>	Benutzername des Benutzers, der die Aufgabe gestartet hat.
<code>\${#__userdisplayname}</code>	Anzeigenname des Benutzers, der die Aufgabe gestartet hat.

**Tabelle 1-53.** Vordefinierte OGNL-Konstantenwerte (Fortsetzung)

Konstantenwert	Beschreibung
<code>\${#__serverurl}</code>	URL mit der IP-Adresse des Servers, von dem aus der Benutzer den Workflow startet. Die URL besteht aus der Server-IP-Adresse und einem Suchport: <code>{ServerIP}:{lookupPort}</code>
<code>\${#__datetime}</code>	Aktuelles Datum und aktuelle Uhrzeit
<code>\${#__date}</code>	Aktuelles Datum mit Uhrzeit 00:00:00
<code>\${#__timezone}</code>	Aktuelle Zeitzone

## (Optional) Anfordern von Benutzerinteraktionen während einer Workflowausführung

Manchmal benötigt ein Workflow während seiner Ausführung zusätzliche Eingabeparameter aus einer externen Quelle. Diese Eingabeparameter können aus einer anderen Anwendung oder einem anderen Workflow stammen, oder der Benutzer kann sie direkt angeben.

Wenn beispielsweise ein bestimmtes Ereignis auftritt, während ein Workflow ausgeführt wird, kann der Workflow eine Benutzerinteraktion anfordern, um über das weitere Vorgehen zu entscheiden. Der Workflow wird erst fortgesetzt, wenn entweder der Benutzer auf die Anforderung nach Informationen geantwortet hat oder die Wartezeit ein mögliches Zeitüberschreitungslimit überschreitet. Wenn die Wartezeit eine Zeitüberschreitung auslöst, gibt der Workflow eine Ausnahme zurück.

Die Standardattribute für Benutzerinteraktionen sind `security.group` und `timeout.date`. Wenn Sie das Attribut `security.group` auf eine bestimmte LDAP-Benutzergruppe festlegen, beschränken Sie die Berechtigung, auf die Anforderung zur Benutzerinteraktion zu antworten, auf diese Benutzergruppe.

Wenn Sie das Attribut `timeout.date` festlegen, stellen Sie mit Uhrzeit und Datum ein, bis wann der Workflow auf die Informationen vom Benutzer wartet. Sie können ein absolutes Datum festlegen oder auch ein Workflowelement mit Skript erstellen, um eine Zeit relativ zur aktuellen Zeit zu berechnen.

### Vorgehensweise

- 1 [Hinzufügen einer Benutzerinteraktion zu einem Workflow](#) auf Seite 109  
Sie können Eingabeparameter von Benutzern während einer Workflowausführung anfordern, indem Sie dem Workflow das Schemaelement **Benutzerinteraktion** hinzufügen. Wenn bei einem Workflow das Element **Benutzerinteraktion** festgestellt wird, wird die Ausführung angehalten und gewartet, bis der Benutzer die erforderlichen Daten angibt.
- 2 [Festlegen des Attributs „security.group“](#) auf Seite 109  
Das Attribut `security.group` eines Benutzerinteraktionselements legt fest, welche Benutzer oder Benutzergruppen die Berechtigung zum Antworten auf eine Benutzerinteraktion haben.
- 3 [Festlegen des Attributs „timeout.date“ auf ein absolutes Datum](#) auf Seite 110  
Mithilfe des Attributs `timeout.date` für eine Benutzerinteraktion legen Sie fest, wie lange der Workflow wartet, bis der Benutzer auf eine Benutzerinteraktion antwortet.
- 4 [Berechnen einer relativen Zeitüberschreitung für Benutzerinteraktionen](#) auf Seite 111  
Sie können in einem Date-Objekt eine relative Uhrzeit und ein relatives Datum berechnen, zu der/dem eine Benutzerinteraktion das Zeitlimit überschreitet.
- 5 [Festlegen des Attributs „timeout.date“ auf ein relatives Datum](#) auf Seite 113  
Das Attribut `timeout.date` einer **Benutzerinteraktion** kann auf eine relative Zeit und ein relatives Datum festgelegt werden, indem es an ein Date-Objekt gebunden wird. Das Objekt wird in einer Skriptfunktion definiert.

- 6 [Definieren der externen Eingaben für eine Benutzerinteraktion](#) auf Seite 113  
Sie legen die Informationen fest, die Benutzer während des Workflows als Eingabeparameter einer Benutzerinteraktion angeben müssen.
- 7 [Definieren des Ausnahmeverhaltens bei der Benutzerinteraktion](#) auf Seite 114  
Wenn ein Benutzer während des Zeitlimits keine Eingabeparameter angibt, wird von der Benutzerinteraktion eine Ausnahme zurückgegeben. Sie können das Verhalten bei Ausnahmen in einer Skriptfunktion definieren.
- 8 [Erstellen des Dialogfelds „Eingabeparameter“ für eine Benutzerinteraktion](#) auf Seite 115  
Benutzer geben wie beim ersten Starten des Workflows auch bei der Ausführung von Workflows Eingabeparameter im Dialogfeld „Eingabeparameter“ ein.
- 9 [Antworten auf eine Anforderung für eine Benutzerinteraktion](#) auf Seite 116  
Workflows, die während ihrer Ausführung Interaktionen von Benutzern erfordern, werden angehalten, bis der Benutzer die erforderlichen Informationen angibt oder es beim Workflow zu einer Zeitüberschreitung kommt.


## Hinzufügen einer Benutzerinteraktion zu einem Workflow

Sie können Eingabeparameter von Benutzern während einer Workflowausführung anfordern, indem Sie dem Workflow das Schemaelement **Benutzerinteraktion** hinzufügen. Wenn bei einem Workflow das Element **Benutzerinteraktion** festgestellt wird, wird die Ausführung angehalten und gewartet, bis der Benutzer die erforderlichen Daten angibt.

### Voraussetzungen

- Erstellen Sie einen Workflow.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.
- Fügen Sie dem Workflowschema einige Elemente hinzu.

### Vorgehensweise

- 1 Ziehen Sie das Element **Benutzerinteraktion** an die entsprechende Position im Workflowschema.
- 2 Klicken Sie auf das Symbol **Bearbeiten** () des Elements **Benutzerinteraktion**.
- 3 Geben Sie auf der Registerkarte **Info** einen Namen sowie eine Beschreibung für die Benutzerinteraktion ein und klicken Sie auf **Schließen**.
- 4 Klicken Sie auf **Speichern**.

Sie haben einem Workflow ein Benutzerinteraktionselement hinzugefügt. Wenn der Workflow dieses Element erreicht, wird auf Informationen vom Benutzer gewartet, bevor seine Ausführung fortgesetzt wird.

### Weiter

Legen Sie das Attribut `security.group` der Benutzerinteraktion fest, um die Berechtigung zum Antworten auf die Benutzerinteraktion für einen Benutzer oder eine Benutzergruppe zu beschränken. Weitere Informationen finden Sie unter [„Festlegen des Attributs „security.group““](#), auf Seite 109.

## Festlegen des Attributs „security.group“

Das Attribut `security.group` eines Benutzerinteraktionselements legt fest, welche Benutzer oder Benutzergruppen die Berechtigung zum Antworten auf eine Benutzerinteraktion haben.

### Voraussetzungen

- Erstellen Sie einen Workflow.

- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.
- Fügen Sie dem Workflowschema einige Elemente und eine Benutzerinteraktion hinzu.
- Legen Sie eine LDAP-Benutzergruppe für die Antwort auf die Benutzerinteraktion fest.

### Vorgehensweise

- 1 Klicken Sie auf das Symbol **Bearbeiten** (✎) für das Element **Benutzerinteraktion** im Workflowschema.
- 2 Klicken Sie auf die Registerkarte **Attribute** für die Benutzerinteraktion.
- 3 Klicken Sie auf **Nicht festgelegt** für den security.group-Quellparameter, um festzulegen, welche Benutzer auf die Benutzerinteraktion antworten können.
- 4 (Optional) Wählen Sie **NULL**, wenn alle Benutzer auf die Anforderung der Benutzerinteraktion antworten können sollen.
- 5 Um die Antwortberechtigung auf einen bestimmten Benutzer oder eine bestimmte Benutzergruppe zu beschränken, klicken Sie auf **Parameter/Attribut in Workflow erstellen**.  
Das Dialogfeld **Parameterinformationen** wird geöffnet.
- 6 Benennen Sie den Parameter.
- 7 Wählen Sie Workflow-ATTRIBUT mit demselben Namen erstellen, um das Attribut LdapGroup im Workflow zu erstellen.
- 8 Klicken Sie auf **Nicht festgelegt**, damit der Parameterwert im Auswahlfeld **LdapGroup** geöffnet wird.
- 9 Geben Sie den Namen der LDAP-Benutzergruppe in das Textfeld **Filter** ein.
- 10 Wählen Sie die LDAP-Benutzergruppe aus der Liste aus und klicken Sie auf **Auswählen**.  
Beispiel: Wenn Sie die Benutzergruppe **Administratoren** auswählen, können nur Benutzer dieser Gruppe auf Benutzerinteraktionsanforderungen antworten.  
Sie haben die Berechtigungen zum Antworten auf Benutzerinteraktionsanforderungen eingeschränkt.
- 11 Klicken Sie auf **OK**, um das Dialogfeld **Parameterinformationen** zu schließen.

Sie haben das Attribut security.group für die Benutzerinteraktion eingerichtet.

### Weiter

Legen Sie für das Attribut timer.date die Zeitüberschreitungsdauer für die Benutzerinteraktion fest.

- Informationen zum Festlegen des Zeitüberschreitungslimits auf ein bestimmtes Datum und eine Uhrzeit finden Sie unter „[Festlegen des Attributs „timeout.date“ auf ein absolutes Datum](#)“, auf Seite 110.
- Informationen zum Erstellen einer Funktion zum Berechnen eines Zeitüberschreitungslimits relativ zum aktuellen Datum und der aktuellen Uhrzeit finden Sie unter „[Berechnen einer relativen Zeitüberschreitung für Benutzerinteraktionen](#)“, auf Seite 111.

## Festlegen des Attributs „timeout.date“ auf ein absolutes Datum


Mithilfe des Attributs timeout.date für eine Benutzerinteraktion legen Sie fest, wie lange der Workflow wartet, bis der Benutzer auf eine Benutzerinteraktion antwortet.

Sie können eine absolute Zeit und ein absolutes Datum in einem Date-Objekt festlegen. Wenn die Zeit zum gegebenen Datum erreicht wird, wird das Zeitlimit des Workflows für die Benutzerinteraktion überschritten und der Workflow wird mit dem Status Failed beendet. Beispiel: Das Zeitüberschreitungslimit für die Benutzerinteraktion wird für 12. Februar mittags angesetzt. Informationen zum Berechnen eines Zeitüberschreitungslimits relativ zum aktuellen Datum und der aktuellen Uhrzeit finden Sie unter „[Berechnen einer relativen Zeitüberschreitung für Benutzerinteraktionen](#)“, auf Seite 111.

### Voraussetzungen

- Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.
- Fügen Sie dem Workflowschema ein Element für die Benutzerinteraktion hinzu.
- Richten Sie für die Benutzerinteraktion das Attribut `security.group` ein.

### Vorgehensweise

- 1 Klicken Sie auf das Symbol **Bearbeiten** () für das Element **Benutzerinteraktion** im Workflowschema.
- 2 Klicken Sie auf die Registerkarte **Attribute** für die Benutzerinteraktion.
- 3 Klicken Sie auf **Nicht festgelegt** für den `timeout.date`-Quellparameter, um festzulegen, welche Benutzer auf die Benutzerinteraktion antworten können.
- 4 (Optional) Wählen Sie **NULL**, damit durch die Benutzerinteraktion festgelegt werden kann, dass der Workflow unbegrenzt wartet, bis der Benutzer auf die Benutzerinteraktion antwortet.
- 5 Klicken Sie auf **Parameter/Attribut in Workflow erstellen**, damit der Workflow nach Ablauf eines Zeitlimits fehlschlägt.  
Das Dialogfeld **Parameterinformationen** wird geöffnet.
- 6 Benennen Sie den Parameter.
- 7 Wählen Sie Workflow-ATTRIBUT mit demselben Namen `erstellenDate`, um das Attribut im Workflow zu erstellen.
- 8 Klicken Sie für den Parameter **Wert** auf **Nicht festgelegt**.
- 9 Wählen Sie im Kalender ein absolutes Datum und eine absolute Zeit aus, bis zu welcher der Workflow auf eine Antwort durch den Benutzer wartet.
- 10 Klicken Sie auf **OK**, um den Kalender zu schließen.
- 11 Klicken Sie auf **OK**, um das Dialogfeld **Parameterinformationen** zu schließen.

Sie haben das Attribut `timeout.date` auf ein absolutes Datum festgelegt. Der Workflow läuft aus, wenn der Benutzer nicht vor Ablauf dieser Zeit zu diesem Datum auf die Benutzerinteraktion antwortet.

### Weiter

Legen Sie die externen Eingabeparameter fest, die die Benutzerinteraktion vom Benutzer erfordert. Weitere Informationen finden Sie unter „[Definieren der externen Eingaben für eine Benutzerinteraktion](#)“, auf Seite 113.

## Berechnen einer relativen Zeitüberschreitung für Benutzerinteraktionen

Sie können in einem `Date`-Objekt eine relative Uhrzeit und ein relatives Datum berechnen, zu der/dem eine Benutzerinteraktion das Zeitlimit überschreitet.

Sie können eine absolute Zeit und ein absolutes Datum in einem `Date`-Objekt festlegen. Wenn die Zeit zum gegebenen Datum erreicht wird, wird das Zeitlimit für die Anforderung für eine Benutzerinteraktion überschritten. Alternativ dazu können Sie ein Workflowelement erstellen, das ein relatives `Date`-Objekt gemäß einer von Ihnen definierten Funktion berechnet und generiert. Beispielsweise können Sie ein relatives `Date`-Objekt erstellen, das 24 Stunden zur momentanen Uhrzeit hinzufügt.

### Voraussetzungen

- Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.
- Fügen Sie dem Workflowschema ein Element für die Benutzerinteraktion hinzu.

- Richten Sie für die Benutzerinteraktion das Attribut `security.group` ein.

### Vorgehensweise

- 1 Ziehen Sie ein **skriptfähiges Aufgabenelement** aus dem Menü **Generisch** in das Schema eines Workflows vor das Element, das das relative Objekt `Date` für sein Attribut `timeout.date` benötigt.
- 2 Klicken Sie auf das Symbol **Bearbeiten** (✎) für das Element **Skriptfähige Aufgabe** im Workflowschema.
- 3 Geben Sie einen Namen und eine Beschreibung für das Element für skriptfähige Aufgaben auf der Eigenschaftenregisterkarte **Info** an.
- 4 Klicken Sie auf die Eigenschaftenregisterkarte **AUS** und auf das Symbol **An Workflowparameter/-attribut binden** (🔗).
- 5 Klicken Sie auf **Parameter/Attribut in Workflow erstellen**, um ein Workflowattribut zu erstellen.
  - a Nennen Sie das Attribut `timerDate`
  - b Wählen Sie `Date` aus der Liste der Attributtypen.
  - c Wählen Sie **Workflow-ATTRIBUT mit demselben Namen erstellen**.
  - d Belassen Sie den Attributwert auf **Nicht festgelegt**, weil eine in einem Skript programmierte Funktion diesen Wert bereitstellen wird.
  - e Klicken Sie auf **OK**.

- 6 Klicken Sie auf die Registerkarte **Skripterstellung** des skriptfähigen Aufgabenelements.

- 7 Definieren Sie eine Funktion zum Berechnen und Regenerieren eines Objekts `Date` mit dem Namen `timerDate` im Skripterstellungsbereich auf der Registerkarte **Skripterstellung**.

Sie könnten beispielsweise ein Objekt `Date` erstellen, indem Sie die folgende JavaScript-Funktion implementieren, in der die Zeitüberschreitungsperiode eine Relativverzögerung in Millisekunden darstellt.

```
timerDate = new Date();
System.log( "Current date : " + timerDate + " " );
timerDate.setTime( timerDate.getTime() + (86400 * 1000) );
System.log( "Timer will expire at " + timerDate + " " );
```

Die JavaScript-Funktion im vorherigen Beispiel definiert ein Objekt `Date`, das das aktuelle Datum und die aktuelle Zeit ermittelt, indem die Methode `getTime` verwendet wird und 68.400.000 Millisekunden oder 24 Stunden hinzu addiert werden. Das **skriptfähige Aufgabenelement** generiert diesen Wert als seinen Ausgabeparameter.

- 8 Klicken Sie auf **Schließen**.
- 9 Klicken Sie auf **Speichern**.

Sie haben eine Funktion erstellt, die eine Uhrzeit und ein Datum berechnet, die/das relativ zur momentanen Uhrzeit und zum momentanen Datum ist, und ein `Date`-Objekt generiert. Ein **Benutzerinteraktions**-Element kann dieses `Date`-Objekt als einen Eingabeparameter empfangen, um festzulegen, wie lange auf die Eingabe vom Benutzer gewartet werden soll. Wenn der Workflow das **Benutzerinteraktions**-Element erreicht, wird die Ausführung angehalten und der Workflow wartet entweder, bis der Benutzer die erforderlichen Informationen eingibt, oder er wartet 24 Stunden, bevor die Zeitüberschreitung ausgelöst wird.

### Weiter

Sie müssen das `Date`-Objekt an den Parameter `timeout.date` des **Benutzerinteraktions**-Elements binden. Weitere Informationen finden Sie unter „[Festlegen des Attributs „timeout.date“ auf ein relatives Datum](#)“, auf Seite 113.



## Festlegen des Attributs „timeout.date“ auf ein relatives Datum

Das Attribut `timeout.date` einer **Benutzerinteraktion** kann auf eine relative Zeit und ein relatives Datum festgelegt werden, indem es an ein Date-Objekt gebunden wird. Das Objekt wird in einer Skriptfunktion definiert.

Wenn ein relatives Date-Objekt in einer Skriptfunktion erstellt wird, können Sie das Attribut `timeout.date` einer Benutzerinteraktion an dieses Date-Objekt binden. Beispiel: Wenn Sie das Attribut `timeout.date` an ein Date-Objekt binden, das 24 Stunden zur jetzigen Uhrzeit hinzufügt, läuft die Benutzerinteraktion nach 24 Stunden ab.

### Voraussetzungen

- Add a user interaction element to the workflow schema.
- Set the `security.group` attribute for the user interaction.
- Erstellen Sie eine Skriptfunktion, die eine relative Zeit und ein relatives Datum berechnet und in ein Date-Objekt im Workflow einschließt. Weitere Informationen finden Sie unter [„Berechnen einer relativen Zeitüberschreitung für Benutzerinteraktionen“](#), auf Seite 111.

### Vorgehensweise

- 1 Klicken Sie auf das Symbol **Bearbeiten** (✎) für das Element **Benutzerinteraktion** im Workflowschema.
- 2 Klicken Sie auf die Registerkarte **Attribute** für die Benutzerinteraktion.
- 3 Klicken Sie auf **Nicht festgelegt** für den `timeout.date`-Quellparameter, um festzulegen, welche Benutzer auf die Benutzerinteraktion antworten können.
- 4 Wählen Sie das Date-Objekt aus, das die relative Zeit und das relative Datum einschließt, die/das in der Skriptfunktion definiert sind, und klicken Sie auf **Auswählen**.

Sie haben das Attribut `timeout.date` auf ein relatives Datum und eine relative Uhrzeit festgelegt, das/die von der Skriptfunktion berechnet wird.

### Weiter

Legen Sie die externen Eingabeparameter fest, die die Benutzerinteraktion vom Benutzer erfordert. Weitere Informationen finden Sie unter [„Definieren der externen Eingaben für eine Benutzerinteraktion“](#), auf Seite 113.

## Definieren der externen Eingaben für eine Benutzerinteraktion

Sie legen die Informationen fest, die Benutzer während des Workflows als Eingabeparameter einer Benutzerinteraktion angeben müssen.


Wenn ein Workflow bei einem Benutzerinteraktionselement anlangt, wartet er, bis ein Benutzer die für die Benutzerinteraktion erforderlichen Informationen als Eingabeparameter bereitstellt.

### Voraussetzungen

- Fügen Sie dem Workflowschema ein Element für die Benutzerinteraktion hinzu.
- Set the `security.group` attribute for the user interaction.
- Festlegen des Attributs `timer.date` für eine Benutzerinteraktion

### Vorgehensweise

- 1 Klicken Sie auf das Symbol **Bearbeiten** (✎) für das Element **Benutzerinteraktion** im Workflowschema.

- 2 Klicken Sie auf die Registerkarte **Externe Eingaben**.
- 3 Klicken Sie auf das Symbol **An Workflowparameter/-attribut binden** () , um die Parameter zu definieren, die der Benutzer bei der Benutzerinteraktion angeben muss.
- 4 (Optional) Wenn Sie die Eingabeparameter bereits im Workflow definiert haben, wählen Sie die Parameter aus der vorgeschlagenen Liste aus.
- 5 Klicken Sie auf **Parameter/Attribut im Workflow erstellen**, um ein Workflowattribut zu erstellen, an das die vom Benutzer angegebenen Eingabeparameter gebunden werden können.
- 6 Geben Sie dem Parameter einen passenden Namen.
- 7 Wählen Sie den Typ der Eingabeparameter aus der Typliste aus, indem Sie im Feld **Filter** nach einem Objekt suchen.  
  
Wenn die Benutzerinteraktion beispielsweise vom Benutzer die Bereitstellung einer virtuellen Maschine als Eingabeparameter verlangt, wählen Sie VC:VirtualMachine.
- 8 Wählen Sie **Workflow-ATTRIBUT mit demselben Namen erstellen**, um den vom Benutzer angegebenen Eingabeparameter an ein neues Attribut im Workflow zu binden.
- 9 Belassen Sie die Einstellung des Eingabeparameters auf **Nicht festgelegt**.  
  
Der Benutzer gibt diesen Wert an, wenn er während der Workflowausführung auf die Benutzerinteraktion antwortet.
- 10 Klicken Sie auf **OK**, um das Dialogfeld **Parameterinformationen** zu schließen.

Sie haben Eingabeparameter definiert, die Benutzer bei der Benutzerinteraktion angeben müssen.

#### Weiter

Definieren Sie das Verhalten bei Ausnahmen, wenn bei der Benutzerinteraktion ein Fehler auftritt. Weitere Informationen finden Sie unter „[Definieren des Ausnahmeverhaltens bei der Benutzerinteraktion](#)“, auf Seite 114.

## Definieren des Ausnahmeverhaltens bei der Benutzerinteraktion


Wenn ein Benutzer während des Zeitlimits keine Eingabeparameter angibt, wird von der Benutzerinteraktion eine Ausnahme zurückgegeben. Sie können das Verhalten bei Ausnahmen in einer Skriptfunktion definieren.

Wenn Sie die Aktion des Workflows bei Überschreiten des Zeitlimits für die Benutzerinteraktion nicht festlegen, endet der Workflow mit dem Status **Failed**. Die Definition des Verhaltens bei Ausnahmen wird bei der Entwicklung von Workflows empfohlen.

#### Voraussetzungen

- Fügen Sie dem Workflowschema ein Element für die Benutzerinteraktion hinzu.
- Legen Sie die Attribute `security.group` und `timer.date` für die Benutzerinteraktion fest.
- Legen Sie die externen Eingabeparameter für die Benutzerinteraktion fest.

#### Vorgehensweise

- 1 Klicken Sie auf das Symbol **Bearbeiten** () für das Element **Benutzerinteraktion** im Workflowschema.
- 2 Klicken Sie auf die Registerkarte **Ausnahme**.
- 3 Klicken Sie für die Ausnahmebindung der Ausgabe auf **Nicht festgelegt**.

- 4 Klicken Sie auf **Parameter/Attribut im Workflow erstellen**, um ein Ausnahmeattribut zu erstellen, an das die Benutzerinteraktion gebunden werden kann.

Das Dialogfeld **Parameterinformationen** wird geöffnet.

- 5 Erstellen Sie ein `errorCode`-Attribut.

Ein `errorCode`-Attribut hat folgende Standardeigenschaften:

- Name: **errorCode**
- Typ: Zeichenfolge
- Erstellen: **Workflow-ATTRIBUT mit demselben Namen erstellen**
- Wert: Passende Fehlermeldung eingeben.

- 6 Klicken Sie auf **OK**, um das Dialogfeld **Parameterinformationen** zu schließen.

- 7 Ziehen Sie ein skriptfähiges Aufgabenelement auf das Benutzerinteraktionselement im Workflowschema.

Ein roter, gestrichelter Pfeil, der den Ausnahmelink repräsentiert, wird zwischen den beiden Elementen angezeigt. Das skriptfähige Aufgabenelement bindet sich automatisch an das `errorCode`-Attribut der Benutzerinteraktion.

- 8 Doppelklicken Sie auf das skriptfähige Element und geben Sie einen passenden Namen ein.

Beispiel: **Zeitüberschreitung protokollieren**.

- 9 Schreiben Sie in der Registerkarte **Skripterstellung** des skriptfähigen Aufgabenelements eine JavaScript-Funktion zur Behandlung von Ausnahmen.

Schreiben Sie beispielsweise folgende Funktion zur Aufzeichnung der Zeitüberschreitung im Protokoll von Orchestrator:

```
System.log("No response from user. Timed out.");
```

- 10 Verknüpfen und binden Sie das skriptfähige Aufgabenelement für die Ausnahmen an das Element, das ihm im Workflow folgt.

Verknüpfen und binden Sie beispielsweise das skriptfähige Aufgabenelement an ein **Ausnahme auslösen**-Element, um den Workflow mit einem Fehler zu beenden.

Sie haben das Verhalten bei Ausnahmen bei Überschreiten des Zeitlimits für die Benutzerinteraktion definiert.

### Weiter

Erstellen Sie ein Dialogfeld, in dem Benutzer Eingabeparameter angeben können. Weitere Informationen finden Sie unter „[Erstellen des Dialogfelds „Eingabeparameter“ für eine Benutzerinteraktion](#)“, auf Seite 115.

## Erstellen des Dialogfelds „Eingabeparameter“ für eine Benutzerinteraktion

Benutzer geben wie beim ersten Starten des Workflows auch bei der Ausführung von Workflows Eingabeparameter im Dialogfeld „Eingabeparameter“ ein.

Sie erstellen das Layout des Dialogfelds auf der Registerkarte **Präsentation** des Benutzerinteraktionselements und nicht auf der Registerkarte **Präsentation** für den gesamten Workflow. Über die Registerkarte **Präsentation** für den gesamten Workflow wird das Layout des Dialogfelds „Eingabeparameter“ erstellt, das beim Starten eines Workflows angezeigt wird. Über die Registerkarte **Präsentation** für das Interaktionselement wird das Layout des Dialogfelds „Eingabeparameter“ erstellt, das geöffnet wird, wenn ein Workflow bei seiner Ausführung bei einem Benutzerinteraktionselement anlangt.

### Voraussetzungen

- Fügen Sie dem Workflowschema ein Element für die Benutzerinteraktion hinzu.
- Legen Sie für die Benutzerinteraktion die Attribute `security.group` und `timer.date` fest.
- Legen Sie die externen Eingabeparameter für die Benutzerinteraktion fest.
- Definieren Sie das Verhalten bei Ausnahmen.

### Vorgehensweise

- 1 Klicken Sie auf das Symbol **Bearbeiten** (✎) für das Element **Benutzerinteraktion** im Workflowschema.
- 2 Klicken Sie auf die Registerkarte **Präsentation** des Benutzerinteraktionselements.  
Die Registerkarte **Präsentation** zeigt externe Eingabeparameter, die Sie für die Benutzerinteraktion erstellt haben.
- 3 (Optional) Klicken Sie mit der rechten Maustaste auf den Knoten **Präsentation** auf der Registerkarte **Präsentation** und wählen Sie **Neuen Schritt erstellen** aus.  
Mithilfe von Schritten können Sie Abschnitte im Dialogfeld erstellen, die Beschreibungen und Überschriften zur Organisation der Eingabeparameter enthalten.
- 4 (Optional) Klicken Sie mit der rechten Maustaste auf den Knoten **Präsentation** auf der Registerkarte **Präsentation** und wählen Sie **Anzeigegruppe erstellen**.  
Mit Anzeigegruppen können Sie die Reihenfolge festlegen, in der die Eingabeparameter in diesem Schritt angezeigt werden. Es ist zudem möglich, dem Dialogfeld Zwischenüberschriften und Beschreibungen hinzuzufügen.
- 5 Klicken Sie auf einen Eingabeparameter in der Liste und fügen Sie auf der Registerkarte **Allgemein** dieses Parameters eine Beschreibung hinzu.  
Der von Ihnen eingegebene Beschreibungstext wird als Bezeichnung im Dialogfeld „Eingabeparameter“ angezeigt, um den Benutzer über die zur Interaktion erforderlichen Informationen aufzuklären.
- 6 Definieren Sie die Eigenschaften der Eingabeparameter.  
Mithilfe der Eigenschaften der Eingabeparameter können Sie die zulässigen Werte, die Benutzer für die Eingabeparameter angeben können, festlegen und zudem Parameterwerte mithilfe von OGNL-Ausdrücken dynamisch bestimmen.
- 7 Klicken Sie auf **Speichern und schließen**, um den Workfloweditor zu schließen.

Sie haben das Dialogfeld Eingabeparameter erstellt, in dem Benutzer Eingabeparameter als Antwort einer Benutzerinteraktion im Rahmen eines Workflows angeben können.

### Weiter

Informationen zum Erstellen von Präsentationsschritten und Gruppen und zum Einrichten der Eigenschaften von Eingabeparametern finden Sie unter [„Erstellen des Dialogfelds „Eingabeparameter“ auf der Registerkarte „Präsentation“](#), auf Seite 103.

## Antworten auf eine Anforderung für eine Benutzerinteraktion

Workflows, die während ihrer Ausführung Interaktionen von Benutzern erfordern, werden angehalten, bis der Benutzer die erforderlichen Informationen angibt oder es beim Workflow zu einer Zeitüberschreitung kommt.

Workflows, die Benutzerinteraktionen erfordern, definieren, welche Benutzer die benötigten Informationen angeben können, und leiten die Interaktionsanforderungen entsprechend weiter.

## Voraussetzungen


Stellen Sie sicher, dass sich mindestens ein Workflow im Zustand „Warten auf Benutzerinteraktion“ befindet.

## Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Ausführen** aus.
- 2 Klicken Sie auf die Ansicht **Mein Orchestrator** im Orchestrator-Client.
- 3 Klicken Sie auf die Registerkarte **Warten auf Eingabe**.

Auf der Registerkarte **Warten auf Eingabe** werden die Workflows aufgelistet, die auf Benutzereingaben von Ihnen oder von Mitgliedern Ihrer Benutzergruppe mit der entsprechenden Berechtigung warten.

- 4 Doppelklicken Sie auf einen Workflow, der auf eine Benutzerinteraktion wartet.

Das Workflowtoken, das auf eine Eingabe wartet, wird in der hierarchischen Liste **Workflows** mit dem folgenden Symbol angezeigt: .

- 5 Klicken Sie mit der rechten Maustaste auf das Workflowtoken und wählen Sie **Antworten** aus.
- 6 Befolgen Sie die Anweisungen im Dialogfeld „Eingabeparameter“, um die vom Workflow benötigten Informationen anzugeben.

Sie haben Informationen für einen Workflow angegeben, der während seiner Ausführung auf eine Benutzereingabe gewartet hat.

## Aufrufen von Workflows innerhalb von Workflows

Workflows können während ihrer Ausführung andere Workflows aufrufen. Ein Workflow kann einen anderen Workflow entweder starten, weil er das Ergebnis des anderen Workflows als Eingabeparameter für seine eigene Ausführung benötigt, oder er kann einen Workflow starten und seine Ausführung unabhängig fortsetzen lassen. Workflows können einen Workflow auch zu einem bestimmten zukünftigen Zeitpunkt starten oder mehrere Workflows gleichzeitig starten.

- [Workflowelemente, die Workflows aufrufen](#) auf Seite 118

Es gibt vier Möglichkeiten, aus einem Workflow andere Workflows aufzurufen. Jede Möglichkeit des Aufrufs eines Workflows wird durch ein anderes Workflowschemaelement dargestellt.

- [Synchrones Aufrufen eines Workflows](#) auf Seite 120

Durch synchrones Aufrufen eines Workflows wird der aufgerufene Workflow als Teil der Ausführung des aufrufenden Workflows ausgeführt. Der aufrufende Workflow kann die Ausgabeparameter des aufgerufenen Workflows als Eingabeparameter für die Ausführung der nachfolgenden Schemaelemente verwenden.

- [Asynchrones Aufrufen eines Workflows](#) auf Seite 121

Durch asynchrones Aufrufen eines Workflows wird der aufgerufene Workflow unabhängig vom aufrufenden Workflow ausgeführt. Der aufrufende Workflow setzt seine Ausführung fort, ohne auf die Durchführung des aufgerufenen Workflows zu warten.

- [Planen eines Workflows](#) auf Seite 122

Sie können einen Workflow aus einem Workflow aufrufen und für einen späteren Zeitpunkt oder ein anderes Datum planen.

- [Voraussetzungen für das Abrufen eines Remoteworkflows aus einem anderen Workflow](#) auf Seite 123

Wenn der Workflow, den Sie entwickeln, einen anderen Workflow aufruft, der sich auf einem entfernten Orchestrator-Server befindet, müssen bestimmte Voraussetzungen erfüllt sein, sodass der Remote-workflow erfolgreich ausgeführt werden kann.

- [Gleichzeitiges Aufrufen mehrerer Workflows](#) auf Seite 124

Durch gleichzeitiges Aufrufen mehrerer Workflows wird der aufgerufene Workflow als Teil der Ausführung des aufrufenden Workflows synchron ausgeführt. Der aufrufende Workflow wartet auf die Beendigung aller aufgerufenen Workflows, bevor er fortgesetzt wird. Der aufrufende Workflow kann die Ergebnisse des aufgerufenen Workflows als Eingabeparameter bei der Ausführung der nachfolgenden Schemaelemente verwenden.

## Workflowelemente, die Workflows aufrufen

Es gibt vier Möglichkeiten, aus einem Workflow andere Workflows aufzurufen. Jede Möglichkeit des Aufrufs eines Workflows wird durch ein anderes Workflowschemaelement dargestellt.

### Synchrone Workflows

Ein Workflow kann einen anderen Workflow synchron starten. Der aufgerufene Workflow wird als integraler Bestandteil des aufrufenden Workflows in demselben Arbeitsbereich wie der aufrufende Workflow ausgeführt. Der aufrufende Workflow startet einen anderen Workflow und wartet dann, bis der aufgerufene Workflow seine Ausführung beendet hat, bevor er das nächste Element in seinem Schema startet. Normalerweise rufen Sie einen Workflow synchron auf, weil der aufrufende Workflow die Ausgabe des aufgerufenen Workflows als Eingabeparameter für ein nachfolgendes Schemaelement benötigt. Ein Workflow kann beispielsweise den Workflow „Virtuelle Maschine starten und warten“ aufrufen, um eine virtuelle Maschine zu starten, und dann die IT-Adresse dieser virtuellen Maschine benötigen, um sie an ein anderes Element oder an einen Benutzer per E-Mail weiterzugeben.

### Asynchrone Workflows

Ein Workflow kann einen Workflow asynchron starten. Der aufrufende Workflow startet einen anderen Workflow, aber der aufrufende Workflow läuft selbst mit seinem nächsten Schemaelement weiter, ohne auf das Ergebnis des aufgerufenen Workflows zu warten. Der aufgerufene Workflow wird mit Eingabeparametern ausgeführt, die der aufrufende Workflow definiert, aber der Lebenszyklus des aufgerufenen Workflows ist unabhängig vom Lebenszyklus des aufrufenden Workflows. Asynchrone Workflows erlauben Ihnen, Ketten von Workflows zu erstellen, die Eingabeparameter von einem Workflow zum nächsten weitergeben. Beispielsweise kann ein Workflow verschiedene Objekte erstellen, während er ausgeführt wird. Der Workflow kann dann asynchrone Workflows starten, die diese Objekte als Eingabeparameter für ihre eigene Ausführung verwenden. Wenn der ursprüngliche Workflow alle erforderlichen Workflows gestartet und seine restlichen Elemente abgearbeitet hat, endet er. Die asynchronen Workflows, die er gestartet hat, laufen aber unabhängig von dem Workflow, der sie gestartet hat, weiter.

Damit der aufrufende Workflow auf das Ergebnis des aufgerufenen Workflows wartet, verwenden Sie entweder einen verschachtelten Workflow oder erstellen eine skriptfähige Aufgabe, die den Status des Workflowtokens des aufgerufenen Workflows ermittelt und dann das Ergebnis des Workflows abruft, wenn er abgeschlossen ist.

### Geplante Workflows

Ein Workflow kann einen Workflow aufrufen, aber den Start dieses Workflows bis zu einem späteren Zeitpunkt oder Datum verschieben. Der aufrufende Workflow wird danach fortgesetzt, bis er endet. Durch das Aufrufen eines geplanten Workflows wird eine Aufgabe zum Start dieses Workflows zu einem bestimmten Zeitpunkt und Datum erstellt. Wenn der aufrufende Workflow fertig ist, können Sie den geplanten Workflow in der Ansicht **Zeitplan** und **Mein Orchestrator** im Orchestrator-Client sehen.

Geplante Workflows werden nur einmal ausgeführt. Sie können einen Workflow so planen, dass er wiederholt ausgeführt wird, indem Sie die Methode `Workflow.scheduleRecurrently` in einem skriptfähigen Aufgabenelement in einem synchronen Workflow aufrufen.

### **Verschachtelte Workflows**

Ein Workflow kann mehrere Workflows simultan starten, indem mehrere Workflows in einem einzelnen Schemaelement verschachtelt werden. Alle Workflows, die im verschachtelten Netzwerkelement aufgelistet sind, starten simultan, wenn der aufrufende Workflow in seinem Schema zum verschachtelten Workflowelement kommt. Das Besondere ist, dass jeder verschachtelte Workflow in einem eigenen Arbeitsspeicherbereich gestartet wird, der von dem des aufrufenden Workflows verschieden ist. Der aufrufende Workflow wartet, bis alle verschachtelten Workflows abgeschlossen sind, bevor er das nächste Element in seinem Schema startet. Der aufrufende Workflow kann daher die Ergebnisse der verschachtelten Workflows als Eingabeparameter nutzen, wenn er seine restlichen Elemente ausführt.

## **Übertragen von Workflowänderungen in andere Workflows**

Wenn Sie einen Workflow aus einem anderen Workflow aufrufen, importiert Orchestrator die Eingabeparameter des untergeordneten Workflows in den übergeordneten Workflow in dem Moment, in dem Sie das Workflowelement hinzufügen.

Wenn Sie den untergeordneten Workflow ändern, nachdem Sie ihn einem anderen Workflow hinzugefügt haben, ruft der übergeordnete Workflow die neue Version des untergeordneten Workflows auf, importiert aber keine neuen Eingabeparameter. Um zu verhindern, dass Änderungen an Workflows das Verhalten von anderen Workflows beeinflusst, die sie aufrufen, überträgt Orchestrator die neuen Eingabeparameter nicht automatisch in die aufrufenden Workflows.

Zur Übertragung von Parametern von einem Workflow auf andere Workflows, die ihn aufrufen, müssen Sie die Workflows ermitteln, die den Workflow aufrufen, und die Workflows manuell synchronisieren.

### **Voraussetzungen**

Überprüfen Sie, ob Sie einen Workflow haben, der einen anderen Workflow aufruft oder von anderen Workflows aufgerufen wird.

### **Vorgehensweise**

- 1 Ändern und speichern Sie einen Workflow, der andere Workflows aufruft.
- 2 Schließen Sie den Workfloweditor.
- 3 Navigieren Sie zu dem von Ihnen geänderten Workflow in der hierarchischen Liste in der Ansicht **Workflows** im Orchestrator-Client.
- 4 Klicken Sie mit der rechten Maustaste auf den Workflow und wählen Sie **Referenzen > Elemente suchen, die dieses Element verwenden**.

Eine Liste der Workflows, die diesen Workflow aufrufen, erscheint.

- 5 Doppelklicken Sie auf einen Workflow in der Liste, um ihn in der Ansicht **Workflows** im Orchestrator-Client hervorzuheben.
- 6 Klicken Sie mit der rechten Maustaste auf den Workflow und wählen Sie **Bearbeiten**.  
Der Workfloweditor wird geöffnet.
- 7 Klicken Sie im Workfloweditor auf die Registerkarte **Schema**.
- 8 Klicken Sie mit der rechten Maustaste auf das Workflowelement für den geänderten Workflow im Workflowschema und wählen Sie **Synchronisieren > Parameter synchronisieren**.

- 9 Wählen Sie **Weiter** im Dialogfeld zur Bestätigung.
- 10 Speichern und schließen Sie den Workfloweditor.
- 11 Wiederholen Sie [Schritt 5](#) bis [Schritt 10](#) für alle Workflows, die den geänderten Workflow verwenden.

Sie haben einen geänderten Workflow auf andere Workflows übertragen, die in aufrufen.

## Weitergeben von Eingabeparametern und Präsentation eines untergeordneten Workflows an den übergeordneten Workflow

Wenn Sie einen Workflow entwickeln, der andere Workflows aufruft, können Sie die Eingabeparameter und die Präsentation der untergeordneten Workflows an den übergeordneten Workflow weitergeben.

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Ausführen** aus.
- 2 Klicken Sie mit der rechten Maustaste auf den zu ändernden Workflow und wählen Sie **Bearbeiten**.  
Der Workfloweditor wird geöffnet.
- 3 Wählen Sie die Registerkarte **Schema**.
- 4 Klicken Sie mit der rechten Maustaste auf das Element des untergeordneten Workflows, dessen Eingabeparameter und Präsentation Sie an den übergeordneten Workflow weitergeben möchten, und wählen Sie **Synchronisieren > Präsentation synchronisieren**.
- 5 Wählen Sie im Bestätigungsdialogfeld **OK** aus.
- 6 (Optional) Wiederholen Sie [Schritt 4](#) und [Schritt 5](#) für alle untergeordneten Workflows, deren Eingabeparameter und Präsentation Sie an den übergeordneten Workflow weitergeben möchten.

Die Eingabeparameter der untergeordneten Workflows werden den Eingabeparametern des übergeordneten Workflows hinzugefügt. Die Präsentation des übergeordneten Workflows wird um die Präsentationen der untergeordneten Workflows erweitert.

## Synchrones Aufrufen eines Workflows

Durch synchrones Aufrufen eines Workflows wird der aufgerufene Workflow als Teil der Ausführung des aufrufenden Workflows ausgeführt. Der aufrufende Workflow kann die Ausgabeparameter des aufgerufenen Workflows als Eingabeparameter für die Ausführung der nachfolgenden Schemaelemente verwenden.

Sie können Workflows über einen anderen Workflow synchron aufrufen, indem Sie das Element **Workflow** verwenden.

### Voraussetzungen

- Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.
- Fügen Sie dem Workflowschema einige Elemente hinzu.

### Vorgehensweise

- 1 Ziehen Sie das Element **Workflow** aus dem Menü **Generisch** an die entsprechende Position im Workflowschema.

Daraufhin wird das Auswahldialogfeld für **Workflow auswählen** angezeigt.

- 2 Suchen Sie nach dem gewünschten Workflow, wählen Sie ihn aus und klicken Sie auf **OK**.

Wenn die Suche ein Teilergebnis zurückgibt, grenzen Sie Ihr Suchkriterium ein oder erhöhen Sie die Anzahl der Suchergebnisse über die Menüfolge **Tools > Benutzereinstellungen** im Client.



- 3 Klicken Sie auf das Element **Workflow** und zeigen Sie in der unteren Hälfte der Registerkarte **Schema** seine Eigenschaften-Registerkarten an.
- 4 Klicken Sie auf das Symbol **Bearbeiten** (✎) für das Element **Workflow** im Workflowschema.
- 5 Binden Sie auf der Registerkarte **EIN** des Workflowschema-Elements die erforderlichen Eingabeparameter an den Workflow.
- 6 Binden Sie auf der Registerkarte **AUS** des Workflowschema-Elements die erforderlichen Ausgabeparameter an den Workflow.
- 7 Definieren Sie das Verhalten des Workflows bei Ausnahmen auf der Registerkarte **Ausnahmen**.
- 8 Klicken Sie auf **Schließen**.
- 9 Klicken Sie unten im Workfloweditor auf **Speichern**.

Sie haben einen Workflow über einen anderen Workflow synchron aufgerufen. Wenn der Workflow während seiner Ausführung den synchronen Workflow erreicht, startet der synchrone Workflow. Der anfängliche Workflow wartet auf seine Beendigung, bevor seine Ausführung fortgesetzt wird.

#### Weiter

Sie können einen Workflow über einen anderen Workflow synchron aufrufen.

## Asynchrones Aufrufen eines Workflows

Durch asynchrones Aufrufen eines Workflows wird der aufgerufene Workflow unabhängig vom aufrufenden Workflow ausgeführt. Der aufrufende Workflow setzt seine Ausführung fort, ohne auf die Durchführung des aufgerufenen Workflows zu warten.

Sie können Workflows über einen anderen Workflow asynchron aufrufen, indem Sie das Element **Asynchroner Workflow** verwenden.

#### Voraussetzungen

- Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.
- Fügen Sie dem Workflowschema einige Elemente hinzu.

#### Vorgehensweise

- 1 Ziehen Sie das Element **Asynchroner Workflow** aus dem Menü **Generisch** an die entsprechende Position im Workflowschema.  
Daraufhin wird das Auswahldialogfeld für **Workflow auswählen** angezeigt.
- 2 Suchen Sie in der Liste nach dem gewünschten Workflow, wählen Sie ihn aus und klicken Sie auf **OK**.
- 3 Klicken Sie auf das Symbol **Bearbeiten** (✎) für das Element **Asynchroner Workflow** im Workflowschema.
- 4 Binden Sie auf der Registerkarte **EIN** des asynchronen Workflowelements die erforderlichen Eingabeparameter an den Workflow.

- 5 Binden Sie auf der Registerkarte **AUS** des asynchronen Workflowelements den erforderlichen Ausgabe-parameter.

Sie können den Ausgabeparameter entweder an den aufgerufenen Workflow oder an das Ergebnis dieses Workflows binden.

- Binden Sie an den aufgerufenen Workflow, um diesen Workflow als Ausgabeparameter zurückzugeben.
- Wenn die Bindung an das Workflow-Token des aufgerufenen Workflows erfolgt, wird das Ergebnis der Ausführung des aufgerufenen Workflows zurückgegeben.

- 6 Definieren Sie das Verhalten bei Ausnahmen des asynchronen Workflowelements auf der Registerkarte **Ausnahmen**.

- 7 Klicken Sie auf **Schließen**.

- 8 Klicken Sie unten im Workfloweditor auf **Speichern**.

Sie haben einen Workflow asynchron über einen anderen Workflow aufgerufen. Wenn der Workflow während seiner Ausführung den asynchronen Workflow erreicht, startet der asynchrone Workflow. Der anfängliche Workflow setzt seine Ausführung fort, ohne auf das Beenden des asynchronen Workflows zu warten.

### Weiter

Sie können den Start eines Workflows zu einer späteren Uhrzeit bzw. einem späteren Datum planen.

## Planen eines Workflows

Sie können einen Workflow aus einem Workflow aufrufen und für einen späteren Zeitpunkt oder ein anderes Datum planen.

Sie planen Workflows in einem anderen Workflow über das Element **Workflow planen**.

### Voraussetzungen

- Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.
- Fügen Sie dem Workflowschema einige Elemente hinzu.

### Vorgehensweise

- 1 Ziehen Sie das Element **Workflow planen** aus dem Menü **Generisch** an die entsprechende Position im Workflowschema.
- 2 Suchen Sie nach dem Workflow, der aufgerufen werden soll, indem Sie einen Teil seines Namens im Textfeld eingeben.
- 3 Wählen Sie den Workflow aus der Liste und klicken Sie auf **OK**.
- 4 Klicken Sie auf das Symbol **Bearbeiten** (✎) für das Element **Workflow planen** im Workflowschema.
- 5 Klicken Sie auf die Eigenschaftenregisterkarte **EIN**.  
Ein Parameter mit der Bezeichnung workflowScheduleDate erscheint in der Liste der zu definierenden Eigenschaften gemeinsam mit den Eingabeparametern des aufrufenden Workflows.
- 6 Klicken Sie auf **Nicht festgelegt** für den Parameter workflowScheduleDate, um ihn einzurichten.
- 7 Klicken Sie auf **Parameter/Attribut in Workflow erstellen**, um den Parameter zu erstellen und den Parameterwert einzurichten.
- 8 Klicken Sie auf **Nicht festgelegt** für den **Wert**, um den Parameterwert festzulegen.

- 9 Verwenden Sie den eingeblendeten Kalender, um das Datum und den Zeitpunkt für den Start des geplanten Workflows festzulegen. Klicken Sie auf **OK**.
- 10 Binden Sie auf der Registerkarte **EIN** des Elements für den geplanten Workflow die restlichen Eingabeparameter an den Workflow.
- 11 Binden Sie die erforderlichen Ausgabeparameter an das Objekt Task auf der Registerkarte **AUS** des Elements für den geplanten Workflow.
- 12 Definieren Sie das Verhalten bei Ausnahmen des Elements für den geplanten Workflow auf der Registerkarte **Ausnahmen**.
- 13 Klicken Sie auf **Schließen**.
- 14 Klicken Sie unten im Workfloweditor auf **Speichern**.

Sie haben geplant, dass ein Workflow zu einem bestimmten Zeitpunkt an einem bestimmten Datum aus einem anderen Workflow startet.

#### Weiter

Sie können mehrere Workflows gleichzeitig aus einem Workflow aufrufen.

## Voraussetzungen für das Abrufen eines Remoteworkflows aus einem anderen Workflow

Wenn der Workflow, den Sie entwickeln, einen anderen Workflow aufruft, der sich auf einem entfernten Orchestrator-Server befindet, müssen bestimmte Voraussetzungen erfüllt sein, sodass der Remoteworkflow erfolgreich ausgeführt werden kann.

- Alle Eingabeparameter des Remoteworkflows müssen auf dem entfernten Orchestrator-Server auflösbar sein.
- Alle Ausgabeparameter des Remoteworkflows müssen auf dem lokalen Orchestrator-Server auflösbar sein.

Damit alle Parameter des Remoteworkflows auflösbar sind, müssen die Bestandslistenobjekte, die der Workflow verwendet, sowohl auf dem entfernten als auch auf dem lokalen Orchestrator-Server verfügbar sein. Sollte der Remoteworkflow Objekte aus einem Plug-In verwenden, muss dasselbe Plug-In auf beiden Servern verfügbar sein. Die Bestandslisten des entfernten Plug-Ins und des lokalen Plug-Ins müssen identisch sein. Sollte der Remoteworkflow Systemobjekte in Orchestrator verwenden, zum Beispiel Workflows und Aktionen, müssen dieselben Workflows und Aktionen in den Bestandslisten des entfernten und des lokalen Orchestrator-Servers vorhanden sein.

Beispiel: Angenommen, Sie geben den Workflow zum Umbenennen einer virtuellen Maschine in ein verschachteltes Workflowelement in dem Testworkflow ein, den Sie entwickeln. Sie möchten den Workflow zum Umbenennen einer virtuellen Maschine auf einem entfernten Orchestrator-Server ausführen. Wenn Sie den Testworkflow ausführen, wird der Workflow zum Umbenennen der virtuellen Maschine innerhalb der Ausführung des Testworkflows aufgerufen. Sie geben eine virtuelle Maschine aus der Bestandsliste des lokalen Orchestrator-Servers zum Umbenennen an. Da der Workflow zum Benennen der virtuellen Maschine auf dem entfernten Orchestrator-Server ausgeführt wird, muss dieselbe virtuelle Maschine in der Bestandsliste dieses Servers verfügbar sein. Andernfalls kann der Workflow zum Umbenennen der virtuellen Maschine den `vm`-Eingabeparameter nicht auflösen. Aus diesem Grund muss das vCenter Server-Plug-in auf dem lokalen und dem entfernten Orchestrator-Server mit derselben vCenter Server-Instanz verbunden sein.

## Gleichzeitiges Aufrufen mehrerer Workflows

Durch gleichzeitiges Aufrufen mehrerer Workflows wird der aufgerufene Workflow als Teil der Ausführung des aufrufenden Workflows synchron ausgeführt. Der aufrufende Workflow wartet auf die Beendigung aller aufgerufenen Workflows, bevor er fortgesetzt wird. Der aufrufende Workflow kann die Ergebnisse des aufgerufenen Workflows als Eingabeparameter bei der Ausführung der nachfolgenden Schemaelemente verwenden.

Sie können mehrere Workflows gleichzeitig über einen anderen Workflow aufrufen, indem Sie das Element **Verschachtelte Workflows** verwenden. Mithilfe verschachtelter Workflows können Sie Workflows mit Benutzeranmeldedaten ausführen, die sich von den Benutzeranmeldedaten des aufrufenden Workflows unterscheiden.

### Voraussetzungen

- Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.
- Fügen Sie dem Workflowschema einige Elemente hinzu.

### Vorgehensweise

- 1 Ziehen Sie das Element **Verschachtelte Workflows** aus dem Menü **Aktion und Workflow** an die entsprechende Position im Workflowschema.

Daraufhin wird das Auswahldialogfeld für **Workflow auswählen** angezeigt.

- 2 Suchen Sie nach einem zu startenden Workflow, wählen Sie ihn aus und klicken Sie auf **OK**.

- 3 Klicken Sie auf das Symbol **Bearbeiten** (✎) für das Element **Verschachtelte Workflows** im Workflowschema.

- 4 Klicken Sie auf die Registerkarte **Workflows**.

Daraufhin wird der Workflow, den Sie in [Schritt 2](#) ausgewählt haben, auf der Registerkarte angezeigt.

- 5 Legen Sie die EIN- und AUS-Bindungen für diesen Workflow auf den Registerkarten **EIN** und **AUS** im rechten Fensterbereich der Registerkarte „Eigenschaften“ für das Schemaelement **Workflows** fest.

- 6 Klicken Sie auf die Registerkarte **Verbindungsinfo** im rechten Fensterbereich der Registerkarte „Eigenschaften“ für das Schema-Element **Workflows**.

Über die Registerkarte **Verbindungsinfo** können Sie mithilfe der entsprechenden Anmeldedaten auf Workflows zugreifen, die auf einem anderen Server als dem lokalen gespeichert sind.

- 7 Wenn Sie auf Workflows auf einem Remoteserver zugreifen möchten, wählen Sie **Remote** aus und klicken Sie auf **Nicht festgelegt**, um einen Hostnamen oder eine IP-Adresse für den Remoteserver anzugeben.

---

**HINWEIS** Sie können das Multi-Node-Plug-In von vRealize Orchestrator zum Aufrufen von Workflows auf einem Remoteserver verwenden.

---

- 8 Legen Sie die Anmeldedaten fest, mit denen auf den Remoteserver zugegriffen werden soll.

- Wählen Sie **Übernehmen** aus, um dieselben Anmeldedaten wie der Benutzer zu verwenden, der den aufrufenden Workflow ausführt.
- Wählen Sie **Dynamisch** aus und klicken Sie auf **Nicht festgelegt**, um einen Satz dynamischer Anmeldedaten auszuwählen, die von einem Parameter des Typs `credentials` irgendwo im Workflow definiert werden.
- Wählen Sie **Statisch** aus, klicken Sie auf **Nicht festgelegt** und geben Sie die Anmeldedaten direkt ein.

- 9 Klicken Sie auf der Registerkarte **Workflows** auf die Schaltfläche **Workflow hinzufügen** und wählen Sie weitere Workflows aus, die dem verschachtelten Workflowelement hinzugefügt werden sollen.
- 10 Wiederholen Sie [Schritt 2](#) bis [Schritt 8](#) und legen Sie die Einstellungen für sämtliche von Ihnen hinzugefügten Workflows fest.
- 11 Klicken Sie auf das verschachtelte Workflowelement im Workflowschema.  
Die Anzahl verschachtelter Workflows im Element wird als Zahl im verschachtelten Workflowelement angezeigt.

Sie haben über einen Workflow mehrere Workflows gleichzeitig aufgerufen.

### Weiter

Sie können Workflows mit langer Ausführungsdauer definieren.

## Ausführen eines Workflows für eine Auswahl von Objekten

Sie können sich wiederholende Aufgaben durch Ausführen eines Workflows für eine Auswahl von Objekten automatisieren. Beispiele: Sie erstellen einen Workflow, der Snapshots aller virtuellen Maschinen in einem Ordner mit virtuellen Maschinen erstellt, oder Sie erstellen einen Workflow, der alle virtuellen Maschinen eines bestimmten Hosts einschaltet.

Zum Ausführen eines Workflows für eine Auswahl von Objekten stehen Ihnen folgende Methoden zur Verfügung:

- Führen Sie den folgenden Workflow aus: **Bibliothek > vCenter > Batch > Workflow für eine Auswahl von Objekten ausführen**.
- Erstellen Sie einen Workflow, der die folgenden Workflows aufruft: **Bibliothek > Orchestrator > Workflows nacheinander starten** oder **Workflows parallel starten**.
- Erstellen Sie einen Workflow, der ein Array von Objekten bezieht und für jedes Objekt des Arrays einen Workflow als Schleife von Workflowelementen durchführt.
- Führen Sie einen Workflow über JavaScript aus, indem Sie die Methode `Workflow.execute()` als For-Schleife in einem Skriptelement in einem Workflow ausführen.

Die Wahl der Methode zur Workflowausführung für eine Auswahl von Objekten hängt vom auszuführenden Workflow ab und kann sich auf dessen Leistung auswirken. Beispiel: Das Ausführen des Workflows „Ausführen“ für eine Auswahl von Objekten ist die einfachste Methode zum Ausführen eines Workflows für mehrere Objekte. Dabei muss kein Workflow entwickelt werden. Es können aber nur Workflows ausgeführt werden, die nur einen Parameter verwenden.

Durch das Erstellen eines Workflows, der den Workflow „Starten“ in einer Reihe von Start-Workflows in parallelen Workflows aufruft, können Sie Workflows für mehrere Objekte ausführen, die mehrere Eingabeparameter verwenden. Der aufrufende Workflow benötigt ein Eigenschaften-Array für die Übergabe der Eingabeparameter an die Start-Workflows in aufeinanderfolgenden oder parallelen Workflows. Diese Workflows sind nur zur Verwendung in anderen Workflows geeignet. Führen Sie sie nicht direkt aus.

Das Ausführen eines Workflows in einer For-Schleife in einem Skriptelement ist schneller als das Ausführen eines Workflows in einer Schleife von Workflowelementen. Es ist aber weniger flexible und beschränkt die Möglichkeiten zur Wiederverwendung. Wichtig: Beim Ausführen eines Workflows in einer Skriptschleife geht die Prüfpunktfunktion verloren, die Orchestrator beim Starten jedes Elements in einer Workflowausführung durchführt. Wenn also der Orchestrator-Server während der Ausführung einer Skriptschleife angehalten wird, wird der Workflow am Beginn des Skriptelements wiederaufgenommen und die gesamte Schleife wird erneut ausgeführt. Wird der Orchestrator-Server angehalten, während ein Workflow mit einer Schleife von Workflowelementen ausgeführt wird, wird der Workflow bei dem Element der Schleife fortgesetzt, das beim Anhalten des Servers gerade ausgeführt wurde.

Weitere Informationen zu Batchworkflows finden Sie unter *Verwenden von VMware vRealize Orchestrator-Plug-Ins*.

Das Erstellen eines Workflows, der einen Workflow für ein Array von Objekten in einer Schleife von Workf-lowelementen ausführt, wird unter „[Entwickeln eines komplexen Workflows](#)“, auf Seite 169 dargestellt.

Das Ausführen eines Workflows in einer For-Skriptschleife wird unter „[Beispiele für Workflow-Skripterstellung](#)“, auf Seite 206 erläutert.

## Implementieren der Workflows „Workflows nacheinander starten“ und „Workflows parallel starten“

Mithilfe von „Workflows nacheinander starten“ und „Workflows parallel starten“ kann ein Workflow für mehrere ausgewählte Objekte ausgeführt werden.

Die beiden Workflows „Workflows nacheinander starten“ und „Workflows parallel starten“ können nicht direkt ausgeführt werden. Stattdessen müssen sie in einen anderen Workflow eingeschlossen werden. Zur Anwendung von „Workflows nacheinander starten“ und „Workflows parallel starten“ auf mehrere ausgewählte Objekte müssen Sie die Objekte abrufen, für die der jeweilige Workflow ausgeführt werden soll. Diese Objekte und alle anderen Eingabeparameter, die der Workflow erfordert, müssen als Eigenschaften-Array übergeben werden. Bei „Workflows nacheinander starten“ und „Workflows parallel starten“ werden die ausgewählten Objekte als Array von WorkflowToken-Objekten ausgeführt.

Nacheinander und parallel gestartete Workflows werden auf dieselbe Art und Weise implementiert. Bei „Workflows nacheinander starten“ werden die Workflows für alle Objekte nacheinander ausgeführt. Bei „Workflows parallel starten“ werden die Workflows für alle Objekte gleichzeitig ausgeführt.

### Voraussetzungen

Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.

### Vorgehensweise

- 1 Fügen Sie dem Workflowschema ein Skriptaufgabenelement oder eine Skriptaktion hinzu, um eine Liste der Objekte zu erhalten, für die der Workflow ausgeführt werden soll.

Wenn die Ausführung zum Beispiel für alle virtuellen Maschinen in einem entsprechenden Ordner erfolgen soll, fügen Sie dem Workflow die Aktion `getAllVirtualMachinesByFolder` hinzu.

- 2 Verknüpfen Sie das Skriptelement oder die Skriptaktion und binden Sie das Element oder die Aktion an Workfloweingaben oder -attribute.

Binden Sie beispielsweise die `vmFolder`-Eingabe der Aktion `getAllVirtualMachinesByFolder` an einen Workflow-Eingabeparameter und die `actionResult`-Ausgabe an ein Workflowattribut im aufrufenden Workflow.

- 3 Fügen Sie ein Skriptaufgabenelement hinzu, um die Liste der Objekte in ein Eigenschaften-Array umzuwandeln.

Wenn es sich bei den Objekten, für die der Workflow ausgeführt werden soll, beispielsweise um ein Array von virtuellen Maschinen handelt (`allVMs`), das die `actionResult`-Ausgabe der Aktion `getAllVirtualMachinesByFolder` zurückgegeben hat, können die Objekte mithilfe des folgenden Skripts in ein Eigenschaften-Array umgewandelt werden.

```
propsArray = new Array();

for each (var vm in allVMs) {
    var prop = new Properties();
    prop.put("vm", vm);
    propsArray.push(prop);
}
```

- 4 Binden Sie die Ein- und Ausgabeattribute des Skriptaufgabenelements an Workflowattribute.  
Beim Skriptaufgabenelement in [Schritt 3](#) binden Sie beispielsweise das Eingabeattribut an das Array von virtuellen Maschinen (allVMs) und erstellen das Ausgabeattribut propsArray als Array von Properties-Objekten.
- 5 Fügen Sie dem Workflowschema ein Workflowelement hinzu.
- 6 Wählen Sie entweder „Workflows nacheinander starten“ oder „Workflows parallel starten“ aus und verknüpfen Sie das Workflowelement mit den anderen Elementen.
- 7 Binden Sie die wf-Eingabe des aufeinander folgenden oder parallelen Workflows an den Workflow, der für die Objekte ausgeführt werden soll.  
Wählen Sie beispielsweise zum Entfernen von Snapshots von allen virtuellen Maschinen, die von der Aktion getAllVirtualMachinesByFolder zurückgegeben wurden, den Workflow „Alle Snapshots entfernen“ aus.
- 8 Binden Sie die parameters-Eingabe des aufeinander folgenden oder parallelen Workflows an das Array von Properties-Objekten, für die der Workflow ausgeführt werden soll.  
Binden Sie beispielsweise die parameters-Eingabe an das propsArray-Attribut, das Sie in [Schritt 4](#) definiert haben.
- 9 (Optional) Binden Sie die workflowTokens-Ausgabe des aufeinander folgenden oder parallelen Workflows an ein Attribut des Workflows.
- 10 (Optional) Fügen Sie weitere Elemente hinzu, die die Ergebnisse des Workflows „Workflows nacheinander starten“ oder „Workflows parallel starten“ nutzen.

Sie haben einen Workflow erstellt, bei dem der Workflow „Workflows nacheinander starten“ oder „Workflows parallel starten“ für eine Auswahl von Objekten ausgeführt wird.

## Entwickeln von Workflows mit langer Ausführungsdauer

Ein Workflow in einem Wartezustand verbraucht Systemressourcen, da er ständig das Objekt abruft, von dem er eine Antwort benötigt. Wenn Sie wissen, dass ein Workflow möglicherweise lange Zeit wartet, bevor er die erforderliche Antwort erhält, können Sie ihm Workflowelemente mit langer Ausführungsdauer hinzufügen.

Jeder ausgeführte Workflow verbraucht einen Systemthread. Wenn ein Workflow ein solches Element erreicht, versetzt es den Workflow in einen passiven Zustand. Anschließend gibt es die Workflowinformationen an einen einzelnen Thread weiter, der alle laufenden Workflowelemente mit langer Ausführungsdauer auf dem Server abruft. Anstatt dass jedes Workflowelement mit langer Ausführungsdauer ständig versucht, Informationen vom System abzurufen, bleiben diese Elemente für eine festgesetzte Zeit passiv, während der Thread des Workflows mit langer Ausführungsdauer an seiner Stelle das System abfragt.

Sie legen die Dauer der Wartezeit auf eine der folgenden Arten fest:

- Legen Sie einen Timer, gekapselt in ein Date-Objekt, fest, der den Workflow bis zu einer bestimmten Uhrzeit und einem Datum aussetzt. Sie implementieren Workflowelement mit langer Ausführungsdauer, die auf einem Timer basieren, durch Hinzufügen des Elements **Warte-Timer** im Schema.
- Definieren Sie ein Auslöserereignis, gekapselt in ein Trigger-Objekt, das den Workflow bei Auftreten neu startet. Sie implementieren Workflowelement mit langer Ausführungsdauer, die auf einem Auslöser basieren, durch Hinzufügen des Elements **Warteereignis** oder des Elements **Benutzerinteraktion** im Schema.

## Setzen eines relativen Zeitpunkts oder Datums für Timer-basierte Workflows



Sie können das Attribut `timer.date` eines Warte-Timer-Elements auf einen relativen Zeitpunkt oder ein relatives Datum setzen, indem Sie es an ein Date-Objekt binden. Das Date-Objekt wird in einer Skriptfunktion definiert.

Wenn der Zeitpunkt an dem angegebenen Datum eintritt, wird der Workflow mit langer Ausführungszeit, der auf einem Timer basiert, reaktiviert und weiter ausgeführt. Beispiel: Sie können festlegen, dass der Workflow am 12. Februar mittags reaktiviert wird. Alternativ dazu können Sie ein Workflowelement erstellen, das ein relatives Date-Objekt gemäß einer von Ihnen definierten Funktion berechnet und generiert. Beispielsweise können Sie ein relatives Date-Objekt erstellen, das 24 Stunden zur momentanen Uhrzeit hinzufügt.

### Voraussetzungen

- Erstellen Sie einen Workflow.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.
- Fügen Sie dem Workflowschema einige Elemente hinzu.

### Vorgehensweise

- 1 Ziehen Sie ein **skriptfähiges Aufgabenelement** aus dem Menü **Generisch** in das Schema eines Workflows vor das Element, das das relative Objekt `Date` für sein Attribut `timeout.date` benötigt.
- 2 Klicken Sie auf das Symbol **Bearbeiten** () für das Element **Skriptfähige Aufgabe** im Workflowschema.
- 3 Geben Sie einen Namen und eine Beschreibung für das Element für skriptfähige Aufgaben auf der Eigenschaftenregisterkarte **Info** an.
- 4 Klicken Sie auf die Eigenschaftenregisterkarte **AUS** und auf das Symbol **An Workflowparameter/-attribut binden** ()
- 5 Klicken Sie auf **Parameter/Attribut in Workflow erstellen**, um ein Workflowattribut zu erstellen.
  - a Nennen Sie das Attribut `timerDate`
  - b Wählen Sie `Date` aus der Liste der Attributtypen.
  - c Wählen Sie **Workflow-ATTRIBUT mit demselben Namen erstellen**.
  - d Belassen Sie den Attributwert auf **Nicht festgelegt**, weil eine in einem Skript programmierte Funktion diesen Wert bereitstellen wird.
  - e Klicken Sie auf **OK**.
- 6 Klicken Sie auf die Registerkarte **Skripterstellung** des skriptfähigen Aufgabenelements.
- 7 Definieren Sie eine Funktion zum Berechnen und Regenerieren eines Objekts `Date` mit dem Namen `timerDate` im Skripterstellungsbereich auf der Registerkarte **Skripterstellung**.

Sie könnten beispielsweise ein Objekt `Date` erstellen, indem Sie die folgende JavaScript-Funktion implementieren, in der die Zeitüberschreitungsperiode eine Relativverzögerung in Millisekunden darstellt.

```
timerDate = new Date();
System.log( "Current date : " + timerDate + " " );
timerDate.setTime( timerDate.getTime() + (86400 * 1000) );
System.log( "Timer will expire at " + timerDate + " " );
```



Die JavaScript-Funktion im vorherigen Beispiel definiert ein Objekt `Date`, das das aktuelle Datum und die aktuelle Zeit ermittelt, indem die Methode `getTime` verwendet wird und 68.400.000 Millisekunden oder 24 Stunden hinzu addiert werden. Das **skriptfähige Aufgabenelement** generiert diesen Wert als seinen Ausgabeparameter.

- 8 Klicken Sie auf **Schließen**.
- 9 Klicken Sie auf **Speichern**.

Sie haben eine Funktion erstellt, die ein Objekt `Date` berechnet und generiert. Ein **Warte-Timer**-Element kann dieses Objekt `Date` als Eingabeparameter übernehmen, um einen Workflow mit langer Ausführungszeit bis zu dem Datum zu unterbrechen, das in diesem Objekt programmiert ist. Wenn der Workflow das **Warte-Timer**-Element erreicht, wird die Ausführung angehalten und der Workflow wartet 24 Stunden.

### Weiter

Sie müssen ein **Warte-Timer**-Element einem Workflow hinzufügen, um einen Workflow mit langer Ausführungszeit zu implementieren, der auf einem Timer basiert.

## Erstellen eines Timer-basierten Workflows mit langer Ausführungsdauer

Wenn Sie wissen, dass ein Workflow während einer vorhersehbaren Zeitspanne auf eine Antwort von einer externen Quelle warten muss, können Sie ihn als Timer-basierten Workflow mit langer Ausführungsdauer implementieren. Ein Timer-basierter Workflow mit langer Ausführungsdauer wartet bis zu einem bestimmten Zeitpunkt und einem bestimmten Datum, bevor er seine Ausführung wieder aufnimmt.

Sie implementieren einen Workflow als Timer-basierten Workflow mit langer Ausführungsdauer, indem Sie das Element **Warte-Timer** verwenden.

### Voraussetzungen

- Erstellen Sie einen Workflow.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.
- Fügen Sie dem Workflowschema einige Elemente hinzu.

### Vorgehensweise

- 1 Ziehen Sie ein **Warte-Timer**-Element aus dem Menü **Generisch** an die Stelle im Workflowschema, an der das Ausführen des Workflows angehalten werden soll.

Wenn Sie eine skriptfähige Aufgabe implementieren, um die Uhrzeit und das Datum zu berechnen, muss dieses Element vor dem Element **Warte-Timer** angeordnet sein.

- 2 Klicken Sie auf das Symbol **Bearbeiten** (✎) für das Element **Warte-Timer** im Workflowschema.
- 3 Geben Sie auf der Eigenschaftenregisterkarte **Info** eine Beschreibung des Grundes für die Implementierung des Timers ein.
- 4 Klicken Sie auf die Eigenschaftenregisterkarte **Attribute**.

Der Parameter `timer.date` erscheint in der Liste der Attribute.

- 5 Klicken Sie beim Parameter `timer.date` auf die Schaltfläche **Nicht festgelegt**, um den Parameter an ein geeignetes `Date`-Objekt zu binden.

Das Dialogfeld **Warte-Timer** für die Auswahl wird mit einer Liste von möglichen Bindungen geöffnet.

- Wählen Sie ein vordefiniertes `Date`-Objekt aus der Liste der Vorschläge, beispielsweise eines, das durch ein Element vom Typ **Skriptfähige Aufgabe** an anderer Stelle im Workflow definiert ist.
- Als Alternative erstellen Sie ein `Date`-Objekt, das ein bestimmtes Datum und eine bestimmte Uhrzeit festlegt, die vom Workflow abgewartet werden.

- 6 (Optional) Erstellen Sie ein Date-Objekt, das ein bestimmtes Datum und eine bestimmte Uhrzeit festlegt, die vom Workflow abgewartet werden.
  - a Klicken Sie auf **Parameter/Attribut in Workflow erstellen** im Auswahldialogfeld **Warte-Timer**.  
Das Dialogfeld **Parameterinformationen** wird geöffnet.
  - b Geben Sie dem Parameter einen passenden Namen.
  - c Lassen Sie den Typ auf Date eingestellt.
  - d Klicken Sie auf **Workflow-ATTRIBUT mit demselben Namen erstellen**.
  - e Klicken Sie bei der Eigenschaft **Wert** auf die Schaltfläche **Nicht festgelegt**, um den Parameterwert einzustellen.  
Ein Kalender wird eingeblendet.
  - f Verwenden Sie den Kalender, um ein Datum und eine Uhrzeit einzustellen, zu denen der Workflow wieder starten soll.
  - g Klicken Sie auf **OK**.
- 7 Klicken Sie auf **Schließen**.
- 8 Klicken Sie unten im Workfloweditor auf **Speichern**.

Sie haben einen Timer definiert, der einen Timer-basierten Workflow mit langer Ausführungszeit bis zu einem bestimmten Datum und Zeitpunkt anhält.

#### Weiter

Sie können einen Workflow mit langer Ausführungszeit erstellen, der auf ein Auslöseereignis wartet, bevor er fortgesetzt wird.

## Erstellen eines Auslöseobjekts

Auslöseobjekte überwachen Ereignisauslöser, die von Plug-Ins definiert werden. Das vCenter Server-Plug-In definiert beispielsweise diese Ereignisse als Task-Objekte. Wenn die Aufgabe endet, sendet der Auslöser eine Nachricht an ein wartendes, auslöserbasiertes Element für einen Workflow mit langer Ausführungszeit, damit der Workflow wieder gestartet wird.

Das Ereignis, das eine bestimmte Zeit braucht und auf das ein auslöserbasierter Workflow mit langer Ausführungszeit wartet, muss ein VC:Task-Objekt zurückgeben. Beispielsweise gibt die startVM-Aktion zum Starten einer virtuellen Maschine ein VC:Task-Objekt zurück, sodass nachfolgende Elemente in einem Workflow seinen Fortschritt beobachten können. Das Auslöseereignis eines auslöserbasierten Workflows mit langer Ausführungszeit erfordert dieses VC:Task-Objekt als Eingabeparameter.

Sie erstellen ein Trigger-Objekt in einer JavaScript-Funktion in einem Element vom Typ **Skriptfähige Aufgabe**. Dieses **Skriptfähige Aufgabe**-Element kann Teil des auslöserbasierten Workflows mit langer Ausführungszeit sein, der auf das Auslöseereignis wartet. Als Alternative kann es auch Teil eines anderen Workflows sein, der Eingabeparameter für den auslöserbasierten Workflow mit langer Ausführungszeit bereitstellt. Die Auslöserfunktion muss die createEndOfTaskTrigger()-Methode aus der Orchestrator-API implementieren.

---

**WICHTIG** Sie müssen eine Zeitüberschreitungsdauer für alle Auslöser definieren, da sonst der Workflow gegebenenfalls unbegrenzt wartet.




---

#### Voraussetzungen

- Erstellen Sie einen Workflow.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

- Fügen Sie dem Workflowschema einige Elemente hinzu.
- Deklarieren Sie im Workflow ein VC:Task-Objekt als Attribut oder Eingabeparameter, beispielsweise ein VC:Task-Objekt aus einem Workflow oder Workflowelement, der bzw. das eine virtuelle Maschine startet oder klonet.

### Vorgehensweise

- 1 Ziehen Sie ein **Skriptfähige Aufgabe**-Element aus dem Menü **Generisch** in das Schema eines Workflows.  
  
Eines der Elemente, die vor dem Element **Skriptfähige Aufgabe** stehen, muss ein VC:Task-Objekt als seinen Ausgabeparameter generieren.
- 2 Klicken Sie auf das Symbol **Bearbeiten** () für das Element **Skriptfähige Aufgabe** im Workflowschema.
- 3 Geben Sie einen Namen und eine Beschreibung für den Auslöser in der Eigenschaftenregisterkarte **Info** an.
- 4 Klicken Sie auf die Eigenschaftenregisterkarte **EIN**.
- 5 Klicken Sie auf das Symbol **An Workflowparameter/-attribut binden** ().  
  
Das Dialogfeld für die Auswahl des Eingabeparameters wird geöffnet.
- 6 Wählen Sie einen Eingabeparameter des Typs **VC:Task** aus oder erstellen Sie einen.  
  
Dieses VC:Task-Objekt stellt das eine bestimmte Zeitdauer benötigende Ereignis dar, das durch einen anderen Workflow oder ein anderes Element gestartet wird.
- 7 (Optional) Wählen Sie einen Eingabeparameter des Typs „Zahl“ aus oder erstellen Sie einen, um eine Zeitüberschreitungsperiode in Sekunden zu definieren.
- 8 Klicken Sie auf die Eigenschaftenregisterkarte **AUS**.
- 9 Klicken Sie auf das Symbol **An Workflowparameter/-attribut binden** ().  
  
Das Dialogfeld für die Auswahl des Ausgabeparameters wird geöffnet.
- 10 Erstellen Sie einen Ausgabeparameter mit den folgenden Eigenschaften.
  - a Erstellen Sie die Eigenschaft „Name“ mit dem Wert `trigger`.
  - b Erstellen Sie die Eigenschaft „Typ“ mit dem Wert `Trigger`.
  - c Klicken Sie auf **ATTRIBUT mit demselben Namen erstellen**, um das Attribut zu erstellen.
  - d Belassen Sie den Wert bei **Nicht festgelegt**.
- 11 Definieren Sie ein Ausnahmeverhalten auf der Eigenschaftenregisterkarte **Ausnahmen**.
- 12 Definieren Sie eine Funktion, um ein Trigger-Objekt auf der Registerkarte **Skripterstellung** zu generieren.  
  
Sie könnten beispielsweise ein Trigger-Objekt erstellen, indem Sie die folgende JavaScript-Funktion implementieren.  
  

```
trigger = task.createEndOfTaskTrigger(timeout);
```

  
Die Methode `createEndOfTaskTrigger()` gibt ein Trigger-Objekt zurück, das ein VC:Task-Objekt mit dem Namen `task` überwacht.
- 13 Klicken Sie auf **Schließen**.
- 14 Klicken Sie unten im Workfloweditor auf **Speichern**.

Sie haben ein Workflowelement definiert, das ein Auslöseereignis für einen auslöserbasierten Workflow mit langer Ausführungszeit erstellt. Das Auslöseelement generiert als Ausgabeparameter ein Trigger-Objekt, an das ein **Warteereignis**-Element gebunden werden kann.

### Weiter

Sie müssen dieses Auslöseereignis an ein **Warteereignis**-Element in einem auslöserbasierten Workflow mit langer Ausführungszeit binden.

## Erstellen eines auslöserbasierten Workflows mit langer Ausführungszeit

Wenn Sie wissen, dass ein Workflow während seiner Ausführungszeit auf eine Antwort von einer externen Quelle warten muss, aber nicht wissen, wie lange diese Wartezeit ist, können Sie ihn als auslöserbasierten Workflow mit langer Ausführungsdauer implementieren. Ein auslöserbasierter Workflow mit langer Ausführungszeit wartet darauf, dass ein definiertes Auslöseereignis eintritt, bevor die Ausführung wiederaufgenommen wird.

Sie implementieren einen Workflow als auslöserbasierten Workflow mit langer Ausführungsdauer, indem Sie das Element **Warteereignis** verwenden. Wenn der auslöserbasierte Workflow mit langer Ausführungszeit ein **Warteereignis**-Element erreicht, unterbricht er seine Ausführung und wartet im passiven Status, bis er eine Nachricht von dem Auslöser erhält. Während der Wartezeit nimmt der passive Workflow den Thread nicht wieder auf, sondern das Element des Workflows mit langer Ausführungszeit übergibt die Workflowinformationen an den einzelnen Thread, der alle Workflows mit langer Ausführungszeit auf dem Server überwacht.

### Voraussetzungen

- Erstellen Sie einen Workflow.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.
- Fügen Sie dem Workflowschema einige Elemente hinzu.
- Definieren Sie ein Auslöseereignis, das in einem Trigger-Objekt eingeschlossen ist.

### Vorgehensweise

- 1 Ziehen Sie ein **Warteereignis**-Element aus dem Menü **Generisch** an die Stelle im Workflowschema, an der die Workflowausführung angehalten werden soll.  
Die skriptfähige Aufgabe, die den Auslöser deklariert, muss unmittelbar vor dem **Warteereignis**-Element platziert sein.
- 2 Klicken Sie auf das Symbol **Bearbeiten** (✎) für das Element **Warteereignis** im Workflowschema.
- 3 Geben Sie auf der Eigenschaftenregisterkarte **Info** eine Beschreibung des Grunds für die Wartezeit ein.
- 4 Klicken Sie auf die Eigenschaftenregisterkarte **Attribute**.  
Der Parameter `trigger.ref` erscheint in der Liste der Attribute.
- 5 Klicken Sie beim Parameter `trigger.ref` auf den Link **Nicht festgelegt**, um den Parameter an ein geeignetes Trigger-Objekt zu binden.  
Das Dialogfeld **Warteereignis** für die Auswahl wird mit einer Liste von möglichen Parametern für Bindungen geöffnet.
- 6 Wählen Sie ein vordefiniertes Trigger-Objekt aus der Liste mit den Vorschlägen aus.  
Dieses Trigger-Objekt stellt das Auslöseereignis dar, das durch einen anderen Workflow oder ein anderes Workflowelement definiert wird.
- 7 Definieren Sie ein Ausnahmeverhalten auf der Eigenschaftenregisterkarte **Ausnahmen**.

- 8 Klicken Sie auf **Schließen**.
- 9 Klicken Sie unten im Workfloweditor auf **Speichern**.

Sie haben ein Workflowelement definiert, das einen auslöserbasierten Workflow mit langer Ausführungszeit anhält, der vor dem Neustart auf ein bestimmtes Auslöseereignis wartet.

#### Weiter

Sie können einen Workflow ausführen.

## Konfigurationselemente

Ein Konfigurationselement ist eine Liste von Attributen, mit deren Hilfe Sie Konstanten über eine gesamte Orchestrator-Serverbereitstellung hinweg konfigurieren können.

Alle Workflows, Aktionen und Richtlinien, die auf einem bestimmten Orchestrator-Server ausgeführt werden, können die von Ihnen in einem Konfigurationselement festgelegten Attribute verwenden. Durch Festlegen von Attributen in Konfigurationselementen können Sie dieselben Attributwerte allen Workflows, Aktionen und Richtlinien, die auf dem Orchestrator-Server ausgeführt werden, zur Verfügung stellen.

Wenn Sie ein Paket erstellen, das einen Workflow, eine Aktion oder Richtlinie enthält, der/die ein Attribut aus einem Konfigurationselement verwendet, wird das Konfigurationselement von Orchestrator automatisch in das Paket eingefügt. Wenn Sie ein Paket, das ein Konfigurationselement enthält, auf einen anderen Orchestrator-Server importieren, können Sie die Attributwerte des Konfigurationselements ebenso importieren. Beispiel: Sie erstellen einen Workflow, der Attributwerte benötigt, welche vom Orchestrator-Server, auf dem er ausgeführt wird, abhängige sind. Wenn Sie diese Attribute in einem Konfigurationselement festlegen, können Sie den Workflow exportieren, sodass ein anderer Orchestrator-Server ihn verwenden kann. Dadurch können Sie Workflows, Aktionen und Richtlinien mithilfe von Konfigurationselementen einfacher zwischen Servern austauschen.

---

**HINWEIS** Sie können keine Werte eines Konfigurationselementattributs aus einem Konfigurationselement importieren, das aus Orchestrator 5.1 oder älter exportiert wurde.



---

## Erstellen von Konfigurationselementen

Mithilfe von Konfigurationselementen können Sie häufige Attribute für einen Orchestrator-Server festlegen. Alle Elemente, die auf dem Server ausgeführt werden, können die von Ihnen in einem Konfigurationselement festgelegten Attribute aufrufen. Durch Erstellen von Konfigurationselementen können Sie häufige Elemente einmal auf dem Server definieren, anstatt es in jedem Element einzeln tun zu müssen.

#### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Konfigurationen**.
- 3 Klicken Sie in der hierarchischen Liste der Ordner mit der rechten Maustaste auf einen Ordner und wählen Sie **Neuer Ordner**, um einen neuen Ordner zu erstellen.
- 4 Geben Sie einen Namen für den Ordner an und klicken Sie auf **OK**.
- 5 Klicken Sie mit der rechten Maustaste auf den von Ihnen erstellten Ordner und wählen Sie **Neues Element**.
- 6 Geben Sie einen Namen für das Konfigurationselement an und klicken Sie auf **OK**.  
Der Editor für Konfigurationselemente wird geöffnet.
- 7 Inkrementieren Sie die Versionsnummer, indem Sie auf der Registerkarte **Allgemein** auf die Versionsziffern klicken und einen Kommentar zur Version eingeben.

- 8 Geben Sie eine Beschreibung des Konfigurationselements in das Textfeld **Beschreibung** auf der Registerkarte **Allgemein** ein.
- 9 Klicken Sie auf die Registerkarte **Attribute**.
- 10 Klicken Sie auf das Symbol **Attribut hinzufügen** () , um ein neues Attribut hinzuzufügen.
- 11 Klicken Sie auf die Attributwerte unter **Name**, **Typ**, **Wert** und **Beschreibung**, um den Namen, Typ, Wert und die Beschreibung des Attributs festzulegen.
- 12 Klicken Sie auf die Registerkarte **Berechtigungen**.
- 13 Klicken Sie auf das Symbol **Zugriffsrechte hinzufügen** () , um einer Gruppe von Benutzern die Berechtigung für den Zugriff auf dieses Konfigurationselement zu gewähren.
- 14 Suchen Sie im Textfeld **Filter** nach einer Benutzergruppe und wählen Sie die entsprechende Benutzergruppe aus der vorgeschlagenen Liste aus.
- 15 Aktivieren Sie die entsprechenden Kontrollkästchen zum Festlegen der Zugriffsrechte für die ausgewählte Benutzergruppe.

Sie können die folgenden Berechtigungen für das Konfigurationselement festlegen.

Berechtigung	Beschreibung
<b>Anzeigen</b>	Benutzer können das Konfigurationselement anzeigen, nicht jedoch Schemas oder Skripterstellung.
<b>Überprüfen</b>	Benutzer können das Konfigurationselement einschließlich der Schemas und Skripterstellung anzeigen.
<b>Admin</b>	Benutzer können Berechtigungen für die Elemente im Konfigurationselement festlegen und verfügen über alle anderen Berechtigungen.
<b>Ausführen</b>	Benutzer können die Elemente im Konfigurationselement ausführen.
<b>Bearbeiten</b>	Benutzer können die Elemente im Konfigurationselement bearbeiten.

- 16 Klicken Sie auf **Auswählen**.
- 17 Klicken Sie auf **Speichern und schließen**, um den Editor für das Konfigurationselement zu beenden.

Sie haben ein Konfigurationselement definiert, das häufige Attribute für einen Orchestrator-Server festlegt.

#### Weiter

Mithilfe des Konfigurationselements können Sie Attribute für Workflows oder Aktionen bereitstellen.

## Benutzerberechtigungen für Workflows

Orchestrator definiert Berechtigungsebenen, die Sie auf Gruppen anwenden können, um Ihnen den Zugriff auf Workflows zu gestatten oder zu verweigern.

<b>Ansicht</b>	Der Benutzer kann die Elemente im Workflow sehen, aber nicht die Schemas oder die Skripte.
<b>Überprüfen</b>	Der Benutzer kann die Elemente im Workflow einschließlich der Schemas oder Skripte sehen.
<b>Ausführen</b>	Der Benutzer können den Workflow ausführen.
<b>Bearbeiten</b>	Der Benutzer kann den Workflow bearbeiten.
<b>Admin</b>	Der Benutzer kann Berechtigungen für den Workflow einrichten und verfügt über alle anderen Berechtigungen.

Die Berechtigung **Admin** umfasst die Berechtigungen für **Ansicht**, **Überprüfen**, **Bearbeiten** und **Ausführen**. Alle Berechtigungen erfordern die Berechtigung **Ansicht**.

Wenn Sie keine Berechtigungen für einen Workflow einrichten, erbt der Workflow die Berechtigungen des Ordners, indem er sich befindet. Wenn Sie Berechtigungen für einen Workflow einrichten, überschreiben diese Berechtigungen die Berechtigungen des Ordners, indem er enthalten ist, auch wenn die Berechtigungen des Ordners einschränkender sind.

## Setzen der Benutzerberechtigungen für einen Workflow


Sie legen Berechtigungsebenen für einen Workflow fest, um den Zugriff von Benutzergruppen auf diesen Workflow zu begrenzen.

Sie können die Benutzer und Benutzergruppen, für die Berechtigungen erstellt werden, aus dem Orchestrator LDAP auswählen.

### Voraussetzungen

- Erstellen Sie einen Workflow.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.
- Fügen Sie dem Workflowschema einige Elemente hinzu.

### Vorgehensweise

- 1 Klicken Sie auf die Registerkarte **Berechtigungen**.
- 2 Klicken Sie auf das Symbol **Zugriffsrechte hinzufügen** () , um Berechtigungen für eine neue Benutzergruppe festzulegen.
- 3 Suchen Sie nach einer Benutzergruppe.  
Die Suchergebnisse enthalten alle Benutzergruppen aus dem Orchestrator LDAP-Server, die mit den Suchkriterien übereinstimmen.
- 4 Wählen Sie eine Benutzergruppe aus und aktivieren Sie die entsprechenden Kontrollkästchen zum Festlegen der Berechtigungsstufe der Benutzergruppe.  
Um einem Benutzer aus dieser Benutzergruppe zu erlauben, den Workflow anzuzeigen, das Schema und die Skripte zu überprüfen, den Workflow auszuführen und zu bearbeiten sowie die Berechtigungen zu ändern, müssen Sie alle Kontrollkästchen aktivieren.
- 5 Klicken Sie auf **Auswählen**.  
Die Benutzergruppe erscheint in der Liste der Berechtigungen.
- 6 Klicken Sie auf **Speichern und schließen**, um den Texteditor zu beenden.

## Validieren von Workflows

Orchestrator bietet ein Tool zur Workflowvalidierung. Durch das Validieren eines Workflows können Sie Fehler im Workflow erkennen und überprüfen, ob die Daten von einem Element zum nächsten korrekt fließen.

Wenn Sie einen Workflow validieren, erstellt das Validierungstool eine Liste der Fehler oder Warnungen. Durch einen Klick auf einen Fehler in der Liste wird das Workflowelement markiert, in dem der Fehler aufgetreten ist.

Wenn Sie das Validierungstool im Workfloweditor ausführen, schlägt das Tool schnell Korrekturen für die erkannten Fehler vor. Einige Schnellkorrekturen erfordern die Eingabe von zusätzlichen Informationen oder Eingabeparametern. Andere Schnellkorrekturen beheben den Fehler an Ihrer Stelle.

Die Workflowvalidierung überprüft die Datenbindungen und Verbindungen zwischen Elementen. Die Workflowvalidierung prüft nicht die Datenverarbeitung, die jedes Element im Workflow durchführt. Daher kann ein gültiger Workflow fehlerhaft ausgeführt werden und fehlerhafte Ergebnisse produzieren, wenn eine Funktion in einem Schemaelement falsch ist.

Standardmäßig führt Orchestrator immer eine Workflowvalidierung durch, wenn ein Workflow ausgeführt wird. Sie können das Standortvalidierungsverhalten im Orchestrator-Client ändern. Siehe „[Testen von Workflows während der Entwicklung](#)“, auf Seite 16. Beispiel: Während der Workflowentwicklung möchten Sie aus Testzwecken gelegentlich einen Workflow ausführen, von dem Sie wissen, dass er ungültig ist.

## Validieren eines Workflows und Behebung von Validierungsfehlern

Sie müssen einen Workflow validieren, bevor Sie ihn ausführen können. Sie können Workflows entweder im Orchestrator-Client oder im Workfloweditor validieren. Validierungsfehler können Sie aber nur beheben, wenn Sie den Workflow im Workfloweditor zur Bearbeitung geöffnet haben.

### Voraussetzungen

Überprüfen Sie, ob Sie einen vollständigen Workflow zum Validieren haben, in dem Schemaelemente verknüpft und Bindungen definiert sind.

### Vorgehensweise

- 1 Klicken Sie auf die Ansicht **Workflows**.
- 2 Navigieren Sie zu einem Workflow in der hierarchischen Liste der **Workflows**.
- 3 (Optional) Klicken Sie mit der rechten Maustaste auf den Workflow und wählen Sie **Workflow validieren**.

Wenn der Workflow gültig ist, erscheint eine Bestätigungsmeldung. Wenn der Workflow ungültig ist, erscheint eine Liste der Fehler.

- 4 (Optional) Schließen Sie das Dialogfeld für die Workflowvalidierung.
- 5 Klicken Sie mit der rechten Maustaste auf den Workflow und wählen Sie **Bearbeiten** aus, um den Workfloweditor zu öffnen.
- 6 Klicken Sie auf die Registerkarte **Schema**.
- 7 Klicken Sie auf die Schaltfläche **Validieren** in der Symbolleiste der Registerkarte **Schema**.

Wenn der Workflow gültig ist, erscheint eine Bestätigungsmeldung. Wenn der Workflow ungültig ist, erscheint eine Liste der Fehler.

- 8 Bei einem ungültigen Workflow klicken Sie auf eine Fehlermeldung.

Das Validierungstool markiert das Schemaelement mit dem Fehler durch ein rotes Symbol. Wenn möglich, zeigt das Validierungstool auch eine Aktion zur schnellen Problembehebung.

- Wenn Sie mit der Aktion zur schnellen Problembehebung einverstanden sind, klicken Sie darauf, um diese Aktion vorzunehmen.
- Wenn Sie mit der vorgeschlagenen Aktion zur schnellen Problembehebung nicht einverstanden sind, schließen Sie das Dialogfeld für die Workflowvalidierung und beheben Sie das Problem des Schemaelements manuell.

---

**WICHTIG** Prüfen Sie immer, ob die von Orchestrator vorgeschlagene Problembehebung geeignet ist.

---

Beispielsweise kann die vorgeschlagene Aktion das Löschen eines nicht benötigten Attributs sein, während das wirkliche Problem in einer fehlerhaften Bindung des Attributs liegt.

- 9 Wiederholen Sie die vorhergehenden Schritte, bis Sie alle Validierungsfehler eliminiert haben.



Sie haben einen Workflow validiert und Validierungsfehler behoben.

### Weiter

Sie können den Workflow jetzt ausführen.

## Debuggen von Workflows

Orchestrator bietet ein Tool zum Debuggen von Workflows. Sie können einen Workflow debuggen, um Ein- und Ausgabeparameter und Attribute beim Starten von Aktivitäten zu überprüfen, Parameter- oder Attributwerte bei Ausführung eines Workflows im Bearbeitungsmodus ersetzen und Workflows ab der letzten fehlgeschlagenen Aktivität fortsetzen.

Sie können Workflows aus der Standardworkflow-Bibliothek sowie benutzerdefinierte Workflows debuggen. Sie können benutzerdefinierte Workflows debuggen, während Sie diese im Workfloweditor entwickeln.

## Debuggen eines Workflows

Sie können Elemente eines Workflows debuggen, indem Sie den Elementen im Workflowschema Unterbrechungspunkte hinzufügen.

Bei Erreichen eines Unterbrechungspunkts haben Sie verschiedene Optionen, den Debugging-Vorgang fortzusetzen. Wenn Sie ein Element aus einem Workflowschema debuggen, können Sie allgemeine Informationen zur Ausführung des Workflows anzeigen, die Workflowvariablen ändern und Protokollmeldungen anzeigen.

### Voraussetzungen

Melden Sie sich beim Orchestrator-Client als Benutzer an, der Workflows ausführen kann.

### Vorgehensweise



- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Workflows**.
- 3 Wählen Sie einen Workflow aus der Workflowbibliothek aus und klicken Sie auf die Registerkarte **Schema**.
- 4 Um dem für das Debuggen vorgesehenen Schemaelement Unterbrechungspunkte hinzuzufügen, klicken Sie mit der rechten Maustaste auf ein Workflowelement und wählen Sie **Unterbrechungspunkt umschalten**.



Sie können die umgeschalteten Unterbrechungspunkte aktivieren bzw. deaktivieren.

- 5 Klicken Sie auf das Symbol **Workflow debuggen** ()

Wenn der Workflow Eingabeparameter erfordert, müssen Sie diese angeben.

- 6 Wenn die Ausführung des Workflows nach Erreichen eines Unterbrechungspunkts angehalten wird, wählen Sie eine der verfügbaren Optionen.

Option	Beschreibung
 <b>Fortsetzen</b>	Die Ausführung des Workflows wird fortgesetzt, bis ein anderer Unterbrechungspunkt erreicht wird.
 <b>Springen zu</b>	Ermöglicht das Springen zu einem Workflowelement. <b>HINWEIS</b> Sie können nicht zu einem verschachtelten Workflowelement springen, wenn Sie einen Workflow im Workfloweditor bearbeiten.

Option	Beschreibung
 <b>Überspringen</b>	Überspringt das aktuelle Element im Schema und hält die Ausführung des Workflows beim nächsten Element an.
 <b>Zurückspringen</b>	Das Workflowelement, zu dem Sie gesprungen sind, wird verlassen.

- 7 (Optional) Ändern Sie über die Registerkarte **Unterbrechungspunkte** die Unterbrechungspunkte. Sie können vorhandene Unterbrechungspunkte aktivieren, deaktivieren oder entfernen.
- 8 (Optional) Überprüfen Sie die Variablen auf der Registerkarte **Variablen**. Sie können die Werte einiger Variablen während des Debuggens ändern.

## Beispiel: Workflow-Debugging




Sie können Workflows aus der Standardworkflow-Bibliothek debuggen.

Bei Angabe einer falschen Empfängeradresse können Sie den Wert beispielsweise durch Debuggen der Beispielinteraktion für E-Mail-Workflows korrigieren.

### Voraussetzungen

Melden Sie sich beim Orchestrator-Client als Benutzer an, der Mail-Workflows ausführen kann.

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Workflows**.
- 3 Öffnen Sie in der hierarchischen Liste der Workflows den Pfad **Bibliothek > E-Mail**.
- 4 Wählen Sie die Beispielinteraktion für E-Mail-Workflows aus und klicken Sie auf die Registerkarte **Schema**.
- 5 Klicken Sie mit der rechten Maustaste auf das Workflowelement **E-Mail senden (Interaktion)** und wählen Sie **Unterbrechungspunkt umschalten**.
- 6 Klicken Sie auf das Symbol **Workflow debuggen** ().
- 7 Geben Sie die erforderlichen Informationen an.
  - a Geben Sie in das Textfeld **Zieladresse** eine unvollständige Empfängeradresse ein.  
Zum Beispiel **Name@Unternehmen.c**.
  - b Wählen Sie eine LDAP-Benutzergruppe aus, die zum Beantworten der Abfrage berechtigt sind.
  - c Klicken Sie auf **Senden**.
- 8 Sobald der Unterbrechungspunkt erreicht ist, klicken Sie auf das Symbol **Springen zu** (.
- 9 Überprüfen Sie die Werte auf der Registerkarte **Variablen**.
- 10 Geben Sie die richtige Empfängeradresse in das Textfeld **Empfängeradresse** ein.  
Zum Beispiel **Name@Unternehmen.c**.
- 11 Klicken Sie zum Fortfahren der Ausführung des Workflows auf das Symbol **Fortsetzen** (.

Der Workflow verwendet den beim Debugging-Vorgang angegebenen Wert und fährt mit der Ausführung des Workflows fort.

## Ausführen von Workflows

Ein Orchestrator-Workflow läuft nach einem logischen Ereignisfluss ab.

Wenn Sie einen Workflow ausführen, läuft jedes Schemaelement im Workflow gemäß der folgenden Sequenz ab.

- 1 Der Workflow bindet die Workflowtokenattribute und Eingabeparameter an die Eingabeparameter des Schemaelements.
- 2 Das Schemaelement wird ausgeführt.
- 3 Die Ausgabeparameter des Schemaelements werden in die Workflowtokenattribute und in die Eingabeparameter des Workflows kopiert.
- 4 Die Workflowtokenattribute und die Eingabeparameter werden in der Datenbank gespeichert.
- 5 Das nächste Schemaelement beginnt seine Ausführung.

Diese Sequenz wird für jedes Schemaelement wiederholt, bis der Workflow endet.

### Checkpoint des Workflowtokens

Wenn ein Workflow ausgeführt wird, ist jedes Schemaelement ein Checkpoint. Nachdem ein Schemaelement ausgeführt ist, speichert Orchestrator die Workflowtokenattribute in der Datenbank und das nächste Schemaelement beginnt mit dem Ausführen. Wenn der Workflow unerwartet anhält, wird das gerade aktive Schema beim nächsten Neustart des Orchestrator-Servers erneut ausgeführt und der Workflow beginnt mit dem Start des Schemaelements, das ausgeführt wurde, als die Unterbrechung eintrat. Orchestrator informiert jedoch keine Transaktionsverwaltung oder eine Wiederherstellungsfunktion.

### Ende des Workflows

Der Workflow endet, wenn das aktuelle aktive Schemaelement ein Endelement ist. Nachdem der Workflow ein Endelement erreicht hat, können die Ausgabeparameter des Workflows von anderen Workflows oder Anwendungen genutzt werden.

## Ausführen eines Workflows im Workfloweditor

Sie können einen Workflow ausführen, während Sie ihn entwickeln.

Durch das Ausführen eines Workflows im Workfloweditor können Sie prüfen, ob der Workfloweditor ordnungsgemäß ausgeführt wird, ohne dabei den Entwicklungsvorgang unterbrechen zu müssen. Informationen zur Workflowausführung werden in Protokollmeldungen erfasst. Wenn der Workflow ein nicht erwartetes Ergebnis zurückgibt, können Sie ihn ändern und erneut ausführen, ohne den Workfloweditor schließen zu müssen.

#### Voraussetzungen

- Erstellen Sie einen Workflow.
- Öffnen Sie den Workflow für die Bearbeitung im Workfloweditor.
- Validieren Sie den Workflow.

#### Vorgehensweise

- 1 Klicken Sie auf die Registerkarte **Schema**.
- 2 Klicken Sie auf **Ausführen**.
- 3 (Optional) Überprüfen Sie die Meldungen auf der Registerkarte **Protokolle**.

## Ausführen eines Workflows

Sie können automatisierte Vorgänge in vCenter Server ausführen, indem Sie Workflows aus der Standardbibliothek oder von Ihnen erstellte Workflows ausführt.

Sie können beispielsweise eine virtuelle Maschine erstellen, indem Sie den Workflow „Einfache virtuelle Maschine erstellen“ ausführen.

### Voraussetzungen

Vergewissern Sie sich, dass Sie das vCenter Server-Plug-In konfiguriert haben. Details finden Sie unter *Installieren und Konfigurieren von VMware vCenter Orchestrator*.

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü des Orchestrator-Clients **Ausführen** oder **Design** aus.
- 2 Klicken Sie auf die Ansicht **Workflows**.
- 3 Öffnen Sie in der hierarchischen Liste der Workflows **Bibliothek > vCenter > Verwaltung von virtuellen Maschinen > Einfach**, um zum Workflow „Einfache virtuelle Maschine erstellen“ zu navigieren.
- 4 Klicken Sie mit der rechten Maustaste auf den Workflow „Einfache virtuelle Maschine erstellen“ und wählen Sie **Workflow starten**.
- 5 Geben Sie die folgenden Informationen im Eingabeparameter-Dialogfeld **Workflow starten** ein, um eine virtuelle Maschine in einem vCenter Server zu erstellen, der mit Orchestrator verbunden ist.

Option	Aktion
<b>Name der virtuellen Maschine</b>	Benennen Sie die virtuelle Maschine als <b>Orchestrator-Test</b> .
<b>Ordner der virtuellen Maschine</b>	a Klicken Sie auf <b>Nicht festgelegt</b> für den Wert <b>Ordner der virtuellen Maschine</b> . b Wählen Sie einen Ordner für eine virtuelle Maschine aus dem Bestand. Die Schaltfläche <b>Auswählen</b> ist inaktiv, bis Sie ein Objekt des richtigen Typs ausgewählt haben, in diesem Fall ist dies VC:VmFolder.
<b>Größe der neuen Festplatte in GB</b>	Geben Sie einen entsprechenden Zahlenwert ein.
<b>Arbeitsspeichergröße in MB</b>	Geben Sie einen entsprechenden Zahlenwert ein.
<b>Anzahl der virtuellen CPUs</b>	Wählen Sie eine geeignete Anzahl von CPUs aus dem Dropdown-Menü <b>Anzahl der virtuellen CPUs</b> .
<b>Gastbetriebssystem der virtuellen Maschine</b>	Klicken Sie auf den Link <b>Nicht festgelegt</b> und wählen Sie ein Gastbetriebssystem aus der Liste.
<b>Host für die Erstellung der virtuellen Maschine</b>	Klicken Sie auf <b>Nicht festgelegt</b> für den Wert <b>Host für die Erstellung der virtuellen Maschine</b> und navigieren Sie über die vCenter Server-Infrastrukturhierarchie bis zu einer Hostmaschine.
<b>Ressourcenpool</b>	Klicken Sie auf <b>Nicht festgelegt</b> für den Wert <b>Ressourcenpool</b> und navigieren Sie über die vCenter Server-Infrastrukturhierarchie bis zu einem Ressourcenpool.
<b>Netzwerk zur Verbindung</b>	Klicken Sie auf <b>Nicht festgelegt</b> für den Wert <b>Netzwerk zur Verbindung</b> und wählen Sie ein Netzwerk aus. Drücken Sie die Eingabetaste im Textfeld <b>Filter</b> , um alle verfügbaren Netzwerke zu sehen.
<b>Datenspeicher, in dem die Dateien für die virtuelle Maschine gespeichert werden sollen</b>	Klicken Sie auf <b>Nicht festgelegt</b> für den Wert <b>Datenspeicher, in dem die Dateien für die virtuelle Maschine gespeichert werden sollen</b> und navigieren Sie über die vCenter Server-Infrastrukturhierarchie bis zu einem Datenspeicher.

- 6 Klicken Sie auf **Übernehmen**, um den Workflow auszuführen.  
Ein Workflowtoken erscheint unter dem Workflow „Einfache virtuelle Maschine erstellen“ mit einem Symbol für einen Workflow, der ausgeführt wird.
- 7 Klicken Sie auf den Workflowtoken, um den Status des Workflows während des Ausführens zu sehen.
- 8 Klicken Sie auf die Registerkarte **Ereignisse** in der Workflowtokenansicht, um den Fortschritt des Workflowtokens bis zu seinem Abschluss zu verfolgen.
- 9 Klicken Sie auf die Ansicht **Bestandsliste**.
- 10 Navigieren Sie über die vCenter Server-Infrastrukturhierarchie zu dem von Ihnen definierten Ressourcenpool.  
  
Wenn die virtuelle Maschine in der Liste nicht erscheint, klicken Sie auf die Schaltfläche zum Aktualisieren, um die Bestandsliste erneut zu laden.  
  
Die virtuelle Maschine `orchestrator-test` ist im Ressourcenpool vorhanden.
- 11 (Optional) Klicken Sie mit der rechten Maustaste auf die virtuelle Maschine `orchestrator-test` in der Ansicht **Bestand**, um eine kontextbezogene Liste der Workflows zu sehen, die Sie auf der virtuellen Maschine `orchestrator-test` ausführen können.

Der Workflow „Einfache virtuelle Maschine erstellen“ wurde erfolgreich ausgeführt.

#### Weiter

Sie können sich beim vSphere Client anmelden und die neue virtuelle Maschine verwalten.

## Fortsetzen einer fehlgeschlagenen Workflowausführung

Falls ein Workflow fehlschlägt, bietet Orchestrator eine Option, den Workflow ab der letzten fehlgeschlagenen Aktivität fortzusetzen.

Sie können die Parameter des Workflows ändern und versuchen, ihn fortzusetzen, oder die Parameter beibehalten und Änderungen an externen Komponenten mit Auswirkungen auf die Workflowausführung vornehmen. Wenn eine Workflowausführung beispielsweise aufgrund eines Problems in einem Drittanbietersystem fehlschlägt, können Sie Änderungen an dem System vornehmen und den Workflow ab der fehlgeschlagenen Aktivität fortsetzen, ohne die Workflowparameter zu ändern oder die erfolgreichen Aktivitäten zu wiederholen.

## Festlegen des Verhaltens für das Fortsetzen einer fehlgeschlagenen Workflowausführung

Sie können das Verhalten für das Fortsetzen einer fehlgeschlagenen Workflowausführung für jeden benutzerdefinierten Workflow festlegen. Die Standardworkflows in der Bibliothek verwenden die Standardsysteminstellungen für das Fortsetzen einer fehlgeschlagenen Workflowausführung.

Das Standardsystemverhalten wird durch das Ändern einer Konfigurationsdatei geändert. Weitere Informationen finden Sie unter [„Festlegen von benutzerdefinierten Eigenschaften zur Fortsetzung fehlgeschlagener Workflowausführungen“](#), auf Seite 142.

#### Voraussetzungen

Vergewissern Sie sich, dass Sie über die Berechtigungen zum Bearbeiten von Workflows verfügen.

#### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Workflows**.

- 3 Erweitern Sie die hierarchische Workflowliste und navigieren Sie zu dem Workflow, dessen Verhalten Sie einrichten möchten.
- 4 Klicken Sie mit der rechten Maustaste auf den Workflow und wählen Sie **Bearbeiten**.  
Der Workfloweditor wird geöffnet.

- 5 Wählen Sie auf der Registerkarte **Allgemein** eine Option aus dem Dropdown-Menü **Fortsetzungsverhalten bei Fehlschlag**.

Option	Beschreibung
<b>Systemstandard</b>	Verwendet das Standardverhalten.
<b>Aktiviert</b>	Wenn eine Workflowausführung fehlschlägt, wird ein Popup-Fenster mit einer Option zum Fortsetzen des Workflows angezeigt.
<b>Deaktiviert</b>	Eine fehlgeschlagene Workflowausführung kann nicht fortgesetzt werden.

- 6 Klicken Sie auf **Speichern und schließen**.

## Festlegen von benutzerdefinierten Eigenschaften zur Fortsetzung fehlgeschlagener Workflowausführungen

Orchestrator ist nicht standardmäßig zur Fortsetzung fehlgeschlagener Workflowausführungen eingerichtet. Sie können Orchestrator so einrichten, dass fehlgeschlagene Workflowausführungen fortgesetzt werden. Sie können auch eine benutzerdefinierte Zeitüberschreitungsdauer festlegen, nach der fehlgeschlagene Workflows nicht mehr fortgesetzt werden können.

### Vorgehensweise

- 1 Navigieren Sie im Orchestrator-Serversystem zu `/etc/vco/app-server/`.
- 2 Öffnen Sie die Konfigurationsdatei `vmo.properties` in einem Texteditor.
- 3 Richten Sie Orchestrator zur Fortsetzung fehlgeschlagener Workflowausführungen ein, indem Sie die folgende Zeile in der Datei `vmo.properties` bearbeiten:  
  
`com.vmware.vco.engine.execute.resume-from-failed=true`
- 4 Legen Sie eine benutzerdefinierte Zeitüberschreitungsdauer für die Wiederaufnahme fehlgeschlagener Workflows fest, indem Sie die folgende Zeile in der Datei `vmo.properties` bearbeiten:  
  
`com.vmware.vco.engine.execute.resume-from-failed.timeout-sec=<seconds>`  
  
Der festgelegte Wert überschreibt den Standardwert für die Zeitüberschreitung von 86400 Sekunden.
- 5 Speichern Sie die Datei `vmo.properties`.
- 6 Starten Sie den Orchestrator-Server neu.

## Fortsetzen einer fehlgeschlagenen Workflowausführung

Sie können eine Workflowausführung ab der letzten fehlgeschlagenen Aktivität fortsetzen, sofern das Fortsetzen einer fehlgeschlagenen Ausführung für den betreffenden Workflow aktiviert ist.

Wenn die Option zum Fortsetzen einer fehlgeschlagenen Workflowausführung aktiviert ist, können Sie mithilfe der Optionen in dem Popup-Fenster, das nach dem Fehlschlagen des Workflows angezeigt wird, die Parameter des Workflows ändern und versuchen, ihn fortzusetzen. Sie können auch die Parameter beibehalten und Änderungen an externen Komponenten mit Auswirkungen auf die Workflowausführung vornehmen. Wenn Sie keine Option auswählen, tritt eine Zeitüberschreitung auf und die Workflowausführung kann nicht fortgesetzt werden. Informationen zum Ändern des Zeitlimits finden Sie unter [„Festlegen von benutzerdefinierten Eigenschaften zur Fortsetzung fehlgeschlagener Workflowausführungen“](#), auf Seite 142.

**Vorgehensweise**

- 1 Wählen Sie im Dropdown-Menü im Popup-Fenster die Option **Fortsetzen** und klicken Sie auf **Weiter**.  
Wenn Sie **Abbrechen** auswählen, kann der Workflow nicht zu einem späteren Zeitpunkt fortgesetzt werden.
- 2 (Optional) Ändern Sie die Workflowparameter.
- 3 Klicken Sie auf **Senden**.

**Generieren einer Workflow-Dokumentation**

Ein ausgewählter Workflow oder Workflow-Ordner kann jederzeit als Dokumentation im PDF-Format exportiert werden.

Das exportierte Dokument enthält ausführliche Informationen zu dem/den ausgewählten Workflow/s im Ordner. Zu den Informationen zählen Name, Versionsverlauf des Workflows, Attribute, Parameter, Präsentation, Workflowschema und -aktionen. Außerdem gibt die Dokumentation den Quellcode der verwendeten Aktionen an.

**Vorgehensweise**

- 1 Wählen Sie im Dropdown-Menü des Orchestrator-Clients **Ausführen** oder **Design** aus.
- 2 Klicken Sie auf die Ansicht **Workflows**.
- 3 Navigieren Sie zum Workflow oder Workflow-Ordner, für den die Dokumentation generiert werden soll, und klicken Sie mit der rechten Maustaste darauf.
- 4 Wählen Sie **Dokumentation generieren** aus.
- 5 Navigieren Sie zu dem Ordner, in dem die PDF-Datei gespeichert werden soll, geben Sie einen Dateinamen an und klicken Sie auf **Speichern**.

Die PDF-Datei mit den Informationen zu dem/den ausgewählten Workflow/s im Ordner wird im System gespeichert.

**Verwenden des Workflowversionsverlaufs**

Mithilfe des Versionsverlaufs können Sie einen Workflow auf einen früher gespeicherten Zustand zurücksetzen. Sie können den Workflowzustand auf eine frühere oder spätere Workflowversion zurücksetzen. Weiterhin können Sie die Unterschiede zwischen dem aktuellen Workflowzustand und einer gespeicherten früheren Version des Workflows vergleichen.

Orchestrator erstellt ein neues Versionsverlaufselement für einen Workflow, wenn Sie die Versionsnummer erhöhen und den Workflow speichern. Nachfolgende Änderungen am Workflow bewirken keine Änderung am aktuellen Versionselement. Beispiel: Wenn Sie Workflowversion 1.0.0 erstellen und speichern, wird der Zustand des Workflows in der Datenbank gespeichert. Wenn Sie Änderungen am Workflow ausführen, können Sie den Workflow im Orchestrator-Client speichern, können die Änderungen jedoch nicht auf Workflowversion 1.0.0 anwenden. Um die Änderungen im Versionsverlauf zu speichern, müssen Sie eine Folgeversion des Workflows erstellen und speichern. Der Versionsverlauf wird zusammen mit dem Workflow in der Datenbank gespeichert.

Wenn Sie einen Workflow löschen, kennzeichnet Orchestrator das Element in der Datenbank als gelöscht, ohne jedoch den Versionsverlauf des Elements aus der Datenbank zu löschen. Dadurch können Sie gelöschte Workflows wiederherstellen. Weitere Informationen finden Sie unter „[Wiederherstellen von gelöschten Workflows](#)“, auf Seite 144.

**Voraussetzungen**

Öffnen Sie einen Workflow zur Bearbeitung im Workfloweditor.

**Vorgehensweise**

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Allgemein** und dann auf **Versionsverlauf anzeigen**.
- 2 Wählen Sie eine Workflowversion aus und klicken Sie auf **Vergleich mit aktuell**, um die Unterschiede zu sehen.  
  
Es wird ein Fenster eingeblendet, in dem die Unterschiede zwischen der aktuellen und der ausgewählten Workflowversion angezeigt werden.
- 3 Wählen Sie eine Workflowversion aus und klicken Sie auf **Zurücksetzen**, um den Zustand des Workflows wiederherzustellen.



**VORSICHT** Wenn Sie die aktuelle Workflowversion nicht gespeichert haben, wird sie aus dem Versionsverlauf gelöscht, und Sie können die aktuelle Version nicht wiederherstellen.

Der Workflowzustand wird auf den Zustand der ausgewählten Version zurückgesetzt.

## Wiederherstellen von gelöschten Workflows

Sie können Workflows wiederherstellen, die aus der Workflow-Bibliothek gelöscht wurden.

**Vorgehensweise**

- 1 Wählen Sie im Dropdown-Menü des Orchestrator-Clients **Ausführen** oder **Design** aus.
- 2 Klicken Sie auf die Ansicht **Workflows**.
- 3 Navigieren Sie zu dem Workflowordner, in dem Sie gelöschte Workflows wiederherstellen möchten.
- 4 Klicken Sie mit der rechten Maustaste auf den Ordner und wählen Sie **Gelöschte Workflows wiederherstellen** aus.
- 5 Wählen Sie die Workflows aus, die Sie wiederherstellen möchten, und klicken Sie auf **Wiederherstellen**.

Die Workflows werden im ausgewählten Ordner angezeigt.

## Entwickeln eines einfachen Beispielworkflows

Das Entwickeln eines einfachen Beispielworkflows zeigt die häufigsten Schritte im Entwicklungsprozess eines Workflows.

Der Beispielworkflow, den Sie erstellen möchten, startet eine vorhandene virtuelle Maschine im vCenter Server und sendet eine E-Mail an den Administrator, um zu bestätigen, dass die virtuelle Maschine gestartet wurde.

Der Beispielworkflow führt die folgenden Tasks aus:

- 1 Fordert den Benutzer auf, eine zu startende virtuelle Maschine auszuwählen.
- 2 Fordert den Benutzer auf, eine E-Mail-Adresse anzugeben, an die er Benachrichtigungen senden kann.
- 3 Prüft, ob die ausgewählte virtuelle Maschine bereits eingeschaltet ist.
- 4 Sendet eine Anforderung an die vCenter Server-Instanz, um die virtuelle Maschine zu starten.
- 5 Wartet darauf, dass vCenter Server die virtuelle Maschine startet und gibt eine Fehlermeldung zurück, wenn sie nicht gestartet wird oder der Vorgang zu lange dauert.
- 6 Wartet darauf, dass vCenter Server VMware Tools auf der virtuellen Maschine startet und gibt eine Fehlermeldung zurück, wenn es nicht gestartet wird oder der Vorgang zu lange dauert.



- 7 Überprüft, ob die virtuelle Maschine ausgeführt wird.
- 8 Sendet eine Benachrichtigung an die angegebene E-Mail-Adresse, dass die Maschine gestartet wurde oder ein Fehler aufgetreten ist.

Die ZIP-Datei von Orchestrator-Beispielen, die Sie von der Startseite der Orchestrator-Dokumentation herunterladen können, enthält eine vollständige Version des Workflows „VM starten und E-Mail senden“.

Der Vorgang für Entwickeln des Beispielworkflows besteht aus mehreren Aufgaben.

### Voraussetzungen

Bevor Sie versuchen, den einfachen Beispielworkflow zu entwickeln, lesen Sie „[Schlüsselkonzepte von Workflows](#)“, auf Seite 13.

### Vorgehensweise

- 1 [Erstellen des Beispiels für einen einfachen Workflow](#) auf Seite 146  
Sie müssen den Entwicklungsprozess des Workflows beginnen, indem Sie den Workflow im Orchestrator-Client erstellen.
- 2 [Erstellen Sie das Schema des Beispiels für einen einfachen Workflow](#) auf Seite 147  
Sie können das Schema eines Workflows im Workfloweditor erstellen. Das Workflowschema enthält die Elemente, die im Workflow ausgeführt werden, und bestimmt den logischen Fluss des Workflows.
- 3 [\(Optional\) Erstellen der Zonen des Beispiels für einen einfachen Workflow](#) auf Seite 149  
Sie können verschiedene Zonen im Workflow hervorheben, indem Sie Workflowanmerkungen in verschiedenen Farben hinzufügen. Durch das Erstellen verschiedener Workflowzonen erleichtern Sie das Lesen und Verstehen eines komplexen Workflowschemas.
- 4 [Definieren der Parameter für das einfache Workflowbeispiel](#) auf Seite 151  
In dieser Phase der Workflowentwicklung definieren Sie die Eingabeparameter, die der Workflow für die Ausführung benötigt. Für den Beispielworkflow benötigen Sie einen Eingabeparameter für die einzuschaltende virtuelle Maschine und einen Parameter für die E-Mail-Adresse der über das Ergebnis des Vorgangs zu informierenden Person. Wenn Benutzer den Workflow ausführen, müssen sie die einzuschaltende virtuelle Maschine und eine E-Mail-Adresse angeben.
- 5 [Definieren der Entscheidungsbindungen im einfachen Workflowbeispiel](#) auf Seite 152  
Sie binden die Elemente eines Workflows im Workfloweditor auf der Registerkarte **Schema** zusammen. Entscheidungsbindungen legen fest, wie Entscheidungselemente die Eingabeparameter von der Entscheidungsanweisung vergleichen und Ausgabeparameter generieren, falls die Eingabeparameter mit der Entscheidungsanweisung übereinstimmen.
- 6 [Binden der Aktionselemente für das einfache Workflowbeispiel](#) auf Seite 152  
Sie können die Elemente eines Workflows im Workfloweditor aneinander binden. Bindungen definieren, wie die Aktionselemente Eingabeparameter verarbeiten und Ausgabeparameter generieren.
- 7 [Binden der skriptfähigen Aufgabenelemente für das einfache Workflowbeispiel](#) auf Seite 155  
Die Elemente eines Workflows werden im Workfloweditor auf der Registerkarte **Schema** gebunden. Bindungen definieren, wie die skriptfähigen Aufgabenelemente Eingabeparameter verarbeiten und Ausgabeparameter generieren. Sie können auch die skriptfähigen Aufgabenelemente an ihre JavaScript-Funktionen binden.
- 8 [Definieren der Ausnahmebindungen im einfachen Workflowbeispiel](#) auf Seite 163  
Sie definieren Ausnahmebindungen im Workfloweditor auf der Registerkarte **Schema**. Ausnahmebindungen definieren, wie Elemente Fehler verarbeiten.

- 9 [Setzen der Schreibschutzeigenschaften für Attribute des Beispiels für einen einfachen Workflow](#) auf Seite 164  
Sie können definieren, ob Parameter und Attribute schreibgeschützte Konstanten oder beschreibbare Variablen sind. Sie können auch Einschränkungen für die Werte festlegen, die Benutzer für die Eingabeparameter eingeben können.
- 10 [Setzen der Parametereigenschaften des Beispiels für einen einfachen Workflow](#) auf Seite 165  
Sie können die Parametereigenschaften im Workfloweditor setzen. Das Setzen von Parametereigenschaften beeinflusst das Verhalten des Parameters und legt Integritätsregeln für die möglichen Werte dieses Parameters fest.
- 11 [Setzen des Layouts für das Dialogfeld der Eingabeparameter des Beispiels für einen komplexen Workflow](#) auf Seite 166  
Sie erstellen das Layout oder die Präsentation des Dialogfelds für die Eingabeparameter im Workfloweditor. Das Dialogfeld für die Eingabeparameter erscheint, wenn Benutzer einen Workflow ausführen, für den Eingabeparameter erforderlich sind.
- 12 [Validieren und Ausführen des Beispiels für einen einfachen Workflow](#) auf Seite 167  
Nachdem Sie einen Workflow erstellt haben, können Sie ihn validieren, um eventuelle Fehler zu erkennen. Wenn der Workflow keine Fehler enthält, können Sie ihn ausführen.

## Erstellen des Beispiels für einen einfachen Workflow

Sie müssen den Entwicklungsprozess des Workflows beginnen, indem Sie den Workflow im Orchestrator-Client erstellen.

### Voraussetzungen

Überprüfen Sie, ob die folgenden Komponenten im System installiert und konfiguriert sind.

- vCenter Server, der einige virtuelle Maschinen steuert, von denen mindestens eine abgeschaltet ist
- Zugriff auf einen SMTP-Server
- Gültige E-Mail-Adresse

Informationen zur Installation und Konfiguration von vCenter Server finden Sie in der Dokumentation *Installation und Einrichtung von vSphere*. Informationen zur Konfiguration von Orchestrator für die Verwendung eines SMTP-Servers finden Sie unter *Installieren und Konfigurieren von VMware vRealize Orchestrator*.

Zum Schreiben eines Workflows müssen Sie ein Orchestrator-Benutzerkonto mit zumindest folgenden Rechten auf dem Server oder für den Workflowordner haben, auf bzw. in dem Sie arbeiten: **Anzeigen, Ausführen, Überprüfen, Bearbeiten** und vorzugsweise **Administrator**.

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Workflows**.
- 3 Klicken Sie mit der rechten Maustaste in die Workflowliste und wählen Sie **Ordner hinzufügen**.
- 4 Benennen Sie den neuen Ordner **Workflowbeispiele** und klicken Sie auf **OK**.
- 5 Klicken Sie mit der rechten Maustaste auf den Ordner **Workflowbeispiele** und wählen Sie **Neuer Workflow**.
- 6 Benennen Sie den neuen Workflow **VM starten und E-Mail senden** und klicken Sie auf **OK**.

Der Workfloweditor wird geöffnet.

- 7 Klicken Sie auf der Registerkarte **Allgemein** auf die Ziffern der Versionsnummer, um die Versionsnummer zu erhöhen.

Da dies die erste Erstellung des Workflows ist, setzen Sie die Version auf **0.0.1**.

- 8 Klicken Sie auf den Wert **Serverneustart-Verhalten** auf der Registerkarte **Allgemein**, um festzulegen, ob der Workflow nach einem Serverneustart fortgesetzt wird.
- 9 Geben Sie eine Beschreibung der Aufgabe des Workflows in das Textfeld **Beschreibung** auf der Registerkarte **Allgemein** ein.

Sie können beispielsweise die folgende Beschreibung hinzufügen.

**Dieser Workflow startet eine virtuelle Maschine und sendet eine Bestätigungs-E-Mail an den Orchestrator-Administrator.**

- 10 Klicken Sie unten auf der Registerkarte **Allgemein** auf **Speichern**.

Sie haben einen Workflow mit dem Namen „VM starten und E-Mail senden“ erstellt, aber seine Funktionen noch nicht definiert.

### Weiter

Erstellen Sie das Schema des Workflows.

## Erstellen Sie das Schema des Beispiels für einen einfachen Workflow

Sie können das Schema eines Workflows im Workfloweditor erstellen. Das Workflowschema enthält die Elemente, die im Workflow ausgeführt werden, und bestimmt den logischen Fluss des Workflows.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen einfachen Workflow“, auf Seite 146.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Schema**.
- 2 Ziehen Sie aus dem Menü **Generisch** das Entscheidungselement auf den Pfeil, der das Startelement und das End-Element im Schema miteinander verknüpft.
- 3 Doppelklicken Sie auf das Entscheidungselement und ändern Sie seinen Namen auf **VM eingeschaltet?**.  
Dieses Entscheidungselement entspricht einer booleschen Funktion, die prüft, ob die virtuelle Maschine bereits eingeschaltet ist.
- 4 Ziehen Sie aus dem Menü **Generisch** ein Aktionselement auf den roten Pfeil, der das Entscheidungselement und ein End-Element miteinander verknüpft.  
Das Dialogfeld für die Auswahl einer Aktion wird eingeblendet.
- 5 Geben Sie den Befehl **start** in das Textfeld **Filter** ein, wählen Sie die Aktion **startVM** aus der gefilterten Aktionsliste und klicken Sie auf **Auswählen**.

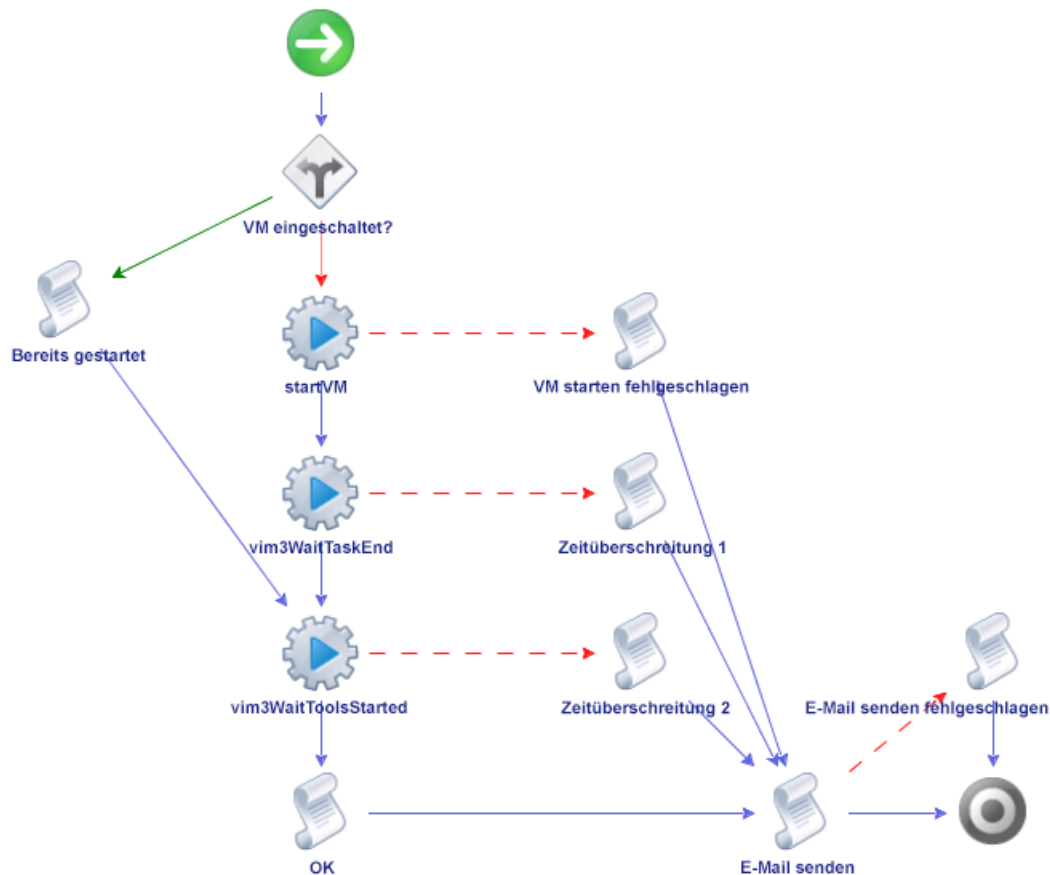
- 6 Ziehen Sie die folgenden Aktionselemente nacheinander auf den blauen Pfeil, der das Aktionselement `startVM` mit einem End-Element verknüpft.

<b><code>vim3WaitTaskEnd</code></b>	Hält das Ausführen des Workflows an und sendet in regelmäßigen Intervallen einen Ping an eine ausgehende vCenter Server-Aufgabe, bis diese Aufgabe abgeschlossen ist. Die Aktion <code>startVM</code> startet eine virtuelle Maschine und die Aktion <code>vim3WaitTaskEnd</code> hält den Workflow an, während die virtuelle Maschine hochfährt. Nachdem die virtuelle Maschine gestartet wurde, bewirkt <code>vim3WaitTaskEnd</code> die Wiederaufnahme des Workflows.
<b><code>vim3WaitToolsStarted</code></b>	Hält das Ausführen des Workflows an und wartet, bis VMware Tools auf der virtuellen Maschine gestartet ist.

- 7 Ziehen Sie aus dem Menü **Generisch** ein Element für eine skriptfähige Aufgabe auf den blauen Pfeil, der das Aktionselement `vim3WaitToolsStarted` mit einem Element `End` verknüpft.
- 8 Doppelklicken Sie auf das Element für eine skriptfähige Aufgabe und benennen Sie es in **OK** um.
- 9 Ziehen Sie ein weiteres Element für eine skriptfähige Aufgabe auf den grünen Pfeil, der das Entscheidungselement `VM powered on?` mit einem End-Element verknüpft, und benennen Sie das Element für eine skriptfähige Aufgabe als **Bereits gestartet**.
- 10 Ändern Sie die Verknüpfung des skriptfähigen Aufgabenelements `Already started`.
  - a Ziehen Sie das skriptfähige Aufgabenelement `Already started` an eine Stelle links vom Aktionselement `startVM`.
  - b Löschen Sie den blauen Pfeil, der das skriptfähige Aufgabenelement `Already started` mit einem End-Element verknüpft.
  - c Verknüpfen Sie das Element `Already started` für eine skriptfähige Aufgabe mit dem Aktionselement `vim3WaitToolsStarted` über einen blauen Pfeil.
- 11 Ziehen Sie aus dem Menü **Generisch** die folgenden Elemente für eine skriptfähige Aufgabe in das Schema.
  - Ziehen Sie ein Element für eine skriptfähige Aufgabe auf das Aktionselement `startVM` und benennen Sie das skriptfähige Aufgabenelement **VM-Start fehlgeschlagen**.
  - Ziehen Sie ein skriptfähiges Aufgabenelement auf das Aktionselement `vim3WaitTaskEnd` und benennen Sie das skriptfähige Aufgabenelement **Zeitüberschreitung 1**.
  - Ziehen Sie ein Element für eine skriptfähige Aufgabe auf das Aktionselement `vim3WaitToolsStarted` und benennen Sie das skriptfähige Aufgabenelement **Zeitüberschreitung 2**.
  - Ziehen Sie ein Element für eine skriptfähige Aufgabe auf den blauen Pfeil, der das skriptfähige Aufgabenelement `OK` mit einem End-Element verknüpft, benennen Sie das neue Element für eine skriptfähige Aufgabe **E-Mail senden** und ziehen Sie es an eine Stelle rechts vom `OK`-Element für eine skriptfähige Aufgabe.
  - Verknüpfen Sie mit blauen Pfeilen die Elemente für eine skriptfähige Aufgabe `Start VM Failed`, `Timeout 1` und `Timeout 2` mit dem `Send Email`-Element für eine skriptfähige Aufgabe.
  - Ziehen Sie ein skriptfähiges Aufgabenelement auf das skriptfähige Aufgabenelement `Send Email`, benennen Sie das neue skriptfähige Aufgabenelement **E-Mail-Versand fehlgeschlagen**, ziehen Sie es an eine Stelle rechts vom skriptfähigen Aufgabenelement `Timeout 2` und verknüpfen Sie es mit einem blauen Pfeil mit dem End-Element.
- 12 Ziehen Sie das End-Element an eine Stelle rechts vom skriptfähigen Aufgabenelement `Send Email`.
- 13 Klicken Sie unten auf der Registerkarte **Schema** auf **Speichern**.

Die folgende Abbildung zeigt das Layout der Schemaelemente des Workflows „VM starten und E-Mail senden“.

**Abbildung 1-10.** Verknüpfen der Elemente des Beispielworkflows „VM starten und E-Mail senden“



### Weiter

Sie können verschiedene Zonen des Workflows hervorheben.

## (Optional) Erstellen der Zonen des Beispiels für einen einfachen Workflow

Sie können verschiedene Zonen im Workflow hervorheben, indem Sie Workflowanmerkungen in verschiedenen Farben hinzufügen. Durch das Erstellen verschiedener Workflowzonen erleichtern Sie das Lesen und Verstehen eines komplexen Workflowschemas.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen einfachen Workflow“, auf Seite 146.
- „Erstellen Sie das Schema des Beispiels für einen einfachen Workflow“, auf Seite 147.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

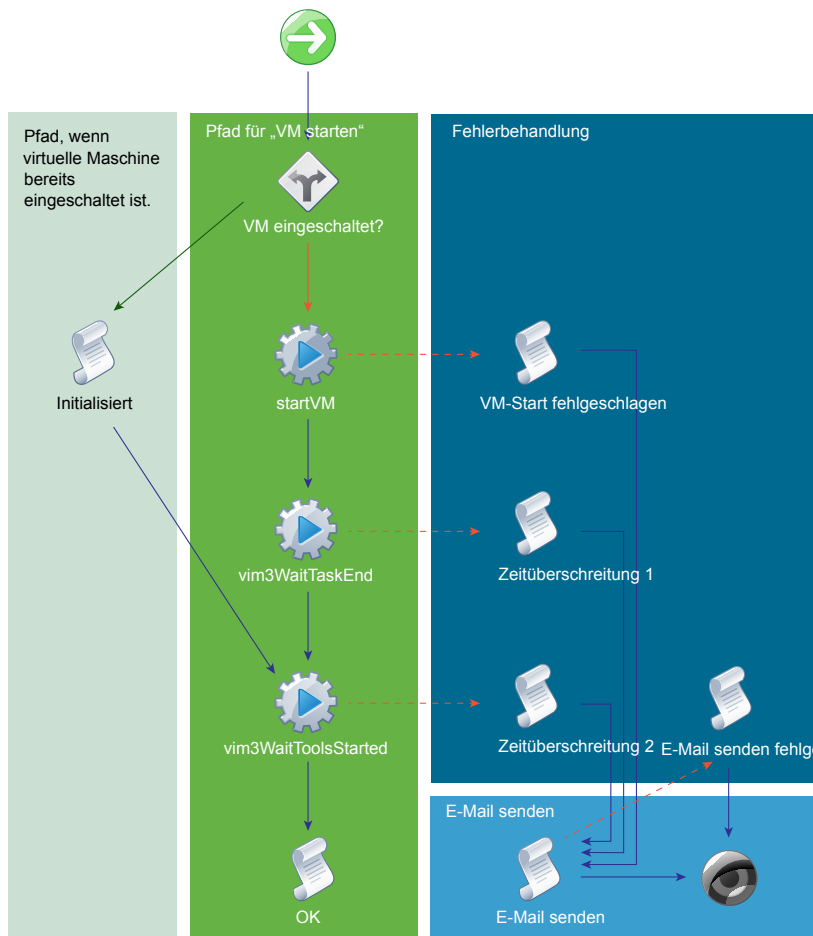
### Vorgehensweise

- 1 Ziehen Sie ein Workflowanmerkungs-element aus dem Menü **Generisch** in den Workfloweditor.
- 2 Positionieren Sie die Workflowanmerkung über dem skriptfähigen Aufgabenelement **Already started**.
- 3 Ziehen Sie die Ränder der Workflowanmerkung, um ihre Größe anzupassen, sodass sie das skriptfähige Aufgabenelement **Already started** umschließt.

- 4 Doppelklicken Sie auf den Text und fügen Sie eine Beschreibung hinzu.  
Beispiel: **Pfad, wenn virtuelle Maschine bereits eingeschaltet ist.**
- 5 Drücken Sie Strg+E, um die Hintergrundfarbe auszuwählen.
- 6 Wiederholen Sie die vorstehenden Schritte, um andere Zonen im Workflow hervorzuheben.
  - Setzen Sie eine Anmerkung über die vertikale Sequenz von Elementen vom Entscheidungselement VM powered on? bis zum OK-Element. Fügen Sie die Beschreibung **VM-Pfad starten** hinzu.
  - Setzen Sie eine Anmerkung über das ElementstartVM failed, beide Timeout-Elemente für eine skriptfähige Aufgabe und das skriptfähige Aufgabenelement Send Email Failed. Fügen Sie die Beschreibung **Fehlerbehandlung** hinzu.
  - Setzen Sie eine Anmerkung über das skriptfähige Aufgabenelement Send Email. Fügen Sie die Beschreibung **E-Mail senden** hinzu.

Die folgende Abbildung zeigt wie die Zonen des Beispielworkflows aussehen sollten.

**Abbildung 1-11.** Zonen des Beispielworkflows „VM starten und E-Mail senden“.



### Weiter

Sie müssen die Attribute sowie die Eingabe- und Ausgabeparameter des Workflows definieren.

## Definieren der Parameter für das einfache Workflowbeispiel

In dieser Phase der Workflowentwicklung definieren Sie die Eingabeparameter, die der Workflow für die Ausführung benötigt. Für den Beispielworkflow benötigen Sie einen Eingabeparameter für die einzuschaltende virtuelle Maschine und einen Parameter für die E-Mail-Adresse der über das Ergebnis des Vorgangs zu informierenden Person. Wenn Benutzer den Workflow ausführen, müssen sie die einzuschaltende virtuelle Maschine und eine E-Mail-Adresse angeben.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen einfachen Workflow“, auf Seite 146.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Eingaben**.
- 2 Klicken Sie mit der rechten Maustaste in die Registerkarte **Eingaben** und wählen Sie **Parameter hinzufügen**.  
Ein Parameter mit dem Namen `arg_in_0` wird auf der Registerkarte **Eingaben** angezeigt.
- 3 Klicken Sie auf **arg\_in\_0**.
- 4 Geben Sie den Namen **vm** in das Dialogfeld „Attributname auswählen“ ein und klicken Sie auf **OK**.
- 5 Klicken Sie auf das Textfeld **Typ** und geben Sie in das Suchfeld für „Parametertyp“ **vc:virtualm** ein.
- 6 Wählen Sie **VC:VirtualMachine** aus der vorgeschlagenen Liste der Parametertypen und klicken Sie auf **Akzeptieren**.
- 7 Fügen Sie im Textfeld **Beschreibung** eine Beschreibung des Parameters hinzu.  
Geben Sie beispielsweise **Die einzuschaltende virtuelle Maschine** ein.
- 8 Wiederholen Sie [Schritt 2](#) über [Schritt 7](#), um einen zweiten Eingabeparameter mit den folgenden Werten zu erstellen.
  - Name: `toAddress`
  - Typ: Zeichenfolge
  - Beschreibung: **Die E-Mail-Adresse zum Senden des Ergebnisses dieses Workflows an**
- 9 Klicken Sie unten in der Registerkarte **Eingaben** auf **Speichern**.

Sie haben die Eingabeparameter des Workflows definiert.

### Weiter

Definieren Sie die Bindungen zwischen den Elementparametern.

## Definieren der Entscheidungsbindungen im einfachen Workflowbeispiel


Sie binden die Elemente eines Workflows im Workfloweditor auf der Registerkarte **Schema** zusammen. Entscheidungsbindungen legen fest, wie Entscheidungselemente die Eingabeparameter von der Entscheidungsanweisung vergleichen und Ausgabeparameter generieren, falls die Eingabeparameter mit der Entscheidungsanweisung übereinstimmen.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen einfachen Workflow“, auf Seite 146.
- „Erstellen Sie das Schema des Beispiels für einen einfachen Workflow“, auf Seite 147.
- „Definieren der Parameter für das einfache Workflowbeispiel“, auf Seite 151.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

### Vorgehensweise

- 1 Klicken Sie auf der Registerkarte **Schema** auf das Symbol **Bearbeiten** () des **VM Powered On?** Entscheidungselements.
- 2 Klicken Sie auf der Registerkarte **Entscheidung** auf die Schaltfläche **Nicht festgelegt (NULL)** und wählen Sie **vm** als Eingabeparameter des Entscheidungsparameters aus der Liste vorgeschlagener Parameter.
- 3 Wählen Sie die Anweisung **Betriebszustand ist gleich** aus der Liste von Entscheidungsanweisungen im Dropdown-Menü.  
  
Die Schaltfläche **Nicht festgelegt** wird im Textfeld „Wert“ angezeigt, wo Sie eine begrenzte Auswahl an möglichen Werten haben.
- 4 Wählen Sie **poweredOn** aus.
- 5 Klicken Sie unten in der Workfloweditor-Registerkarte **Schema** auf **Speichern**.

Sie haben die Anweisung „true“ oder „false“ definiert, anhand derer das Entscheidungselement den Wert der Eingabeparameter, die es empfängt, vergleicht.

### Weiter

Sie müssen die Bindungen für die anderen Elemente im Workflow definieren.

## Binden der Aktionselemente für das einfache Workflowbeispiel

Sie können die Elemente eines Workflows im Workfloweditor aneinander binden. Bindungen definieren, wie die Aktionselemente Eingabeparameter verarbeiten und Ausgabeparameter generieren.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen einfachen Workflow“, auf Seite 146.
- „Erstellen Sie das Schema des Beispiels für einen einfachen Workflow“, auf Seite 147.
- „Definieren der Parameter für das einfache Workflowbeispiel“, auf Seite 151.
- „Definieren der Entscheidungsbindungen im einfachen Workflowbeispiel“, auf Seite 152.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.



## Vorgehensweise

- 1 Klicken Sie auf der Registerkarte **Schema** auf das Symbol **Bearbeiten** (✎) des Aktionselements startVM.
- 2 Legen Sie die folgenden allgemeinen Informationen auf der Registerkarte **Info** fest.

Option	Aktion
<b>Interaktion</b>	Wählen Sie <b>Keine externe Interaktion</b> .
<b>Business-Status</b>	Aktivieren Sie das Kontrollkästchen und fügen Sie den Text „ <b>VM starten</b> “ <b>senden</b> .
<b>Beschreibung</b>	Behalten Sie den Text „VM starten/fortsetzen“ bei. Geben Sie die Startaufgabe zurück.

- 3 Klicken Sie auf die Registerkarte **EIN**.  
Auf der Registerkarte **EIN** werden die beiden möglichen Eingabeparameter, die für die Aktion startVM zur Verfügung stehen, angezeigt: vm und host .  
Orchestrator bindet automatisch den Parameter vm an vm[in-parameter], weil die Aktion startVM nur eine VC:VirtualMachine als Eingabeparameter verwenden kann. Orchestrator erkennt den Parameter vm, den Sie beim Festlegen der Eingabeparameter für den Workflow definiert haben, und bindet ihn automatisch an die Aktion.
- 4 Legen Sie host auf **NULL** fest.  
Dies ist ein optionaler Parameter. Daher können Sie ihn auf Null festlegen. Wenn Sie ihn auf **Nicht festgelegt** lassen, kann der Workflow nicht geprüft werden.
- 5 Klicken Sie auf die Registerkarte **AUS**.  
Daraufhin wird actionResult angezeigt, der Standardausgabeparameter, der bei allen Aktionen generiert wird.
- 6 Klicken Sie für den Parameter actionResult auf **Nicht festgelegt**.
- 7 Klicken Sie auf **Parameter/Attribut in Workflow erstellen**.  
Im Dialogfeld „Parameterinformationen“ werden die Werte angezeigt, die Sie für diesen Ausgabeparameter festlegen können. Der Ausgabeparametertyp für die Aktion startVM ist ein VC:Task-Objekt.
- 8 Nennen Sie den Parameter **powerOnTask** und geben Sie eine Beschreibung an.  
Beispiel: **Enthält das Ergebnis des Einschaltens einer VM.**
- 9 Klicken Sie auf **Workflow-ATTRIBUT mit demselben Namen erstellen** und auf **OK**, um das Dialogfeld „Parameterinformationen“ zu beenden.
- 10 Wiederholen Sie die vorhergehenden Schritte zum Binden der Eingabe- und Ausgabeparameter an die Aktionselemente vim3WaitTaskEnd und vim3WaitToolsStarted.  
„[Elementbindungen im Beispiel für einen einfachen Workflow](#)“, auf Seite 154 listet die Bindungen für die Aktionselemente vim3WaitTaskEnd und vim3WaitToolsStarted auf.
- 11 Klicken Sie unten auf der Registerkarte **Schema** des Workfloweditors auf **Speichern**.

Die Eingabe- und Ausgabeparameter der Aktionselemente werden an die entsprechenden Parametertypen und -werte gebunden.

## Weiter

Binden Sie die skriptfähigen Aufgabenelemente und definieren Sie ihre Funktionen.

## Elementbindungen im Beispiel für einen einfachen Workflow

Bindungen definieren, wie die Aktionselemente des einfachen Workflowbeispiels Eingabe- und Ausgabeparameter verarbeiten.

Beim Definieren von Bindungen präsentiert Orchestrator Parameter, die Sie bereits im Workflow definiert haben, als Bewerber für Bindungen. Wenn Sie den erforderlichen Parameter im Workflow noch nicht definiert haben, steht nur der Parameter NULL zur Auswahl. Klicken Sie auf **Parameter/Attribut in Workflow erstellen**, um einen neuen Parameter zu erstellen.

### vim3WaitTaskEnd-Aktion

Das Aktionselement vim3WaitTaskEnd deklariert Konstanten, um den Fortschritt einer Aufgabe zu verfolgen und eine Abfragerate zu bestimmen. In der folgenden Tabelle sind die Bindungen für Eingabe- und Ausgabeparameter enthalten, die die Aktion vim3WaitTaskEnd benötigt.

**Tabelle 1-54.** Binden von Werten der vim3WaitTaskEnd-Aktion

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
task	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: powerOnTask</li> <li>■ Quellparameter: task[attribute]</li> <li>■ Typ: VC:Task</li> <li>■ Beschreibung: <b>Enthält das Ergebnis des Einschaltens einer VM.</b></li> </ul>
progress	EIN	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: progress</li> <li>■ Quellparameter: progress[attribute]</li> <li>■ Typ: Boolesch</li> <li>■ Wert: No (false)</li> <li>■ Beschreibung: <b>Protokollieren des Fortschritts, während auf den Abschluss der vCenter Server-Aufgabe gewartet wird.</b></li> </ul>
pollRate	EIN	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: pollRate</li> <li>■ Quellparameter: pollRate[attribute]</li> <li>■ Typ: Zahl</li> <li>■ Wert: 2</li> <li>■ Beschreibung: <b>Abfragerate in Sekunden, mit der vim3WaitTaskEnd den Fortschritt der vCenter Server-Aufgabe prüft.</b></li> </ul>
actionResult	AUS	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: actionResult[attribute]</li> <li>■ Quellparameter: returnedManagedObject[attribute]</li> <li>■ Typ: Beliebig</li> <li>■ Beschreibung: <b>Das zurückgegebene verwaltete Objekt von der waitTaskEnd-Aktion.</b></li> </ul>

### vim3WaitToolsStarted-Aktion

Das Aktionselement `vim3WaitToolsStarted` wartet, bis VMware Tools auf einer virtuellen Maschine installiert ist, und definiert eine Abfragerate sowie eine Zeitüberschreitungsperiode. In der folgenden Tabelle sind die Bindungen für Eingabeparameter enthalten, die die Aktion `vim3WaitToolsStarted` benötigt.

Das Aktionselement `vim3WaitToolsStarted` hat keine Ausgabe und benötigt daher keine Ausgabebindung.

**Tabelle 1-55.** Bindungswerte der Aktion `vim3WaitToolsStarted`

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
<code>vm</code>	EIN	Automatische Bindung	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: <code>vm</code></li> <li>■ Quellparameter: <code>vm[in-parameter]</code></li> <li>■ Typ: <code>VC:VirtualMachine</code></li> <li>■ Wert: nicht bearbeitet war, variable ist kein Workflowattribut.</li> <li>■ Beschreibung: <b>Die zu startende virtuelle Maschine.</b></li> </ul>
<code>pollingRate</code>	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: <code>pollRate</code></li> <li>■ Quellparameter: <code>pollRate[attribute]</code></li> <li>■ Typ: Zahl</li> <li>■ Beschreibung: <b>Abfragerate in Sekunden, mit der vim3WaitTaskEnd den Fortschritt der vCenter Server-Aufgabe prüft.</b></li> </ul>
<code>timeout</code>	EIN	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: <code>timeout</code></li> <li>■ Quellparameter: <code>timeout[attribute]</code></li> <li>■ Typ: Zahl</li> <li>■ Wert: <b>10</b></li> <li>■ Beschreibung: <b>Der Grenzwert für die Zeitüberschreitung, die vim3WaitToolsStarted wartet, bevor eine Ausnahme gemeldet wird.</b></li> </ul>

## Binden der skriptfähigen Aufgabenelemente für das einfache Workflowbeispiel

Die Elemente eines Workflows werden im Workfloweditor auf der Registerkarte **Schema** gebunden. Bindungen definieren, wie die skriptfähigen Aufgabenelemente Eingabeparameter verarbeiten und Ausgabeparameter generieren. Sie können auch die skriptfähigen Aufgabenelemente an ihre JavaScript-Funktionen binden.


### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen einfachen Workflow“, auf Seite 146.
- „Erstellen Sie das Schema des Beispiels für einen einfachen Workflow“, auf Seite 147.
- „Definieren der Parameter für das einfache Workflowbeispiel“, auf Seite 151.
- „Definieren der Entscheidungsbindungen im einfachen Workflowbeispiel“, auf Seite 152.

- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

### Vorgehensweise

- 1 Klicken Sie auf der Registerkarte **Schema** auf das Symbol **Bearbeiten** () des skriptfähigen Aufgabenelements **Already Started**.
- 2 Legen Sie auf der Registerkarte **Info** die folgenden allgemeinen Informationen fest.

Option	Aktion
<b>Interaktion</b>	Wählen Sie <b>Keine externe Interaktion</b> .
<b>Business-Status</b>	Aktivieren Sie das Kontrollkästchen und fügen Sie den Text <b>VM bereits eingeschaltet</b> ein.
<b>Beschreibung</b>	Der Text „VM bereits eingeschaltet“ wird beibehalten, „startVM“ und „waitTaskEnd“ werden umgangen und es wird überprüft, ob die VM-Tools ausgeführt werden.

- 3 Klicken Sie auf die Registerkarte **EIN**.  
Da es sich hierbei um ein benutzerdefiniertes, skriptfähiges Aufgabenelement handelt, sind keine Eigenschaften für Sie vordefiniert.

- 4 Klicken Sie auf das Symbol **An Workflow-Parameter/-Attribut binden** ()

- 5 Wählen Sie aus der vorgeschlagenen Liste von Parametern **vm** aus.

- 6 Lassen Sie die Registerkarten **AUS** und **Ausnahme** unausgefüllt.

Dieses Element generiert keinen Ausgabeparameter bzw. keine Ausnahme.

- 7 Klicken Sie auf die Registerkarte **Skripterstellung**.

- 8 Fügen Sie die folgende JavaScript-Funktion hinzu.

```
//Writes the following event in the Orchestrator database
Server.log("VM '"+ vm.name +" already started");
```

- 9 Wiederholen Sie die vorhergehenden Schritte zum Binden der übrigen Eingabeparameter an die anderen skriptfähigen Aufgabenelemente.

„[Elementbindungen für skriptfähige Aufgabe im Beispiel für einen einfachen Workflow](#)“, auf Seite 157 listet die Bindungen für die skriptfähigen Aufgabenelemente **Start VM failed** (Timeout oder Error) **Send Email Failed** und **OK** auf.

- 10 Klicken Sie unten auf der Registerkarte **Schema** des Workfloweditors auf **Speichern**.

Sie haben die skriptfähigen Aufgabenelemente an ihre Eingabe- und Ausgabeparameter gebunden und das Skript zur Definition ihrer Funktion bereitgestellt.

### Weiter

Sie müssen die Ausnahmebehandlung definieren.

## Elementbindungen für skriptfähige Aufgabe im Beispiel für einen einfachen Workflow

Bindungen definieren, wie die Elemente für eine skriptfähige Aufgabe des einfachen Workflowbeispiels Eingabe- und Ausgabeparameter verarbeiten. Sie können auch die skriptfähigen Aufgabenelemente an ihre JavaScript-Funktionen binden.

Beim Definieren von Bindungen präsentiert Orchestrator Parameter, die Sie bereits im Workflow definiert haben, als Bewerber für Bindungen. Wenn Sie den erforderlichen Parameter im Workflow noch nicht definiert haben, steht nur der Parameter NULL zur Auswahl. Klicken Sie auf **Parameter/Attribut in Workflow erstellen**, um einen neuen Parameter zu erstellen.

### Skriptfähige Aufgabe „VM-Start fehlgeschlagen“

Die skriptfähige Aufgabe „VM-Start fehlgeschlagen“ behandelt alle Ausnahmen, die die Aktion startVM generiert, indem der Inhalt einer E-Mail-Benachrichtigung über den Fehlschlag beim Start der virtuellen Maschine festgelegt und das Ereignis in das Orchestrator-Protokoll geschrieben wird.

In der folgenden Tabelle sind die Bindungen für Eingabe- und Ausgabeparameter enthalten, die das skriptfähige Aufgabenelement „VM-Start fehlgeschlagen“ benötigt.

**Tabelle 1-56.** Bindungen des skriptfähigen Aufgabenelements „VM-Start fehlgeschlagen“

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
vm	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vm</li> <li>■ Quellparameter: vm[in-parameter]</li> <li>■ Typ: VC:VirtualMachine</li> <li>■ Beschreibung: <b>Die einzuschaltende virtuelle Maschine.</b></li> </ul>
errorCode	EIN	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: errorCode</li> <li>■ Quellparameter: errorCode[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Abfangen von Ausnahmen beim Einschalten einer VM.</b></li> </ul>
body	AUS	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: body</li> <li>■ Quellparameter: body[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Der Haupttext der E-Mail</b></li> </ul>

Das skriptfähige Aufgabenelement „VM-Start fehlgeschlagen“ führt die folgende Skriptfunktion durch.

```
body = "Unable to execute powerOnVM_Task() on VM '"+vm.name+"', exception found: "+errorCode;
//Writes the following event in the Orchestrator database
Server.error("Unable to execute powerOnVM_Task() on VM '"+vm.name+"', exception found: "+errorCode);
```

### Skriptfähiges Aufgabenelement Zeitüberschreitung 1

Die skriptfähige Aufgabe „Zeitüberschreitung 1“ behandelt alle Ausnahmen, die die Aktion vim3WaitTaskEnd generiert, in dem der Inhalt einer E-Mail-Benachrichtigung über den Fehlschlag der Aufgabe festgelegt und das Ereignis in das Orchestrator-Protokoll geschrieben wird.

In der folgenden Tabelle sind die Bindungen für Eingabe- und Ausgabeparameter enthalten, die das skriptfähige Aufgabenelement „Zeitüberschreitung 1“ benötigt.

**Tabelle 1-57.** Bindungen des skriptfähigen Aufgabenelements für Zeitüberschreitung 1

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
vm	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vm</li> <li>■ Quellparameter: vm[in-parameter]</li> <li>■ Typ: VC:VirtualMachine</li> <li>■ Beschreibung: <b>Die zu startende virtuelle Maschine.</b></li> </ul>
errorCode	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: errorCode</li> <li>■ Quellparameter: errorCode[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Abfangen von Ausnahmen beim Einschalten einer VM.</b></li> </ul>
body	AUS	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: body</li> <li>■ Quellparameter: body[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Der Haupttext der E-Mail</b></li> </ul>

Das skriptfähige Aufgabenelement für Zeitüberschreitung 1 erfordert die folgende Skriptfunktion.

```
body = "Error while waiting for poweredOnVM_Task() to complete on VM '"+vm.name+"', exception found: "+errorCode;
//Writes the following event in the Orchestrator database
Server.error("Error while waiting for poweredOnVM_Task() to complete on VM '"+vm.name+"', exception found: "+errorCode);
```

### Skriptfähiges Aufgabenelement Zeitüberschreitung 2

Die skriptfähige Aufgabe „Zeitüberschreitung 2“ behandelt alle Ausnahmen, die die Aktion vim3WaitToolsStarted generiert, indem der Inhalt einer E-Mail-Benachrichtigung über den Fehlschlag der Aufgabe festgelegt und das Ereignis in das Orchestrator-Protokoll geschrieben wird.

In der folgenden Tabelle sind die Bindungen für Eingabe- und Ausgabeparameter enthalten, die das skriptfähige Aufgabenelement „Zeitüberschreitung 2“ benötigt.

**Tabelle 1-58.** Bindungen des skriptfähigen Aufgabenelements für Zeitüberschreitung 2

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
vm	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vm</li> <li>■ Quellparameter: vm[in-parameter]</li> <li>■ Typ: VC:VirtualMachine</li> <li>■ Beschreibung: <b>Die einzuschaltende virtuelle Maschine.</b></li> </ul>
errorCode	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: errorCode</li> <li>■ Quellparameter: errorCode[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Abfangen von Ausnahmen beim Einschalten einer VM.</b></li> </ul>
body	AUS	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: body</li> <li>■ Quellparameter: body[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Der Haupttext der E-Mail</b></li> </ul>

Das skriptfähige Aufgabenelement für Zeitüberschreitung 2 erfordert die folgende Skriptfunktion.

```
body = "Error while waiting for VMware tools to be up on VM '"+vm.name+"', exception found: "+errorCode;
//Writes the following event in the Orchestrator database
Server.error("Error while waiting for VMware tools to be up on VM '"+vm.name+"', exception found: "+errorCode);
```

### Skriptfähiges Aufgabenelement „OK“

Das skriptfähige Aufgabenelement „OK“ wird verständigt, dass die virtuelle Maschine erfolgreich gestartet wurde, legt den Inhalt einer E-Mail-Benachrichtigung über den erfolgreichen Start der virtuellen Maschine fest und schreibt das Ereignis in das Orchestrator-Protokoll.

In der folgenden Tabelle sind die Bindungen für Eingabe- und Ausgabeparameter enthalten, die das skriptfähige Aufgabenelement „OK“ benötigt.

**Tabelle 1-59.** Bindungen des skriptfähigen Aufgabenelements „OK“

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
vm	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vm</li> <li>■ Quellparameter: vm[in-parameter]</li> <li>■ Typ: VC:VirtualMachine</li> <li>■ Beschreibung: <b>Die einzuschaltende virtuelle Maschine.</b></li> </ul>
body	AUS	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: body</li> <li>■ Quellparameter: body[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Der Haupttext der E-Mail</b></li> </ul>

Das skriptfähige Aufgabenelement „OK“ erfordert die folgende Skriptfunktion.

```
body = "The VM '"+vm.name+"' has started successfully and is ready for use";
//Writes the following event in the Orchestrator database
Server.log(body);
```

#### **Skriptfähiges Aufgabenelement „E-Mail senden fehlgeschlagen“**

Das skriptfähige Aufgabenelement „E-Mail senden fehlgeschlagen“ empfängt eine Mitteilung, dass das Senden der E-Mail fehlgeschlagen ist, und schreibt das Ereignis in das Orchestrator-Protokoll.

In der folgenden Tabelle sind die Bindungen für Eingabeparameter enthalten, die das skriptfähige Aufgabenelement „E-Mail senden fehlgeschlagen“ benötigt.



**Tabelle 1-60.** Bindungen des skriptfähigen Aufgabenelements „E-Mail senden fehlgeschlagen“

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
vm	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vm</li> <li>■ Quellparameter: vm[in-parameter]</li> <li>■ Typ: VC:VirtualMachine</li> <li>■ Beschreibung: <b>Die einzuschaltende virtuelle Maschine.</b></li> </ul>
toAddress	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: toAddress</li> <li>■ Quellparameter: toAddress[in-parameter]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Die E-Mail-Adresse der Person, die über das Ergebnis dieses Workflows informiert werden soll</b></li> </ul>
emailErrorCode	EIN	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: emailErrorCode</li> <li>■ Quellparameter: emailErrorCode[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Abfangen von Ausnahmen beim Senden einer E-Mail</b></li> </ul>

Das skriptfähige Aufgabenelement „E-Mail senden fehlgeschlagen“ erfordert die folgende Skriptfunktion.

```
//Writes the following event in the Orchestrator database
Server.error("Couldn't send result email to '"+toAddress+"' for VM '"+vm.name+"', exception
found: "+emailErrorCode);
```

### Skriptfähiges Aufgabenelement „E-Mail senden“

Der Zweck des Workflows „VM starten und E-Mail senden“ besteht darin, einen Administrator zu informieren, wenn vom Workflow eine virtuelle Maschine gestartet wird. Dazu müssen Sie die skriptfähige Aufgabe definieren, die eine E-Mail versendet. Um die E-Mail zu versenden, benötigt die skriptfähige Aufgabe „E-Mail senden“ einen SMTP-Server, Adressen für den Absender und Empfänger der E-Mail, den E-Mail-Betreff und den E-Mail-Inhalt.

In der folgenden Tabelle sind die Bindungen für Eingabe- und Ausgabeparameter enthalten, die das skriptfähige Aufgabenelement „E-Mail senden“ benötigt.

**Tabelle 1-61.** Bindungen des skriptfähigen Aufgabenelements „E-Mail senden“

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
vm	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vm</li> <li>■ Quellparameter: vm[in-parameter]</li> <li>■ Typ: VC:VirtualMachine</li> <li>■ Beschreibung: <b>Die einzuschaltende virtuelle Maschine.</b></li> </ul>
toAddress	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: toAddress</li> <li>■ Quellparameter: toAddress[in-parameter]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Die E-Mail-Adresse der Person, die über das Ergebnis dieses Workflows informiert werden soll</b></li> </ul>
body	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: body</li> <li>■ Quellparameter: body[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Der Haupttext der E-Mail</b></li> </ul>
smtpHost	EIN	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: smtpHost</li> <li>■ Quellparameter: smtpHost[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Der SMTP-Server der E-Mail</b></li> </ul>
fromAddress	EIN	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: fromAddress</li> <li>■ Quellparameter: fromAddress[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Die E-Mail-Adresse des Absenders</b></li> </ul>
subject	EIN	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: subject</li> <li>■ Quellparameter: subject[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Der Betreff der E-Mail</b></li> </ul>

Das skriptfähige Aufgabenelement „E-Mail senden“ erfordert die folgende Skriptfunktion.

```
//Create an instance of EmailMessage
var myEmailMessage = new EmailMessage() ;

//Apply methods on this instance that populate the email message
myEmailMessage.smtpHost = smtpHost;
myEmailMessage.fromAddress = fromAddress;
myEmailMessage.toAddress = toAddress;
```

```
myEmailMessage.subject = subject;
myEmailMessage.addMimePart(body , "text/html");

//Apply the method that sends the email message
myEmailMessage.sendMessage();
System.log("Sent email to '"+toAddress+"'");
```

## Definieren der Ausnahmebindungen im einfachen Workflowbeispiel

Sie definieren Ausnahmebindungen im Workfloweditor auf der Registerkarte **Schema**. Ausnahmebindungen definieren, wie Elemente Fehler verarbeiten.

Die folgenden Elemente im Workflow geben Ausnahmen zurück: `startVM`, `vim3WaitTaskEnd`, `Send Email` und `vim3WaitToolsStarted`.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen einfachen Workflow“, auf Seite 146.
- „Erstellen Sie das Schema des Beispiels für einen einfachen Workflow“, auf Seite 147.
- „Definieren der Parameter für das einfache Workflowbeispiel“, auf Seite 151.
- „Definieren der Entscheidungsbindungen im einfachen Workflowbeispiel“, auf Seite 152.
- „Binden der Aktionselemente für das einfache Workflowbeispiel“, auf Seite 152.
- „Binden der skriptfähigen Aufgabenelemente für das einfache Workflowbeispiel“, auf Seite 155.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

### Vorgehensweise

- 1 Klicken Sie auf der Registerkarte **Schema** auf das Symbol **Bearbeiten** (✎) des **startVM** Aufgabenelements.
- 2 Klicken Sie auf die Registerkarte **Ausnahme**.
- 3 Klicken Sie auf **Nicht festgelegt**.
- 4 Wählen Sie **errorCode** aus der vorgeschlagenen Liste aus.
- 5 Wiederholen Sie die vorherigen Schritte, um die Ausnahmebindung an `errorCode` für `vim3WaitTaskEnd` und `vim3WaitToolsStarted` festzulegen.
- 6 Klicken Sie auf das Symbol **Bearbeiten** (✎) des **Send Email** skriptfähigen Aufgabenelements.
- 7 Klicken Sie auf die Registerkarte **Ausnahme**.
- 8 Klicken Sie auf **Nicht festgelegt**.
- 9 Wählen Sie **emailErrorCode** aus der vorgeschlagenen Liste aus.
- 10 Klicken Sie unten in der Workfloweditor-Registerkarte **Schema** auf **Speichern**.

Sie haben die Ausnahmebindung für die Elemente definiert, die Ausnahmen zurückgegeben haben.

### Weiter

Sie müssen die Lese- und Schreibigenschaften für die Attribute und Parameter festlegen.

## Setzen der Schreibschutzeigenschaften für Attribute des Beispiels für einen einfachen Workflow

Sie können definieren, ob Parameter und Attribute schreibgeschützte Konstanten oder beschreibbare Variablen sind. Sie können auch Einschränkungen für die Werte festlegen, die Benutzer für die Eingabeparameter eingeben können.

Wenn Sie bestimmte Parameter als schreibgeschützt definieren, können andere Entwickler den Workflow adaptieren oder ändern, ohne die Kernfunktionen des Workflows zu beeinträchtigen.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen einfachen Workflow“, auf Seite 146.
- „Erstellen Sie das Schema des Beispiels für einen einfachen Workflow“, auf Seite 147.
- „Definieren der Parameter für das einfache Workflowbeispiel“, auf Seite 151.
- „Definieren der Entscheidungsbindungen im einfachen Workflowbeispiel“, auf Seite 152.
- „Binden der Aktionselemente für das einfache Workflowbeispiel“, auf Seite 152.
- „Binden der skriptfähigen Aufgabenelemente für das einfache Workflowbeispiel“, auf Seite 155.
- „Definieren der Ausnahmebindungen im einfachen Workflowbeispiel“, auf Seite 163.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

### Vorgehensweise

- 1 Klicken Sie oben im Workfloweditor auf die Registerkarte **Allgemein**.

Unter **Attribute** befindet sich eine Liste aller definierten Attribute mit Kontrollkästchen neben jedem Attribut. Wenn Sie diese Kontrollkästchen aktivieren, definieren Sie Attribute als schreibgeschützt.

- 2 Wählen Sie die Kontrollkästchen, um folgende Attribute als schreibgeschützte Konstanten festzulegen:

- progress
- pollRate
- timeout
- smtpHost
- fromAddress
- subject

Sie haben definiert, welche Attribute des Workflows Konstanten oder Variablen sind.

### Weiter

Setzen Sie die Parametereigenschaften und platzieren Sie Integritätsregeln für die möglichen Werte dieses Parameters.

## Setzen der Parametereigenschaften des Beispiels für einen einfachen Workflow

Sie können die Parametereigenschaften im Workfloweditor setzen. Das Setzen von Parametereigenschaften beeinflusst das Verhalten des Parameters und legt Integritätsregeln für die möglichen Werte dieses Parameters fest.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen einfachen Workflow“, auf Seite 146.
- „Erstellen Sie das Schema des Beispiels für einen einfachen Workflow“, auf Seite 147.
- „Definieren der Parameter für das einfache Workflowbeispiel“, auf Seite 151.
- „Definieren der Entscheidungsbindungen im einfachen Workflowbeispiel“, auf Seite 152.
- „Binden der Aktionselemente für das einfache Workflowbeispiel“, auf Seite 152.
- „Binden der skriptfähigen Aufgabenelemente für das einfache Workflowbeispiel“, auf Seite 155.
- „Definieren der Ausnahmebindungen im einfachen Workflowbeispiel“, auf Seite 163.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Präsentation**.  
Die beiden von Ihnen für diesen Workflow definierten Eingabeparameter werden aufgelistet.
- 2 Klicken Sie auf den Parameter **(VC:VirtualMachine)vm**.
- 3 Fügen Sie eine Beschreibung auf der Registerkarte **Allgemein** in der unteren Hälfte des Bildschirms ein.  
Beispiel: **Die zu startende virtuelle Maschine**.
- 4 Klicken Sie auf die Registerkarte **Eigenschaften** in der unteren Hälfte des Bildschirms.  
Auf dieser Registerkarte können Sie die Eigenschaften für den Parameter (VC:VirtualMachine)vm festlegen.
- 5 Klicken Sie auf das Symbol **Eigenschaft hinzufügen** (🔍+).
- 6 Wählen Sie aus der Liste der vorgeschlagenen Eigenschaften die Eigenschaft **Erforderliche Eingabe**, klicken Sie auf **OK** und legen Sie den Wert mit **Ja** fest.  
Wenn Sie diese Eigenschaft aktivieren, können Benutzer den Workflow „VM starten und E-Mail senden“ nur starten, wenn sie eingegeben haben, welche virtuelle Maschine zu starten ist.
- 7 Klicken Sie auf das Symbol **Eigenschaft hinzufügen** (🔍+).
- 8 Wählen Sie aus der Liste der vorgeschlagenen Eigenschaften **Wert auswählen als**, klicken Sie auf **OK** und wählen Sie **Liste** aus der Liste der möglichen Werte aus.  
Wenn Sie diese Eigenschaft festlegen, bestimmen Sie, wie der Benutzer den Wert des Eingabeparameters (VC:VirtualMachine)vm auswählt.
- 9 Klicken Sie auf den Parameter **(string)toAddress** in der oberen Hälfte der Registerkarte **Präsentation**.
- 10 Fügen Sie eine Beschreibung auf der Registerkarte **Beschreibung** in der unteren Hälfte des Bildschirms ein.  
Beispiel: Geben Sie **Die E-Mail-Adresse der Person, die benachrichtigt werden soll** ein.

- 11 Klicken Sie auf die Registerkarte **Eigenschaften** für (string)toAddress und klicken Sie auf das Symbol **Eigenschaft hinzufügen** (➤+).
- 12 Wählen Sie aus der Liste der vorgeschlagenen Eigenschaften die Eigenschaft **Erforderliche Eingabe**, klicken Sie auf **OK** und legen Sie den Wert mit **Ja** fest.
- 13 Klicken Sie auf das Symbol **Eigenschaft hinzufügen** (➤+).
- 14 Wählen Sie aus der Liste der vorgeschlagenen Eigenschaften die Option **Übereinstimmung mit regulärem Ausdruck** und klicken Sie auf **OK**.  
Diese Eigenschaft ermöglicht Ihnen, Integritätsregeln für die Eingaben der Benutzer aufzustellen.
- 15 Klicken Sie auf das Textfeld **Wert** für **Übereinstimmung mit regulärem Ausdruck** und legen Sie die Integritätsregel wie folgt fest: `[a-zA-Z0-9_%-+.]+@[a-zA-Z0-9-+.]+\.[a-zA-Z]{2,4}`.  
Durch diese Integritätsregeln wird die Eingabe von Benutzern auf Zeichen beschränkt, die für E-Mail-Adressen zulässig sind. Wenn der Benutzer versucht, ein anderes Zeichen für die E-Mail-Adresse des Empfängers für den Start des Workflows einzugeben, startet der Workflow nicht.

Sie haben beide Parameter als erforderlich festgelegt, definiert, wie der Benutzer die virtuelle Maschine für den Start auswählen kann, und die Zeichen beschränkt, die ein Benutzer für die E-Mail-Adresse des Empfängers eingeben kann.

#### Weiter

Sie müssen das Layout oder die Präsentation des Dialogfelds für die Eingabeparameter erstellen, in dem die Benutzer die Werte von Eingabeparametern für den Workflow bestimmen, wenn Sie diesen ausführen.

## Setzen des Layouts für das Dialogfeld der Eingabeparameter des Beispiels für einen komplexen Workflow

Sie erstellen das Layout oder die Präsentation des Dialogfelds für die Eingabeparameter im Workfloweditor. Das Dialogfeld für die Eingabeparameter erscheint, wenn Benutzer einen Workflow ausführen, für den Eingabeparameter erforderlich sind.

#### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen einfachen Workflow“, auf Seite 146.
- „Erstellen Sie das Schema des Beispiels für einen einfachen Workflow“, auf Seite 147.
- „Definieren der Parameter für das einfache Workflowbeispiel“, auf Seite 151.
- „Definieren der Entscheidungsbindungen im einfachen Workflowbeispiel“, auf Seite 152.
- „Binden der Aktionselemente für das einfache Workflowbeispiel“, auf Seite 152.
- „Binden der skriptfähigen Aufgabenelemente für das einfache Workflowbeispiel“, auf Seite 155.
- „Definieren der Ausnahmebindungen im einfachen Workflowbeispiel“, auf Seite 163.
- „Setzen der Schreibschutzigenschaften für Attribute des Beispiels für einen einfachen Workflow“, auf Seite 164.
- „Setzen der Parametereigenschaften des Beispiels für einen einfachen Workflow“, auf Seite 165.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

#### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Präsentation**.

- 2 Klicken Sie mit der rechten Maustaste auf den Knoten **Präsentation** in der hierarchischen Liste für die Präsentation und wählen Sie **Anzeigegruppe erstellen**.

Ein Knoten **Neuer Schritt** und ein Unterknoten **Neue Gruppe** erscheinen unter dem Knoten **Präsentation**.

- 3 Klicken Sie mit der rechten Maustaste auf **Neuer Schritt** und wählen Sie **Löschen**.

Da dieser Workflow nur über zwei Parameter verfügt, benötigen Sie nicht mehrere Ebenen von Abschnitten im Dialogfeld für die Eingabeparameter.

- 4 Doppelklicken Sie auf **Neue Gruppe**, um den Gruppennamen zu bearbeiten. Drücken Sie die Eingabetaste.

Beispiel: Benennen Sie die Anzeigegruppe **Virtuelle Maschine**.

Der Text, den Sie hier eingeben, erscheint als Überschrift im Dialogfeld der Eingabeparameter, wenn Benutzer den Workflow starten.

- 5 Geben Sie im Textfeld **Beschreibung** auf der Registerkarte **Allgemein** unten auf der Registerkarte **Präsentation** eine Beschreibung für die neue Anzeigegruppe ein.

Beispiel: **Die zu startende virtuelle Maschine auswählen**.

Der Text, den Sie hier eingeben, erscheint als Eingabeaufforderung im Dialogfeld der Eingabeparameter, wenn Benutzer den Workflow starten.

- 6 Ziehen Sie den Parameter **(VC:VirtualMachine)vm** unter die Anzeigegruppe **Virtuelle Maschine**.

Im Dialogfeld für die Eingabeparameter erscheint ein Textfeld, in das der Benutzer den Namen der virtuellen Maschine eingibt, unter der Überschrift „Virtuelle Maschine“.

- 7 Wiederholen Sie die vorangehenden Schritte, um eine Anzeigegruppe für den Parameter **toAddress** zu erstellen, indem Sie die folgenden Eigenschaften einstellen:

- a Erstellen Sie eine Anzeigegruppe und benennen Sie sie als **E-Mail-Adresse des Empfängers**.
- b Fügen Sie eine Beschreibung für die Anzeigegruppe hinzu, beispielsweise **Geben Sie die E-Mail-Adresse der Person ein, die zu verständigen ist, wenn diese virtuelle Maschine eingeschaltet wird**.
- c Ziehen Sie den Parameter **toAddress** unter die Anzeigegruppe **E-Mail-Adresse des Empfängers**.

Sie haben das Layout des Dialogfelds für die Eingabeparameter erstellt, das erscheint, wenn Benutzer den Workflow ausführen.

### Weiter

Sie haben die Entwicklung des Beispiels für einen einfachen Workflow abgeschlossen. Sie können jetzt den Workflow validieren und ausführen.

## Validieren und Ausführen des Beispiels für einen einfachen Workflow

Nachdem Sie einen Workflow erstellt haben, können Sie ihn validieren, um eventuelle Fehler zu erkennen. Wenn der Workflow keine Fehler enthält, können Sie ihn ausführen.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen einfachen Workflow“, auf Seite 146.
- „Erstellen Sie das Schema des Beispiels für einen einfachen Workflow“, auf Seite 147.
- „Definieren der Parameter für das einfache Workflowbeispiel“, auf Seite 151.

- „Definieren der Entscheidungsbindungen im einfachen Workflowbeispiel“, auf Seite 152.
- „Binden der Aktionselemente für das einfache Workflowbeispiel“, auf Seite 152.
- „Binden der skriptfähigen Aufgabenelemente für das einfache Workflowbeispiel“, auf Seite 155.
- „Definieren der Ausnahmebindungen im einfachen Workflowbeispiel“, auf Seite 163.
- „Setzen der Schreibschutzeigenschaften für Attribute des Beispiels für einen einfachen Workflow“, auf Seite 164.
- „Setzen der Parametereigenschaften des Beispiels für einen einfachen Workflow“, auf Seite 165.
- „Setzen des Layouts für das Dialogfeld der Eingabeparameter des Beispiels für einen komplexen Workflow“, auf Seite 166.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

### Vorgehensweise

- 1 Klicken Sie auf **Validieren** auf der Registerkarte **Schema** des Workfloweditors.  
Das Validierungstool erkennt Fehler in der Definition des Workflows.
- 2 Nachdem Sie Fehler behoben haben, klicken Sie auf **Speichern und schließen** unten im Workfloweditor.  
Kehren Sie zum Orchestrator-Client zurück.
- 3 Klicken Sie auf die Ansicht **Workflows**.
- 4 Wählen Sie **Workflowbeispiele > VM starten und E-Mail senden** in der hierarchischen Liste der Workflows.
- 5 Klicken Sie mit der rechten Maustaste auf den Workflow **VM starten und E-Mail senden** und wählen Sie **Workflow starten** aus.  
Das Dialogfeld der Eingabeparameter wird geöffnet und fordert Sie auf, eine virtuelle Maschine für den Start und eine E-Mail-Adresse für den Versand von Benachrichtigungen anzugeben.
- 6 Wählen Sie eine virtuelle Maschine für den Start aus dem vCenter Server-Bestand.
- 7 Geben Sie eine E-Mail-Adresse ein, an die E-Mail-Benachrichtigungen geschickt werden sollen.
- 8 Klicken Sie auf **Übernehmen**, um den Workflow zu starten.  
Ein Workflowtoken erscheint unter dem Workflow „VM starten und E-Mail senden“.
- 9 Klicken Sie auf den Workflowtoken, um den Fortschritt des Workflows während des Ausführens zu beobachten.

Wenn der Workflow erfolgreich ausgeführt wird, befindet sich die von Ihnen ausgewählte virtuelle Maschine im eingeschalteten Status und der von Ihnen definierte E-Mail-Empfänger erhält eine Bestätigungs-E-Mail.

### Weiter

Sie können ein Dokument erstellen, in dem Informationen über den Workflow überprüft werden können. Siehe „Generieren einer Workflow-Dokumentation“, auf Seite 143.



## Entwickeln eines komplexen Workflows

Das Entwickeln eines komplexen Beispielworkflows zeigt die häufigsten Schritte im Entwicklungsprozess eines Workflows und erweiterte Szenarien, etwa das Erstellen von benutzerdefinierten Entscheidungen und Schleifen.

Bei der Übung zum komplexen Workflow entwickeln Sie einen Workflow, der Snapshots aller virtuellen Maschinen in einem bestimmten Ressourcenpool erstellt. Der von Ihnen erstellte Workflow führt folgende Aufgaben aus:

- 1 Fordert vom Benutzer einen Ressourcenpool an, der die virtuellen Maschinen für Snapshots enthält.
- 2 Bestimmt, ob der Ressourcenpool laufende virtuelle Maschinen enthält.
- 3 Bestimmt, wie viele laufende virtuelle Maschinen die Ressource enthält.
- 4 Überprüft, ob eine einzelne virtuelle Maschine, die im Pool ausgeführt wird, bestimmte Kriterien für einen Snapshot erfüllt.
- 5 Erstellt den Snapshot der virtuellen Maschine.
- 6 Bestimmt, ob weitere virtuelle Maschinen im Pool vorhanden sind, von denen Snapshots erstellt werden.
- 7 Wiederholt die Überprüfung und Snapshot-Erstellung, bis der Workflow Snapshots aller geeigneten virtuellen Maschinen im Ressourcenpool erstellt hat.

Die ZIP-Datei von Orchestrator-Beispielen, die Sie von der Startseite der Orchestrator-Dokumentation herunterladen können, enthält eine vollständige Version des Workflows „Snapshot aller virtuellen Maschinen in einem Ressourcenpool erstellen“.

### Voraussetzungen

Bevor Sie versuchen, diesen komplexen Workflow zu entwickeln, absolvieren Sie die Übungen in „[Entwickeln eines einfachen Beispielworkflows](#)“, auf Seite 144. Die Verfahren zum Entwickeln eines komplexen Workflows liefern die breiten Schritte des Entwicklungsprozesses, sind aber nicht so ausführlich wie die Übungen des einfachen Workflows.

### Vorgehensweise

- 1 [Erstellen des Beispiels für einen komplexen Workflow](#) auf Seite 170  
Sie müssen den Entwicklungsprozess des Workflows beginnen, indem Sie den Workflow im Orchestrator-Client erstellen.
- 2 [Erstellen einer benutzerdefinierten Aktion für das Beispiel eines komplexen Workflows](#) auf Seite 171  
Das Skriptelement `Check VM` ruft eine Aktion auf, die in der Orchestrator-API nicht vorhanden ist. Sie müssen die Aktion `getVMDiskModes` erstellen.
- 3 [Erstellen des Schemas für das Beispiel eines komplexen Workflows](#) auf Seite 172  
Sie können das Schema eines Workflows im Workfloweditor erstellen. Das Workflowschema enthält die Elemente, die im Workflow ausgeführt werden, und bestimmt den logischen Fluss des Workflows.
- 4 [\(Optional\) Erstellen der Zonen für das Beispiel eines komplexen Workflows](#) auf Seite 174  
Optional können Sie verschiedene Zonen des Workflows hervorheben, indem Sie Workflowanmerkungen hinzufügen. Durch das Erstellen verschiedener Workflowzonen erleichtern Sie das Lesen und Verstehen eines komplexen Workflowschemas.
- 5 [Definieren der Parameter für das komplexe Workflowbeispiel](#) auf Seite 176  
Sie definieren Workflowparameter im Workfloweditor. Die Eingabeparameter bieten Daten für den Workflow zur Verarbeitung. Die Ausgabeparameter sind die Daten, die der Workflow nach der Ausführung zurückgibt.

- 6 [Definieren der Bindungen für das komplexe Workflowbeispiel](#) auf Seite 176  
Sie können die Elemente eines Workflows im Workfloweditor zusammenbinden. Bindungen definieren den Datenfluss des Workflows. Sie können auch die skriptfähigen Aufgabenelemente an ihre JavaScript-Funktionen binden.
- 7 [Setzen der Attributeigenschaften im Beispiel für einen komplexen Workflow](#) auf Seite 187  
Die Attributeigenschaften werden auf der Registerkarte **Allgemein** im Workfloweditor gesetzt.
- 8 [Erstellen des Layouts der Eingabeparameter für das Beispiel eines komplexen Workflows](#) auf Seite 187  
Sie erstellen das Layout oder die Präsentation des Dialogfelds für die Eingabeparameter auf der Registerkarte **Präsentation** des Workfloweditors. Das Dialogfeld für die Eingabeparameter erscheint, wenn Benutzer einen Workflow ausführen. Hier geben die Benutzer die Eingabeparameter ein, mit denen der Workflow ausgeführt wird.
- 9 [Validieren und Ausführen des Beispiels für einen komplexen Workflow](#) auf Seite 188  
Nachdem Sie einen Workflow erstellt haben, können Sie ihn validieren, um eventuelle Fehler zu erkennen. Wenn der Workflow keine Fehler enthält, können Sie ihn ausführen.

## Erstellen des Beispiels für einen komplexen Workflow

Sie müssen den Entwicklungsprozess des Workflows beginnen, indem Sie den Workflow im Orchestrator-Client erstellen.

Informationen zur Installation und Konfiguration von vCenter Server finden Sie in der Dokumentation *Installation und Einrichtung von vSphere*. Informationen zur Konfiguration von Orchestrator finden Sie unter *Installieren und Konfigurieren von VMware vRealize Orchestrator*

### Voraussetzungen

Überprüfen Sie, ob die folgenden Komponenten im System installiert und konfiguriert sind.

- vCenter Server für die Steuerung eines Ressourcenpools, der einige virtuelle Maschinen enthält
- Ordner **Workflowbeispiele** in der hierarchischen Liste der Workflows, die Sie in „[Erstellen des Beispiels für einen einfachen Workflow](#)“, auf Seite 146 erstellt haben.

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Wählen Sie **Workflows > Workflowbeispiele**.
- 3 Klicken Sie mit der rechten Maustaste auf den Ordner **Workflowbeispiele** und wählen Sie **Neuer Workflow**.
- 4 Nennen Sie den Workflow **Snapshots von allen virtuellen Maschinen in einem Ressourcenpool erstellen** und klicken Sie auf **OK**.  
Der Workfloweditor wird geöffnet.
- 5 Klicken Sie auf der Registerkarte **Allgemein** im Workfloweditor auf die Ziffern der Versionsnummer, um die Versionsnummer zu erhöhen.  
Bei der ersten Erstellung des Workflows setzen Sie die Version auf **0.0.1**.
- 6 Klicken Sie auf den Wert **Serverneustart-Verhalten**, um festzulegen, ob der Workflow nach einem Serverneustart fortgesetzt wird.
- 7 Geben Sie im Textfeld **Beschreibung** eine Beschreibung der Aufgabe des Workflows ein.
- 8 Klicken Sie unten auf der Registerkarte **Allgemein** auf **Speichern**.

Sie haben den Workflow „Snapshots von allen virtuellen Maschinen in einem Ressourcenpool erstellen“ erstellt.

### Weiter

Sie müssen eine benutzerdefinierte Aktion erstellen.

## Erstellen einer benutzerdefinierten Aktion für das Beispiel eines komplexen Workflows

Das Skriptelement `Check VM` ruft eine Aktion auf, die in der Orchestrator-API nicht vorhanden ist. Sie müssen die Aktion `getVMDiskModes` erstellen.

Weitere Informationen über das Erstellen von Aktionen finden Sie unter [Kapitel 3, „Entwicklungsaktionen“](#), auf Seite 209.

### Voraussetzungen

Erstellen Sie den Workflow „Snapshots von allen virtuellen Maschinen in einem Ressourcenpool erstellen“. Weitere Informationen finden Sie unter [„Erstellen des Beispiels für einen komplexen Workflow“](#), auf Seite 170.

### Vorgehensweise

- 1 Schließen Sie den Workfloweditor mit einem Klick auf **Speichern und schließen**.
- 2 Klicken Sie im Orchestrator-Client auf die Ansicht **Aktionen**.
- 3 Klicken Sie auf die Stammebene der hierarchischen Liste für Aktionen und wählen Sie **Neues Modul**.
- 4 Nennen Sie das neue Modul **com.vmware.example**.
- 5 Klicken Sie mit der rechten Maustaste auf das Modul **com.vmware.example** und wählen Sie **Aktionen hinzufügen**.
- 6 Erstellen Sie eine Aktion mit der Bezeichnung `getVMDiskModes`.
- 7 Inkrementieren Sie die Versionsnummer auf der Registerkarte **Allgemein** in der Bearbeitungsfunktion für Aktionen, indem Sie auf die Versionsnummer klicken.
- 8 Fügen Sie die folgende Beschreibung der Aktion auf der Registerkarte **Allgemein** hinzu.  
  
 This action returns an array containing the disk modes of all disks on a VM.  
 The elements in the array each have one of the following string values:  
 – persistent  
 – independent-persistent  
 – nonpersistent  
 – independent-nonpersistent  
 Legacy values:  
 – undoable  
 – append
- 9 Klicken Sie auf die Registerkarte **Skripterstellung**.
- 10 Klicken Sie mit der rechten Maustaste im oberen Bereich der Registerkarte **Skripterstellung** und wählen Sie **Parameter hinzufügen** aus, um den folgenden Eingabeparameter zu erstellen.

- Name: `vm`
- Typ: `VC:VirtualMachine`
- Beschreibung:  
**Die virtuelle Maschine, für die Informationen über den Festplattenmodus zurückgegeben werden sollen**

- 11 Fügen Sie den folgenden Skriptcode unten auf der Registerkarte **Skripterstellung** hinzu.

Der folgende Code gibt ein Array von Festplattenmodi für die Festplatten der virtuellen Maschine zurück.

```
var devicesArray = vm.config.hardware.device;
var retArray = new Array();
if (devicesArray!=null && devicesArray.length!=0) {
    for (i in devicesArray) {
        if (devicesArray[i] instanceof VcVirtualDisk) {
            retArray.push(devicesArray[i].backing.diskMode);
        }
    }
}
return retArray;
```

- 12 Klicken Sie auf **Speichern und schließen**, um die Palette **Aktionen** zu verlassen.

Sie haben die benutzerdefinierte Aktion erstellt, die vom Workflow „Snapshots von allen virtuellen Maschinen in einem Ressourcenpool erstellen“ benötigt wird.

### Weiter

Erstellen Sie das Schema des Workflows.

## Erstellen des Schemas für das Beispiel eines komplexen Workflows

Sie können das Schema eines Workflows im Workfloweditor erstellen. Das Workflowschema enthält die Elemente, die im Workflow ausgeführt werden, und bestimmt den logischen Fluss des Workflows.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen komplexen Workflow“, auf Seite 170.
- „Erstellen einer benutzerdefinierten Aktion für das Beispiel eines komplexen Workflows“, auf Seite 171.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Schema**.
- 2 Fügen Sie dem Workflowschema die folgenden Schemaelemente hinzu.

Elementtyp	Elementname	Position im Schema
Skriptfähige Aufgabe	<b>Initializing</b>	Unter dem Start-Element
Entscheidung	<b>VMs to Process?</b>	Unter dem skriptfähigen Aufgabenelement <b>Initializing</b>
Skriptfähige Aufgabe	<b>Pool Has No VMs</b>	Unter dem Element <b>VMs to Process?</b> für eine benutzerdefinierte Entscheidung, verknüpft mit einem roten Pfeil
Benutzerdefinierte Entscheidung	<b>Remaining VMs?</b>	Rechts vom Element <b>VMs to Process?</b> für eine benutzerdefinierte Entscheidung, verknüpft mit einem grünen Pfeil
Aktion	<b>getVMDiskModes</b>	Rechts vom Element <b>Remaining VMs?</b> für eine benutzerdefinierte Entscheidung, verknüpft mit einem grünen Pfeil
Benutzerdefinierte Entscheidung	<b>Create Snapshot?</b>	Rechts vom Aktionselement <b>getVMDiskModes</b> , verknüpft mit einem blauen Pfeil
Workflow	<b>Create a snapshot</b>	Über dem Element <b>Create Snapshot?</b> für eine benutzerdefinierte Entscheidung, verknüpft mit einem grünen Pfeil

Elementtyp	Elementname	Position im Schema
Skriptfähige Aufgabe	<b>VM Snapshots</b>	Links vom Workflow Create a snapshot, verknüpft mit einem blauen Pfeil
Skriptfähige Aufgabe	<b>Increment</b>	Links vom Element VM Snapshots für eine skriptfähige Aufgabe, verknüpft mit einem blauen Pfeil
Skriptfähige Aufgabe	<b>Set Output</b>	Rechts vom Element Pool Has No VMs für eine skriptfähige Aufgabe, verknüpft mit einem blauen Pfeil

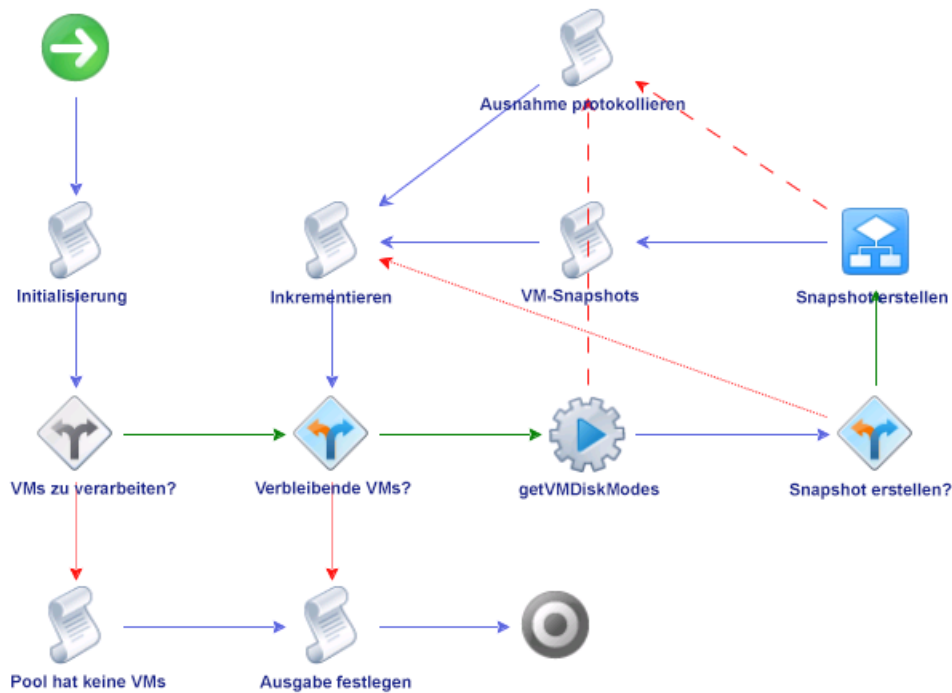
- 3 Fügen Sie ein skriptfähiges Aufgabenelement für Log Exception hinzu.
  - a Erstellen Sie einen Ausnahmebehandlungslink vom Workflow „Snapshot erstellen“ zu einem End-Element.
  - b Ziehen Sie ein skriptfähiges Aufgabenelement auf den roten gestrichelten Pfeil, der den Workflow „Snapshot erstellen“ mit einem End-Element verbindet.
  - c Doppelklicken Sie auf das skriptfähige Aufgabenelement und benennen Sie es in **Ausnahme protokollieren** um.
  - d Verschieben Sie das skriptfähige Aufgabenelement Log Exception über das skriptfähige Aufgabenelement VM Snapshots.
- 4 Entfernen Sie die Verknüpfung aller End-Elemente mit Ausnahme des End-Elements, das rechts vom skriptfähigen Aufgabenelement Set Output steht.
- 5 Verknüpfen Sie die restlichen Elemente wie in der folgenden Tabelle beschrieben.

Element	Verknüpfung mit	Art des Pfeils	Beschreibung
getVMDiskModes-Aktionselement	Log Exception-Element für eine skriptfähige Aufgabe	Rot gestrichelt	Ausnahmebehandlung
Create Snapshot?-Element für eine benutzerdefinierte Entscheidung	Increment-Element für eine skriptfähige Aufgabe	Rot	False-Ergebnis
Log Exception-Element für eine skriptfähige Aufgabe	Increment-Element für eine skriptfähige Aufgabe	Blau	Normaler Workflowfortschritt
Increment-Element für eine skriptfähige Aufgabe	Remaining VMs?-Element für eine benutzerdefinierte Entscheidung	Blau	Normaler Workflowfortschritt
Remaining VMs?-Element für eine benutzerdefinierte Entscheidung	Set Output-Element für eine skriptfähige Aufgabe	Rot	False-Ergebnis

- 6 Klicken Sie unten auf der Registerkarte **Schema** auf **Speichern**.

Die folgende Abbildung zeigt, wie die verknüpften Elemente des Workflows „Snapshots von allen virtuellen Maschinen in einem Ressourcenpool erstellen“ aussehen sollten.

**Abbildung 1-12.** Verknüpfen des Beispielworkflows „Snapshots von allen virtuellen Maschinen in einem Ressourcenpool erstellen“



#### Weiter

Sie können optional Workflowzonen definieren, indem Sie Workflowanmerkungen verwenden.

### (Optional) Erstellen der Zonen für das Beispiel eines komplexen Workflows

Optional können Sie verschiedene Zonen des Workflows hervorheben, indem Sie Workflowanmerkungen hinzufügen. Durch das Erstellen verschiedener Workflowzonen erleichtern Sie das Lesen und Verstehen eines komplexen Workflowschemas.

#### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen komplexen Workflow“, auf Seite 170.
- „Erstellen des Schemas für das Beispiel eines komplexen Workflows“, auf Seite 172.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

#### Vorgehensweise

- 1 Erstellen Sie die folgenden Workflowzonen Mithilfe von Workflowanmerkungen.

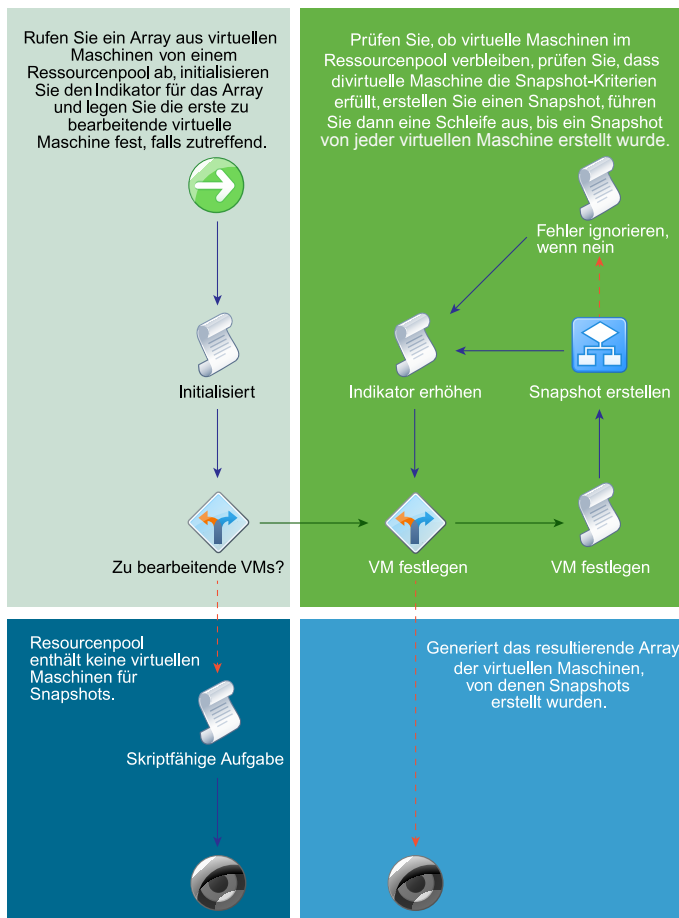
Elemente in der Zone	Beschreibung
<b>Startelement; „Initialisieren“</b> (skriptfähige Aufgabe); „Zu verarbeitende VMs?“ (benutzerdefinierte Entscheidung)	Rufen Sie ein Array aus virtuellen Maschinen von einem Ressourcenpool ab, initialisieren Sie den Indikator für das Array und legen Sie die erste zu bearbeitende virtuelle Maschine fest, falls zutreffend.
<b>Der Pool verfügt über keine skriptfähige Aufgabe für VMs.</b>	Ressourcenpool enthält keine VMs für Snapshots.

Elemente in der Zone	Beschreibung
„Verbleibende VMs?“ (benutzerdefinierte Entscheidung); getVMDisks-Modes (Aktion); „Snapshot erstellen?“ (Entscheidung); „Snapshot erstellen“ (Workflow); „VM-Snapshots“ (skriptfähige Aufgabe); „Inkrement“ (skriptfähige Aufgabe); „Ausnahme protokollieren“ (skriptfähige Aufgabe)	Prüfen Sie, ob virtuelle Maschinen im Ressourcenpool verbleiben, prüfen Sie, dass eine virtuelle Maschine die Snapshot-Kriterien erfüllt, erstellen Sie einen Snapshot, führen Sie dann eine Schleife aus, bis ein Snapshot von jeder virtuellen Maschine erstellt wurde.
„Ausgabe festlegen“ (skriptfähige Aufgabe); Endelement	Generiert das resultierende Array der virtuellen Maschinen, von denen Snapshots erstellt wurden.

- 2 Wählen Sie eine Workflowanmerkung aus und verwenden Sie Strg+E, um die Hintergrundfarbe auszuwählen.
- 3 Klicken Sie unten auf der Registerkarte **Schema** des Workfloweditors auf **Speichern**.

Ihre Workflowzonen müssten so aussehen wie im folgenden Diagramm gezeigt.

**Abbildung 1-13.** Schemadiagramm für den Beispielworkflow „Snapshots von allen virtuellen Maschinen in einem Ressourcenpool erstellen“



## Weiter

Sie müssen die Eingabe- und Ausgabeparameter des Workflows definieren.

## Definieren der Parameter für das komplexe Workflowbeispiel

Sie definieren Workflowparameter im Workfloweditor. Die Eingabeparameter bieten Daten für den Workflow zur Verarbeitung. Die Ausgabeparameter sind die Daten, die der Workflow nach der Ausführung zurückgibt.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- [„Erstellen des Beispiels für einen komplexen Workflow“](#), auf Seite 170.
- [„Erstellen des Schemas für das Beispiel eines komplexen Workflows“](#), auf Seite 172.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Eingaben**.
- 2 Definieren Sie die folgenden Eingabeparameter.
  - Name: resourcePool
  - Typ: VC:ResourcePool
  - Beschreibung: **Der Ressourcenpool mit den virtuellen Maschinen für Snapshots.**
- 3 Klicken Sie im Workfloweditor auf die Registerkarte **Ausgaben**.
- 4 Definieren Sie die folgenden Ausgabeparameter.
  - Name: snapshotVmArrayOut
  - Typ: Array/VC:VirtualMachine
  - Beschreibung: **Das Array der virtuellen Maschinen, von denen Snapshots erstellt wurden.**

Sie haben die Eingabe- und Ausgabeparameter des Workflows definiert.

### Weiter

Sie müssen die Bindungen zwischen den Elementparametern definieren.

## Definieren der Bindungen für das komplexe Workflowbeispiel

Sie können die Elemente eines Workflows im Workfloweditor zusammenbinden. Bindungen definieren den Datenfluss des Workflows. Sie können auch die skriptfähigen Aufgabenelemente an ihre JavaScript-Funktionen binden.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- [„Erstellen des Beispiels für einen komplexen Workflow“](#), auf Seite 170.
- [„Erstellen des Schemas für das Beispiel eines komplexen Workflows“](#), auf Seite 172
- [„Definieren der Parameter für das komplexe Workflowbeispiel“](#), auf Seite 176
- Überprüfen Sie die Bindungen, die Sie definieren müssen. Siehe [„Bindungen für komplexes Workflowbeispiel“](#), auf Seite 177.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.



## Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Schema**.
- 2 Definieren Sie die Bindungen.
- 3 Klicken Sie unten in der Registerkarte **Schema** auf **Speichern**.

Alle Eingabe- und Ausgabeparameter der Elemente werden an die entsprechenden Parametertypen und -werte gebunden.

## Weiter

Legen Sie die Attributeigenschaften fest.

## Bindungen für komplexes Workflowbeispiel

Bindungen definieren, wie die Aktionselemente des einfachen Workflowbeispiels Eingabe- und Ausgabeparameter verarbeiten.

Der Workflow „Snapshots von allen virtuellen Maschinen in einem Ressourcenpool erstellen“ benötigt die folgenden Bindungen für Eingabe- und Ausgabeparameter. Sie können auch die JavaScript-Funktionen für die skriptfähigen Ausgabenelemente definieren.

In Fällen, in denen Sie an vorhandene Parameter binden, übernimmt die Bindung die Typ- und Beschreibungswerte vom ursprünglichen Parameter.

### Skriptfähige Aufgabe „Initialisierung“

Das skriptfähige Aufgabenelement „Initialisierung“ initialisiert die Attribute des Workflows. In der folgenden Tabelle sind die Bindungen für Eingabe- und Ausgabeparameter enthalten, die das skriptfähige Aufgabenelement „Initialisierung“ benötigt.

**Tabelle 1-62.** Bindungen des skriptfähigen Aufgabenelements „Initialisierung“

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
resourcePool	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: resourcePool</li> <li>■ Quellparameter: resourcePool[in-parameter]</li> <li>■ Typ: VC:ResourcePool</li> <li>■ Beschreibung: <b>Der Ressourcenpool enthält die virtuellen Maschinen, von denen Snapshots angefertigt werden sollen</b></li> </ul>
allVMs	AUS	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: allVMs</li> <li>■ Quellparameter: allVMs[attribute]</li> <li>■ Typ: Array/VC:VirtualMachine</li> <li>■ Beschreibung: <b>Die virtuellen Maschinen im Ressourcenpool</b></li> </ul>

**Tabelle 1-62.** Bindungen des skriptfähigen Aufgabenelements „Initialisierung“ (Fortsetzung)

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
numberOfVms	AUS	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: numberOfVms</li> <li>■ Quellparameter: numberOfVms[attribute]</li> <li>■ Typ: Zahl</li> <li>■ Beschreibung: <b>Die Anzahl virtueller Maschinen im resourcePool</b></li> </ul>
vmCounter	AUS	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vmCounter</li> <li>■ Quellparameter: vmCounter[attribute]</li> <li>■ Typ: Zahl</li> <li>■ Beschreibung: <b>Die Anzahl der virtuellen Maschinen innerhalb eines Arrays</b></li> </ul>
vm	AUS	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vm</li> <li>■ Quellparameter: vm[attribute]</li> <li>■ Typ: VC:VirtualMachine</li> <li>■ Beschreibung: <b>Die aktuelle virtuelle Maschine mit erstelltem Snapshot</b></li> </ul>
snapshotVmArray	AUS	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: snapshotVmArray</li> <li>■ Quellparameter: snapshotVmArray[attribute]</li> <li>■ Typ: Array/VC:VirtualMachine</li> <li>■ Beschreibung: <b>Das Array der virtuellen Maschinen, von denen Snapshots erstellt wurden</b></li> </ul>

Das skriptfähige Aufgabenelement „Initialisierung“ führt die folgende Skriptfunktion durch.

```
//Retrieve an array of virtual machines contained in the specified Resource Pool
allVms = resourcePool.vm;
//Initialize the size of the Array and the first VM to snapshot
if (allVms!=null && allVms.length!=0) {
    numberOfVms = allVms.length;
    vm = allVms[0];
} else {
    numberOfVms = 0;
}
//Initialize the VM counter
vmCounter = 0;
//Initializing the array of VM snapshots
snapshotVmArray = new Array();
```

### Zu verarbeitende VMs? – Entscheidungselement

Das Entscheidungselement „Zu verarbeitenden VMs?“ ermittelt, ob sämtliche virtuellen Maschinen, von denen Snapshots erstellt werden sollen, im Ressourcenpool vorhanden sind. Die folgende Tabelle zeigt die Bindungen, die das Entscheidungselement „Zu verarbeitende VMs?“ benötigt.

**Tabelle 1-63.** Bindungen des Entscheidungselements „Zu verarbeitende VMs?“

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
numberOfVMs	Entscheidung	Binden	<ul style="list-style-type: none"> <li>■ Quellparameter: numberOfVMs[attribute]</li> <li>■ Entscheidungsanweisung: Größer als</li> <li>■ Wert: 0.0</li> <li>■ Beschreibung: <b>Die Anzahl virtueller Maschinen im resourcePool</b></li> </ul>

### Skriptfähiges Aufgabenelement „Pool verfügt über keine VMs“

Das skriptfähige Aufgabenelement „Pool verfügt über keine VMs“ protokolliert die Tatsache, dass der Ressourcenpool keine berechtigten virtuellen Maschinen in der Orchestrator-Datenbank enthält. Die folgende Tabelle zeigt die Bindungen, die das skriptfähige Aufgabenelement „Pool verfügt über keine VMs“ benötigt.

**Tabelle 1-64.** Bindungen des skriptfähigen Aufgabenelements „Pool verfügt über keine VMs“

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
resourcePool	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: resourcePool</li> <li>■ Quellparameter: resourcePool[in-parameter]</li> <li>■ Typ: VC:ResourcePool</li> <li>■ Beschreibung: <b>Der Ressourcenpool mit den virtuellen Maschinen, von denen Snapshots angefertigt werden sollen.</b></li> </ul>

Das skriptfähige Aufgabenelement „Pool verfügt über keine VMs“ führt die folgende Skriptfunktion durch.

```
//Writes the following event in the Orchestrator database
Server.warn("The specified ResourcePool "+resourcePool.name+" does not contain any VMs.");
```

### Verbleibende VMs? – benutzerdefiniertes Entscheidungselement

Das benutzerdefinierte Entscheidungselement „Verbleibende VMs?“ bestimmt, ob sämtliche virtuellen Maschinen, von denen Snapshots erstellt werden sollen, im Ressourcenpool verbleiben. Die folgende Tabelle zeigt die Bindungen, die das benutzerdefinierte Entscheidungselement „Verbleibende VMs?“ benötigt.

**Tabelle 1-65.** Bindungen des benutzerdefinierten Entscheidungselements „Verbleibende VMs?“

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
numberOfVMs	EIN	Binden	<ul style="list-style-type: none"> <li>■ Quellparameter: numberOfVMs[attribute]</li> <li>■ Entscheidungsanweisung: Größer als</li> <li>■ Wert: 0.0</li> <li>■ Beschreibung: <b>Die Anzahl virtueller Maschinen im resourcePool</b></li> </ul>
vmCounter	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vmCounter</li> <li>■ Quellparameter: vmCounter[attribute]</li> <li>■ Typ: Zahl</li> <li>■ Beschreibung: <b>Die Anzahl der virtuellen Maschinen innerhalb eines Arrays</b></li> </ul>

Das benutzerdefinierte Entscheidungselement „Verbleibende VMs?“ führt die folgende Skriptfunktion durch.

```
//Checks if the workflow has reached the end of the array of VMs
if (vmCounter < numberOfVMs) {
    return true;
} else {
    return false;
}
```

#### Aktionselement getVMDisksModes

Das Aktionselement getVMDisksModes ruft die Modi der Festplatten ab, die in einer virtuellen Maschine ausgeführt werden. Die folgende Tabelle zeigt die Bindungen, die das Aktionselement getVMDisksModes benötigt.

**Tabelle 1-66.** Bindungen des Aktionselements getVMDisksModes

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
vm	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vm</li> <li>■ Quellparameter: vm[attribute]</li> <li>■ Typ: VC:VirtualMachine</li> <li>■ Beschreibung: <b>Die aktuelle virtuelle Maschine mit erstelltem Snapshot</b></li> </ul>
actionResult	AUS	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: actionResult</li> <li>■ Quellparameter: vmDisksModes[attribute]</li> <li>■ Typ: Array/Zeichenfolge</li> <li>■ Beschreibung: <b>Die aktuellen Festplattenmodi der virtuellen Maschine</b></li> </ul>
errorCode	Ausnahme	Erstellen	Lokaler Parameter: errorCode

**Snapshot erstellen? – benutzerdefiniertes Entscheidungselement**

Das benutzerdefinierte Entscheidungselement „Snapshot erstellen?“ bestimmt, ob Snapshots virtueller Maschinen erstellt werden sollen, und zwar unabhängig von den Festplattenmodi der virtuellen Maschinen. Die folgende Tabelle zeigt die Bindungen, die das benutzerdefinierte Entscheidungselement „Snapshot erstellen?“ benötigt.

**Tabelle 1-67.** Bindungen des Entscheidungselements „Snapshot erstellen?“

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
vmDisksMode	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vmDisksMode</li> <li>■ Quellparameter: vmDisksMode[attribute]</li> <li>■ Typ: Array/Zeichenfolge</li> <li>■ Beschreibung: <b>Die aktuellen Festplattenmodi der virtuellen Maschine</b></li> </ul>
vm	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vm</li> <li>■ Quellparameter: vm[attribute]</li> <li>■ Typ: VC:VirtualMachine</li> <li>■ Beschreibung: <b>Die aktuelle virtuelle Maschine mit erstelltem Snapshot</b></li> </ul>

Das benutzerdefinierte Entscheidungselement „Snapshot erstellen?“ führt die folgende Skriptfunktion durch.

```
//A snapshot cannot be taken if one of its disks is in independent mode
// (independent-persistent or independent-nonpersistent)
var containsIndependentDisks = false;
if (vmDisksModes!=null && vmDisksModes.length>0) {
    for (i in vmDisksModes) {
        if (vmDisksModes[i].charAt(0)=="i") {
            containsIndependentDisks = true;
        }
    }
} else {
    //if no disk found no need to try to snapshot the VM
    System.warn("Won't snapshot '"+vm.name+"', no disks found");
    return false;
}
if (containsIndependentDisks) {
    System.warn("Won't snapshot '"+vm.name+"', independent disk(s) found");
    return false;
} else {
    System.log("Snapshotting '"+vm.name+"'");
    return true;
}
```

### Workflowelement „Snapshot erstellen“

Das Workflow-Element „Snapshot erstellen“ erstellt Snapshots von virtuellen Maschinen. Die folgende Tabelle zeigt die Bindungen, die das Workflow-Element „Snapshot erstellen“ benötigt.

**Tabelle 1-68.** Bindungen des Workflowelements „Snapshot erstellen“

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
vm	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vm</li> <li>■ Quellparameter: vm[attribute]</li> <li>■ Typ: VC:VirtualMachine</li> <li>■ Beschreibung: <b>Eine aktive virtuelle Maschine, von der ein Snapshot erstellt werden soll.</b></li> </ul>
name	EIN	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: name</li> <li>■ Quellparameter: snapshotName[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Der Name dieses Snapshots. Der Name muss für diese virtuelle Maschine nicht eindeutig sein.</b></li> </ul>

**Tabelle 1-68.** Bindungen des Workflowelements „Snapshot erstellen“ (Fortsetzung)

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
description	EIN	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: description</li> <li>■ Quellparameter: snapshotDescription[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Eine Beschreibung für diesen Snapshot.</b></li> </ul>
memory	EIN	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: memory</li> <li>■ Quellparameter: snapshotMemory[attribute]</li> <li>■ Typ: Boolesch</li> <li>■ Wert: no</li> <li>■ Beschreibung: <b>Wenn TRUE, wird ein Abbild des internen Zustands der virtuellen Maschine (ein Speicherabbild) in den Snapshot eingefügt.</b></li> </ul>
quiesce	EIN	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: quiesce</li> <li>■ Quellparameter: snapshotQuiesce[attribute]</li> <li>■ Typ: Boolesch</li> <li>■ Wert: yes</li> <li>■ Beschreibung: <b>Wenn TRUE und wenn die virtuelle Maschine beim Erstellen des Snapshots eingeschaltet ist, werden die VMware Tools zum Stilllegen des Dateisystems in der virtuellen Maschine verwendet.</b></li> </ul>
snapshot	AUS	Erstellen	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: snapshot</li> <li>■ Quellparameter: NULL</li> <li>■ Typ: VC:VirtualMachineSnapshot</li> <li>■ Beschreibung: <b>Der Snapshot wurde erstellt.</b></li> </ul>
errorCode	Ausnahme	Erstellen	Lokaler Parameter: errorCode

**Skriptfähiges Aufgabenelement „VM-Snapshots“**

Das skriptfähige Aufgabenelement für VM-Snapshots fügt die Snapshots einem Array hinzu. Die folgende Tabelle zeigt die Bindungen, die das skriptfähige Aufgabenelement für VM-Snapshots benötigt.

**Tabelle 1-69.** Bindungen des skriptfähigen Aufgabenelements für VM-Snapshots

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
vm	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vm</li> <li>■ Quellparameter: vm[attribute]</li> <li>■ Typ: VC:VirtualMachine</li> <li>■ Beschreibung: <b>Eine aktive virtuelle Maschine, von der ein Snapshot erstellt werden soll.</b></li> </ul>
snapshotVmArray	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: snapshotVmArray</li> <li>■ Quellparameter: snapshotVmArray[attribute]</li> <li>■ Typ: Array/VC:VirtualMachine</li> <li>■ Beschreibung: <b>Das Array der virtuellen Maschinen, von denen Snapshots erstellt wurden</b></li> </ul>
snapshotVmArray	AUS	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: snapshotVmArray</li> <li>■ Quellparameter: snapshotVmArray[attribute]</li> <li>■ Typ: Array/VC:VirtualMachine</li> <li>■ Beschreibung: <b>Das Array der virtuellen Maschinen, von denen Snapshots erstellt wurden</b></li> </ul>

Das skriptfähige Aufgabenelement für VM-Snapshots führt die folgende Skriptfunktion durch.

```
//Writes the following event in the Orchestrator database
Server.log("Successfully took snapshot of the VM '"+vm.name);
//Inserts the VM snapshot in an array
snapshotVmArray.push(vm);
```

### Skriptfähiges Aufgabenelement für Inkrement

Das skriptfähige Aufgabenelement für Inkrement inkrementiert den Indikator, der die Anzahl virtueller Maschinen im Array zählt. Die folgende Tabelle zeigt die Bindungen, die das skriptfähige Aufgabenelement für Inkrement benötigt.



**Tabelle 1-70.** Bindungen des skriptfähigen Aufgabenelements für Inkrement

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
vmCounter	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vmCounter</li> <li>■ Quellparameter: vmCounter[attribute]</li> <li>■ Typ: Zahl</li> <li>■ Beschreibung: <b>Die Anzahl der virtuellen Maschinen innerhalb eines Arrays</b></li> </ul>
allVMs	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: allVMs</li> <li>■ Quellparameter: allVMs[attribute]</li> <li>■ Typ: Array/VC:VirtualMachine</li> <li>■ Beschreibung: <b>Die virtuellen Maschinen im Ressourcenpool</b></li> </ul>
vmCounter	AUS	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vmCounter</li> <li>■ Quellparameter: vmCounter[attribute]</li> <li>■ Typ: Zahl</li> <li>■ Beschreibung: <b>Die Anzahl der virtuellen Maschinen innerhalb eines Arrays</b></li> </ul>
vm	AUS	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vm</li> <li>■ Quellparameter: vm[attribute]</li> <li>■ Typ: VC:VirtualMachine</li> <li>■ Beschreibung: <b>Die aktuelle virtuelle Maschine mit erstelltem Snapshot</b></li> </ul>

Das skriptfähige Aufgabenelement für Inkrement führt die folgende Skriptfunktion durch.

```
//Increases the array VM counter
vmCounter++;
//Sets the next VM to be snapshot in the attribute vm
vm = allVMs[vmCounter];
```

#### **Skriptfähiges Aufgabenelement „Ausnahme protokollieren“**

Das skriptfähige Aufgabenelement „Ausnahme protokollieren“ behandelt Ausnahmen vom Workflow und den Aktionselementen. Die folgende Tabelle zeigt die Bindungen, die das skriptfähige Aufgabenelement „Ausnahme protokollieren“ benötigt.

**Tabelle 1-71.** Bindungen des Aufgabenelements „Ausnahme protokollieren“

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
vm	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: vm</li> <li>■ Quellparameter: vm[attribute]</li> <li>■ Typ: VC:VirtualMachine</li> <li>■ Beschreibung: <b>Die aktuelle virtuelle Maschine mit erstelltem Snapshot</b></li> </ul>
errorCode	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: errorCode</li> <li>■ Quellparameter: errorCode[attribute]</li> <li>■ Typ: Zeichenfolge</li> <li>■ Beschreibung: <b>Beim Erstellen eines Snapshots einer virtuellen Maschine wurde eine Ausnahme gefunden</b></li> </ul>

Das skriptfähige Aufgabenelement „Ausnahme protokollieren“ führt die folgende Skriptfunktion durch.

```
//Writes the following event in the Orchestrator database
Server.error("Couldn't snapshot the VM '"+vm.name+"', exception: '"+errorCode);
```

#### Skriptfähiges Aufgabenelement „Ausgabe festlegen“

Das skriptfähige Aufgabenelement „Ausgabe festlegen“ generiert den Ausgabeparameter des Workflows, in dem das Array virtueller Maschinen enthalten ist, von denen Snapshots erstellt wurden. Die folgende Tabelle zeigt die Bindungen, die das skriptfähige Aufgabenelement „Ausgabe festlegen“ benötigt.

**Tabelle 1-72.** Bindungen des Aufgabenelements „Ausgabe festlegen“

Parametername	Bindungstyp	Binden an vorhandenen Parameter oder Parameter erstellen?	Bindungswerte
snapshotVmArray	EIN	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: snapshotVmArray</li> <li>■ Quellparameter: snapshotVmArray[attribute]</li> <li>■ Typ: Array/VC:VirtualMachine</li> <li>■ Beschreibung: <b>Das Array der virtuellen Maschinen, von denen Snapshots erstellt wurden</b></li> </ul>
snapshotVmArrayOut	AUS	Binden	<ul style="list-style-type: none"> <li>■ Lokaler Parameter: snapshotVmArrayOut</li> <li>■ Quellparameter: snapshotVmArrayOut[out-parameter]</li> <li>■ Typ: Array/VC:VirtualMachine</li> <li>■ Beschreibung: <b>Das Array der virtuellen Maschinen, von denen Snapshots erstellt wurden</b></li> </ul>

Das skriptfähige Aufgabenelement „Ausgabe festlegen“ führt die folgende Skriptfunktion durch.

```
//Passes the value of the internal attribute to a workflow output parameter
snapshotVmArrayOut = snapshotVmArray;
```

## Setzen der Attributeigenschaften im Beispiel für einen komplexen Workflow

Die Attributeigenschaften werden auf der Registerkarte **Allgemein** im Workfloweditor gesetzt.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen komplexen Workflow“, auf Seite 170.
- „Erstellen des Schemas für das Beispiel eines komplexen Workflows“, auf Seite 172.
- „Definieren der Bindungen für das komplexe Workflowbeispiel“, auf Seite 176.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

### Vorgehensweise

- 1 Klicken Sie auf die Registerkarte **Allgemein**.
- 2 Wählen Sie das Kontrollkästchen für den Schreibschutz der folgenden Attribute aus, um sie zu schreibgeschützten Konstanten zu machen:
  - snapshotName
  - snapshotDescription
  - snapshotMemory
  - snapshotQuiesce

Sie haben definiert, welche Attribute des Workflows Konstanten oder Variablen sind.

### Weiter

Sie müssen die Workflow-Präsentation erstellen, die das Layout des Dialogfelds für die Eingabeparameter bestimmt, in dem die Benutzer die Werte von Eingabeparametern für den Workflow bestimmen, wenn Sie diesen ausführen.

## Erstellen des Layouts der Eingabeparameter für das Beispiel eines komplexen Workflows

Sie erstellen das Layout oder die Präsentation des Dialogfelds für die Eingabeparameter auf der Registerkarte **Präsentation** des Workfloweditors. Das Dialogfeld für die Eingabeparameter erscheint, wenn Benutzer einen Workflow ausführen. Hier geben die Benutzer die Eingabeparameter ein, mit denen der Workflow ausgeführt wird.

### Voraussetzungen

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen komplexen Workflow“, auf Seite 170.
- „Erstellen des Schemas für das Beispiel eines komplexen Workflows“, auf Seite 172.
- „Definieren der Parameter für das komplexe Workflowbeispiel“, auf Seite 176.
- „Definieren der Bindungen für das komplexe Workflowbeispiel“, auf Seite 176.
- „Setzen der Attributeigenschaften im Beispiel für einen komplexen Workflow“, auf Seite 187.

- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

### Vorgehensweise

- 1 Klicken Sie im Workfloweditor auf die Registerkarte **Präsentation**.  
Der Workflow „Snapshots von allen virtuellen Maschinen in einem Ressourcenpool erstellen“ hat nur einen Eingabeparameter. Die Erstellung der Präsentation ist daher einfach.
- 2 Klicken Sie mit der rechten Maustaste auf den Knoten **Präsentation** in der hierarchischen Liste für die Präsentation und wählen Sie **Anzeigegruppe erstellen**.
- 3 Löschen Sie das Element **Neuer Schritt**, das über dem Element **Neue Gruppe** angezeigt wird.
- 4 Doppelklicken Sie auf das Element **Neue Gruppe** und ändern Sie den Gruppennamen auf **Ressourcenpool**.
- 5 Geben Sie eine Beschreibung der Anzeigegruppe **Ressourcenpool** in das Textfeld **Beschreibung** auf der Registerkarte **Allgemein** unten auf der Registerkarte **Präsentation** ein.  
Beispiel:  
**Geben Sie den Namen eines Ressourcenpools ein, der die virtuellen Maschinen enthält, von denen der Snapshot erstellt wird.**
- 6 Klicken Sie auf den Parameter (VC:ResourcePool)resourcePool.
- 7 Klicken Sie auf die Registerkarte **Eigenschaften** für (VC:ResourcePool)resourcePool.
- 8 Klicken Sie mit der rechten Maustaste in die Registerkarte **Eigenschaften** und wählen Sie **Eigenschaft hinzufügen > Erforderliche Eingabe**.
- 9 Klicken Sie mit der rechten Maustaste in die Registerkarte **Eigenschaften** und wählen Sie **Eigenschaft hinzufügen > Wert auswählen als**.  
Wenn Sie diese Eigenschaft festlegen, bestimmen Sie, wie der Benutzer den Wert des (VC:ResourcePool)resourcePool-Eingabeparameters auswählt.
- 10 Ziehen Sie den (VC:ResourcePool)resourcePool-Parameter unter die Anzeigegruppe **Ressourcenpool**.

Sie haben das Layout des Dialogfelds erstellt, das erscheint, wenn Benutzer den Workflow ausführen.

### Weiter

Sie haben die Entwicklung des Beispiels für einen komplexen Workflow abgeschlossen. Sie können jetzt den Workflow validieren und ausführen.

## Validieren und Ausführen des Beispiels für einen komplexen Workflow

Nachdem Sie einen Workflow erstellt haben, können Sie ihn validieren, um eventuelle Fehler zu erkennen. Wenn der Workflow keine Fehler enthält, können Sie ihn ausführen.

### Voraussetzungen

Erstellen Sie einen Workflow, entwerfen Sie sein Schema, definieren Sie die Verknüpfungen und Bindungen, definieren Sie die Parametereigenschaften und erstellen Sie die Präsentation des Dialogfelds der Eingabeparameter.

Führen Sie die folgenden Aufgaben durch.

- „Erstellen des Beispiels für einen komplexen Workflow“, auf Seite 170.
- „Erstellen einer benutzerdefinierten Aktion für das Beispiel eines komplexen Workflows“, auf Seite 171.
- „Erstellen des Schemas für das Beispiel eines komplexen Workflows“, auf Seite 172.
- „Definieren der Parameter für das komplexe Workflowbeispiel“, auf Seite 176.

- „Definieren der Bindungen für das komplexe Workflowbeispiel“, auf Seite 176.
- „Setzen der Attributeigenschaften im Beispiel für einen komplexen Workflow“, auf Seite 187.
- „Erstellen des Layouts der Eingabeparameter für das Beispiel eines komplexen Workflows“, auf Seite 187.
- Öffnen Sie den Workflow für die Bearbeitung im Workflow-Editor.

### Vorgehensweise

- 1 Klicken Sie auf **Validierung** auf der Registerkarte **Schema** des Workfloweditors.  
Das Validierungstool erkennt Fehler in der Definition des Workflows.
- 2 Nachdem Sie Fehler behoben haben, klicken Sie auf **Speichern und schließen** unten im Workfloweditor.  
Kehren Sie zum Orchestrator-Client zurück.
- 3 Klicken Sie auf die Ansicht **Workflows**.
- 4 Wählen Sie in der hierarchischen Liste der Workflows **Workflowbeispiele > Snapshots von allen virtuellen Maschinen in einem Ressourcenpool erstellen**.
- 5 Klicken Sie mit der rechten Maustaste auf den Workflow **Snapshots von allen virtuellen Maschinen in einem Ressourcenpool erstellen** und wählen Sie **Workflow starten**.  
Das Dialogfeld der Eingabeparameter wird geöffnet und fordert Sie auf, einen Ressourcenpool anzugeben, der die virtuellen Maschinen enthält, deren Snapshot aufgenommen werden soll.
- 6 Klicken Sie auf **Übernehmen**, um den Workflow auszuführen.  
Ein Workflowtoken erscheint unter dem Workflow „Snapshots von allen virtuellen Maschinen in einem Ressourcenpool erstellen“.
- 7 Klicken Sie auf den Workflowtoken, um den Fortschritt des Workflows während des Ausführens zu beobachten.

Wenn der Workflow erfolgreich abläuft, erstellt der Workflow einen Snapshot aller virtuellen Maschinen im ausgewählten Ressourcenpool.

### Weiter

Sie können ein Dokument erstellen, in dem Informationen über den Workflow überprüft werden können. Siehe „Generieren einer Workflow-Dokumentation“, auf Seite 143.



# Skripterstellung

---

Orchestrator verwendet JavaScript, um Bausteine zu erstellen, aus denen Sie Aktionen, Workflowelemente und Richtlinien erstellen können, die auf die API der von Ihnen in Orchestrator integrierten Technologien zugreifen.

Orchestrator verwendet die Mozilla Rhino 1.7R4-JavaScript-Engine als Engine für die Skripterstellung. Die Skripterstellungseengine bietet die Prüfung von Variablentypen, die Verwaltung von Namespaces, die automatische Fertigstellung und die Verarbeitung von Ausnahmen.

Die Orchestrator-Workflowengine ermöglicht Ihnen die Verwendung einfacher Funktionen der JavaScript-Sprache wie as-if, loops, arrays und strings. Sie können Objekte in der Skripterstellung verwenden, die von der Orchestrator-API zur Verfügung gestellt werden, oder Objekte aus einer anderen API, die Sie in Orchestrator über ein Plug-In importieren und JavaScript-Objekten zuordnen. Informationen über Rhino finden Sie auf der Mozilla Rhino-Website.

Dieses Kapitel behandelt die folgenden Themen:

- [„Orchestrator-Elemente, für die eine Skripterstellung erforderlich ist“](#), auf Seite 192
- [„Einschränkungen der Mozilla Rhino-Implementierung in Orchestrator“](#), auf Seite 192
- [„Verwenden der Orchestrator-API zur Skripterstellung“](#), auf Seite 193
- [„Verwenden von XPath-Ausdrücken mit dem vCenter Server-Plug-In“](#), auf Seite 199
- [„Richtlinien für Ausnahmebehandlung“](#), auf Seite 200
- [„JavaScript-Beispiele für Orchestrator“](#), auf Seite 201

## Orchestrator-Elemente, für die eine Skripterstellung erforderlich ist

Nicht alle Orchestrator-Elemente erfordern, dass Sie Skripte schreiben. Um maximale Flexibilität für Ihre Anwendungen bereitzustellen, können Sie bestimmte Elemente anpassen, indem Sie JavaScript-Funktionen hinzufügen.

Sie können Skripte in den folgenden Orchestrator-Elementen hinzufügen.

<b>Aktionen</b>	Aktionen sind Funktionen, für die ein Skript erstellt wurde. Sie können die Skripterstellung für eine Aktion auf einen einzelnen Vorgang beschränken, um das Potenzial für die Wiederverwendung der Aktion durch andere Elemente, beispielsweise andere Workflows, zu maximieren. Als Alternative kann eine Aktion mehrere Vorgänge enthalten, um Workflows weniger komplex zu machen, wodurch aber die Möglichkeit zur Wiederverwendung der Aktion reduziert wird.
<b>Richtlinien</b>	Sie legen Richtlinien fest, indem Sie Skripte verwenden, die Auslöseereignisse überwachen. Wenn die Auslöseereignisse eintreffen, starten die Richtlinien Orchestrationsvorgänge, die Sie in Skripten definieren.
<b>Workflows</b>	Das Workflowelement für skriptfähige Aufgaben ermöglicht Ihnen das Schreiben eines in einem benutzerdefinierten Skript festgelegten Vorgangs oder einer Sequenz von Vorgängen, die Sie in den Workflows verwenden können. Sie definieren auch die boolesche Entscheidungsanweisung für benutzerdefinierte Entscheidungselemente in Skripten, die entweder <code>true</code> oder <code>false</code> zurückgeben.

## Einschränkungen der Mozilla Rhino-Implementierung in Orchestrator

Orchestrator verwendet die Mozilla Rhino 1.7R4-JavaScript-Engine. Die Implementierung von Rhino in Orchestrator birgt jedoch einige Einschränkungen.

Beim Schreiben von Skripten für Workflows müssen Sie die folgenden Einschränkungen der Mozilla Rhino-Implementierung in Orchestrator berücksichtigen.

- Wenn ein Workflow ausgeführt wird, handelt es sich bei den Objekten, die von einem Workflow-Element zum nächsten weitergegeben werden, nicht um JavaScript-Objekte. Es wird die Serialisierung eines Java-Objekts mit einem JavaScript-Bild von einem Element zum nächsten weitergegeben. Folglich können Sie nicht die ganze JavaScript-Sprache verwenden, sondern lediglich die im API-Explorer vorhandenen Klassen. Sie können Funktionsobjekte nicht von einem Workflowelement an das nächste weitergeben.
- Orchestrator führt den Code in skriptfähigen Aufgabenelementen in einem anderen Kontext als dem des Rhino-Root-Kontexts durch. Orchestrator umhüllt skriptfähige Aufgabenelemente und Aktionen transparent mit JavaScript-Funktionen und führt diese dann aus. Ein skriptfähiges Aufgabenelement, das `System.log(this);` enthält, zeigt das globale Objekt `this` nicht auf dieselbe Weise an wie eine standardmäßige Rhino-Implementierung.
- Sie können nur Aktionen aufrufen, die nicht serialisierbare Objekte aus dem Skript und nicht aus Workflows zurückgeben. Um Aktionen aufzurufen, die ein nicht serialisierbares Objekt zurückgeben, müssen Sie ein skriptfähiges Element schreiben, das die Aktion mithilfe der Methode `System.getModuleModuleName.action()` aufruft.
- Bei der Workflowvalidierung wird nicht überprüft, ob sich ein Workflow-Attributtyp vom Eingabetyp einer Aktion oder eines untergeordneten Workflows unterscheidet. Wenn Sie den Typ eines Workfloweingabeparameters zum Beispiel von `VIM3:VirtualMachine` in `VC:VirtualMachine` ändern, aber keine skriptfähigen Aufgaben oder Aktionen aktualisieren, die den ursprünglichen Eingabetyp verwenden, führt der Workflow die Validierung durch, wird aber nicht ausgeführt.



## Verwenden der Orchestrator-API zur Skripterstellung

Die Orchestrator-API stellt alle Objekte und Funktionen der Technologien bereit, auf die von Orchestrator über seine Plug-Ins als JavaScript-Objekten und -Methoden zugegriffen wird.

Sie können beispielsweise auf JavaScript-Implementierungen der vCenter Server-API über die Orchestrator-API zugreifen, um vCenter-Vorgänge in von Ihnen erstellten Skriptelementen zu verwenden. Sie können auch auf JavaScript-Implementierungen von Objekten aus allen anderen Plug-Ins zugreifen, die Sie im Orchestrator-Server installieren. Wenn Sie ein benutzerdefiniertes Plug-In für eine Drittanbieteranwendung erstellen, ordnen Sie die Objekte von seiner API den JavaScript-Objekten zu, die die Orchestrator-API dann bereitstellt.

### Vorgehensweise

- 1 [Zugriff auf das Skriptmodul über den Workfloweditor](#) auf Seite 194  
Das Skriptmodul von Orchestrator benutzt die Mozilla Rhino 1.7R4-JavaScript-Engine, um Sie beim Schreiben von Skripts für Skriptelemente in Workflows zu unterstützen. Sie können auf der Registerkarte **Skripterstellung** im Workfloweditor auf das Skriptmodul für Workflow-Skriptelemente zugreifen.
- 2 [Zugriff auf das Skriptmodul über den Aktions- oder Richtlinieneditor](#) auf Seite 194  
Das Skriptmodul von Orchestrator benutzt die Mozilla Rhino-JavaScript-Engine, um Sie beim Schreiben von Skripts für Aktionen und Richtlinien zu unterstützen. Der Zugriff auf das Skriptmodul für Aktionen und Richtlinien erfolgt über die Registerkarten für die **Skripterstellung** im Aktions- und im Richtlinieneditor.
- 3 [Zugriff auf den Explorer der Orchestrator-API](#) auf Seite 195  
Orchestrator verfügt über einen API-Explorer, mit dessen Hilfe Sie die Orchestrator-API durchsuchen und die Dokumentation für JavaScript-Objekte, die Sie in Skriptelementen verwenden können, sehen können.
- 4 [Verwenden Sie den Orchestrator-API-Explorer, um Objekte zu finden.](#) auf Seite 195  
Die Orchestrator-API macht die API aller Plug-In-Technologien einschließlich der gesamten vCenter Server-API verfügbar. Der Orchestrator-API-Explorer unterstützt Sie bei der Suche nach Objekten, die Sie benötigen, um sie Skripten hinzuzufügen.
- 5 [Schreiben von Skripts](#) auf Seite 196  
Das Orchestrator-Skriptmodul unterstützt Sie beim Erstellen von Skripts. Durch automatisches Einfügen von Funktionen und automatisches Vervollständigen von Skriptzeilen wird der Skripterstellungsprozess beschleunigt und das Risiko von Schreibfehlern in Skripten minimiert.
- 6 [Hinzufügen von Parametern zu Skripts](#) auf Seite 197  
Mithilfe des Orchestrator-Skriptmoduls können Sie verfügbare Parameter in Skripts importieren.
- 7 [Zugreifen auf das Orchestrator-Server-Dateisystem über JavaScript und Workflows](#) auf Seite 198  
Orchestrator schränkt den Zugriff auf das Orchestrator-Server-Dateisystem über JavaScript und Workflows auf bestimmte Verzeichnisse ein.
- 8 [Zugreifen auf Java-Klassen über JavaScript](#) auf Seite 199  
Standardmäßig schränkt Orchestrator den JavaScript-Zugriff auf einen begrenzten Satz von Java-Klassen. Wenn Sie JavaScript-Zugriff auf mehr Java-Klassen benötigen, müssen Sie eine Orchestrator-Systemeigenschaft festlegen, um diesen Zugriff zuzulassen.

- 9 [Zugreifen auf Betriebssystembefehle über JavaScript](#) auf Seite 199

Die Orchestrator-API stellt eine Skripterstellungsklasse, `Command`, bereit, die Befehle im Orchestrator-Server-Hostbetriebssystem durchführt. Um nicht autorisierten Zugriff auf den Orchestrator-Serverhost zu verhindern, haben Orchestrator-Anwendungen standardmäßig keine Berechtigungen zum Ausführen der Klasse `Command`.

## Zugriff auf das Skriptmodul über den Workfloweditor

Das Skriptmodul von Orchestrator benutzt die Mozilla Rhino 1.7R4-JavaScript-Engine, um Sie beim Schreiben von Skripts für Skriptelemente in Workflows zu unterstützen. Sie können auf der Registerkarte **Skripterstellung** im Workfloweditor auf das Skriptmodul für Workflow-Skriptelemente zugreifen.

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie in der Ansicht **Workflows** des Orchestrator-Clients mit der rechten Maustaste auf einen Workflow und wählen Sie **Bearbeiten**.
- 3 Klicken Sie im Workfloweditor auf die Registerkarte **Schema**.
- 4 Fügen Sie dem Workflowschema ein skriptfähiges Aufgabenelement oder ein benutzerdefiniertes Entscheidungselement hinzu.
- 5 Klicken Sie auf die Registerkarte **Skripterstellung** des skriptfähigen Elements.

Sie haben auf das Skriptmodul zugegriffen und die Skriptfunktionen von Workflowelementen definiert. Über die Registerkarte **Skripterstellung** können Sie in der API navigieren, Dokumentation zu den Objekten einsehen, Objekte suchen und JavaScript schreiben.

### Weiter

Durchsuchen Sie die Orchestrator-API mit dem API-Explorer.

## Zugriff auf das Skriptmodul über den Aktions- oder Richtlinieneditor

Das Skriptmodul von Orchestrator benutzt die Mozilla Rhino-JavaScript-Engine, um Sie beim Schreiben von Skripts für Aktionen und Richtlinien zu unterstützen. Der Zugriff auf das Skriptmodul für Aktionen und Richtlinien erfolgt über die Registerkarten für die **Skripterstellung** im Aktions- und im Richtlinieneditor.

### Vorgehensweise

- 1 Wählen Sie je nach dem Elementtyp, dessen Skript Sie bearbeiten möchten, eine Option aus dem Dropdown-Menü im Orchestrator-Client aus.

Option	Beschreibung
<b>Design</b>	Mithilfe dieser Option bearbeiten Sie das Skript eines Aktionselements.
<b>Ausführen</b>	Mithilfe dieser Option bearbeiten Sie das Skript einer Richtlinie.

- 2 Klicken Sie mit der rechten Maustaste in der Ansicht **Aktionen** oder **Richtlinien** auf eine Aktion oder Richtlinie und wählen Sie **Bearbeiten**.
- 3 Klicken Sie im Aktions- bzw. Richtlinieneditor auf die Registerkarte **Skripterstellung**.

Sie haben auf das Skriptmodul zugegriffen und die Skriptfunktionen von Aktions- oder Richtlinienelementen definiert. Über die Registerkarte **Skripterstellung** können Sie in der API navigieren, Dokumentation zu den Objekten einsehen, Objekte suchen und JavaScript schreiben.

### Weiter

Durchsuchen Sie die Orchestrator-API mit dem API-Explorer.

## Zugriff auf den Explorer der Orchestrator-API

Orchestrator verfügt über einen API-Explorer, mit dessen Hilfe Sie die Orchestrator-API durchsuchen und die Dokumentation für JavaScript-Objekte, die Sie in Skriptelementen verwenden können, sehen können.

Auf der Startseite der Dokumentation zu Orchestrator finden Sie eine Onlineversion der Skripterstellungs-API für das vCenter Server-Plug-In.

### Vorgehensweise

- 1 Melden Sie sich beim Orchestrator-Client an.
- 2 Wählen Sie **Tools > API-Explorer**.

Der API-Explorer wird angezeigt. Er kann zum Durchsuchen der Objekte und Funktionen der Orchestrator-API verwendet werden.

### Weiter

Sie können mit dem API-Explorer Skripts für Skriptelemente schreiben.

## Verwenden Sie den Orchestrator-API-Explorer, um Objekte zu finden.

Die Orchestrator-API macht die API aller Plug-In-Technologien einschließlich der gesamten vCenter Server-API verfügbar. Der Orchestrator-API-Explorer unterstützt Sie bei der Suche nach Objekten, die Sie benötigen, um sie Skripten hinzuzufügen.

### Voraussetzungen

Öffnen Sie den API-Explorer.

### Vorgehensweise

- 1 Geben Sie den Namen oder einen Teil eines Namens eines Objekts in das Textfeld **Suchen** des API-Explorers ein und klicken Sie auf **Suchen**.

Um Ihre Suche auf einen bestimmten Objekttyp zu beschränken, aktivieren oder deaktivieren Sie die Kontrollkästchen **Skripterstellungsklasse**, **Attribute und Methoden** und **Typen und Aufzählungen**.

- 2 Doppelklicken Sie auf das Element in der vorgeschlagenen Liste.

Das Objekt wird in der hierarchischen Liste auf der linken Seite hervorgehoben. Ein Dokumentationsbereich unter der hierarchischen Liste enthält Informationen über das Objekt.

### Weiter











Verwenden Sie die gefundenen Objekte in Skripten.

## JavaScript-Objekte im API-Explorer

Der Orchestrator-API-Explorer ermittelt und gruppiert die verschiedenen Arten von JavaScript-Objekten im Hierarchiebaum auf der linken Seite der Registerkarte **Skripterstellung** im Dialogfeld „API-Explorer“. Der API-Explorer nutzt Symbole zur Identifizierung der verschiedenen Objektarten.

Die folgende Tabelle beschreibt die Objekte der Orchestrator-API und präsentiert ihre Symbole.

**Tabelle 2-1.** JavaScript-Objekte in der Orchestrator-API

Objekt	Symbol in der hierarchischen Liste	Beschreibung
Typ		Typen
Funktionsgruppe		Interner Typ, der eine Gruppe statischer Methoden enthält
Primitive		Primitive Typen
Objekt		Standard-Orchestrator-Skripterstellungsobjekte
Attribut		JavaScript-Attribute
Methode		JavaScript-Methoden
Konstruktor		JavaScript-Konstrukturen
Enumeration		JavaScript-Aufzählungen
Zeichenfolgensatz		Zeichenfolgensatz, Standardwerte
Modul		Eine Sammlung von Aktionen
Plug-In	Bild, das das Plug-In definiert	Die APIs, die Plug-Ins für den Orchestrator bereitstellen

## Schreiben von Skripts

Das Orchestrator-Skriptmodul unterstützt Sie beim Erstellen von Skripten. Durch automatisches Einfügen von Funktionen und automatisches Vervollständigen von Skriptzeilen wird der Skripterstellungsprozess beschleunigt und das Risiko von Schreibfehlern in Skripten minimiert.

### Voraussetzungen

Öffnen Sie ein Skriptelement zur Bearbeitung und klicken Sie auf die Registerkarte **Skripterstellung**.

### Vorgehensweise

- 1 Navigieren Sie durch die hierarchische Liste von Objekten auf der linken Seite der Registerkarte **Skripterstellung** oder wählen Sie mithilfe der API-Explorer-Suchfunktion einen Typ, eine Klasse oder Methode aus, der/die dem Skript hinzugefügt werden soll.
- 2 Klicken Sie mit der rechten Maustaste auf den Typ, die Klasse oder Methode und wählen Sie **Kopieren**.  
Wenn Sie mit dem Skriptmodul das von Ihnen ausgewählte Element nicht kopieren können, ist dieses Objekt im Rahmen des Skripts nicht möglich.
- 3 Klicken Sie mit der rechten Maustaste in den Skriptblock und fügen Sie das von Ihnen kopierte Element an der entsprechenden Stelle im Skript ein.

Das Skriptmodul gibt das Element in das Skript ein, und zwar vollständig mit seinem Konstruktor und einem Instanznamen.

Beispiel: Wenn Sie das Objekt `Date` kopiert haben, wird der folgende Code vom Skriptmodul in das Skript eingefügt.

```
var myDate = new Date();
```

- 4 Fügen Sie dem Skript eine Methode durch Kopieren und Einfügen hinzu.

Das Skriptmodul vervollständigt den Methodenaufruf und fügt die erforderlichen Attribute hinzu.

Beispiel: Wenn Sie die Methode `cloneVM()` vom Modul `com.vmware.library.vc.vm` kopiert haben, wird der folgende Code vom Skriptmodul in das Skript eingefügt.

```
System.getModule("com.vmware.library.vc.vm").cloneVM(vm, folder, name, spec)
```

Das Skriptmodul hebt jene Parameter hervor, die Sie bereits im Element definiert haben. Sämtliche nicht definierten Parameter bleiben nicht hervorgehoben.

- 5 Platzieren Sie den Cursor am Ende eines von Ihnen in das Skript eingefügten Elements und drücken Sie Strg+Leertaste, um aus einer Kontextliste mögliche Methoden und Attribute auszuwählen, die das Objekt aufrufen kann.

---

**HINWEIS** Die Auto-Vervollständigungsfunktion ist derzeit experimentell.

---

Sie haben dem Skript ein Objekt und Funktionen hinzugefügt.

### Weiter

Fügen Sie dem Skript Parameter hinzu.

## Farbcodierung von Skriptschlüsselwörtern

Wenn Sie Skripte auf der Registerkarte **Skripterstellung** eines Workflowskriptelements hinzufügen, werden bestimmte Typen von Schlüsselwörtern in verschiedenen Farben angezeigt, um die Lesbarkeit des Codes zu verbessern.

Sofern nicht anderweitig festgelegt, wird Skriptinhalt als Schrift in Standardschwarz angezeigt.

**Tabelle 2-2.** Farbcodierung von Skriptschlüsselwörtern

Schlüsselworttyp	Textfarbe auf der Registerkarte „Skripterstellung“
JavaScript-Standardschlüsselwörter z. B. <code>if</code> , <code>else</code> , <code>for</code> und <code>new</code>	Fett schwarz
Variablendeklarationen, nämlich <code>var</code>	Grün
Modifizierer in Schleifen, z. B. <code>in</code>	Rot
Null-Variablenwerte	Violett
Nicht-Null-Variablenwerte	Grün
Codekommentare	Kursiv grau
Orchestrator Plug-In-Objekttypen, z. B. <code>VC:VirtualMachine</code> oder <code>VC:Host</code>	Grün
Ausgabertext	Grün
Workflowattribute	Pink
Workfloweingaben	Pink
Workflowausgaben	Pink

## Hinzufügen von Parametern zu Skripten

Mithilfe des Orchestrator-Skriptmoduls können Sie verfügbare Parameter in Skripten importieren.

Wenn Sie bereits Parameter für das Element, das Sie gerade bearbeiten, definiert haben, werden sie als Links in der Symbolleiste der Registerkarte **Skripterstellung** angezeigt.

### Voraussetzungen

Ein Skriptelement ist zur Bearbeitung geöffnet und die Registerkarte **Skripterstellung** wird angezeigt.

### Vorgehensweise

- 1 Bewegen Sie den Cursor in einem Skript im Skriptblock der Registerkarte **Skripterstellung** an die entsprechende Position.
- 2 Klicken Sie in der Symbolleiste der Registerkarte **Skripterstellung** auf den Parameterlink.  
Orchestrator fügt den Parameter an der Cursorposition ein.
- 3 Fügen Sie einen Parameter mit Nullwert in das Skript ein.  
Wenn Sie Nullwerte an einfache Typen wie Ganzzahlen, Boolesche Werte und Zeichenfolgen übergeben, legt die Orchestrator-Skript-API automatisch den Standardwert für dieses Argument fest.

Sie haben dem Skript Parameter hinzugefügt.

### Weiter

Fügen Sie den Zugriff auf Java-Klassen in Skripten hinzu.

## Zugreifen auf das Orchestrator-Server-Dateisystem über JavaScript und Workflows

Orchestrator schränkt den Zugriff auf das Orchestrator-Server-Dateisystem über JavaScript und Workflows auf bestimmte Verzeichnisse ein.

JavaScript-Funktionen und Workflows haben nur Lese-, Schreib und Ausführungsberechtigung im permanenten Verzeichnis `c:\orchestrator`.

Der Orchestrator-Administrator kann die Ordner ändern, für die JavaScript-Funktionen und Workflows Lese-, Schreib- und Ausführungsberechtigungen haben, indem er eine Systemeigenschaft festlegt. Weitere Informationen zum Einrichten von Systemeigenschaften finden Sie unter *Installieren und Konfigurieren VMware vRealize Orchestrator*.

JavaScript-Funktionen und Workflows haben auch Lese-, Schreib und Ausführungsberechtigung im temporären E-/A-Standardordner des Serversystems. Das Schreiben in den temporären E-/A-Standardordner ist die einzige unabhängige, garantierte und konfigurationsunabhängige Möglichkeit, auf das Dateisystem mit vollen Berechtigungen zuzugreifen. Dateien, die Sie in den temporären E-/A-Standardordner schreiben, gehen jedoch verloren, wenn Sie den Server neu starten.

Sie können den temporären E-/A-Standardordner abrufen, indem Sie die Methode `System.getTempDirectory` in JavaScript-Funktionen aufrufen.

### Zugriff auf das Serverdateisystem mithilfe der Methode „`System.getTempDirectory`“

Als Alternative zum Schreiben in die Ordner auf dem Orchestrator-Serversystem, in dem der Administrator die entsprechenden Berechtigungen festgelegt hat, können Sie in den temporären E-/A-Standardordner schreiben.

Orchestrator hat standardmäßig volle Lese-, Schreib- und Ausführungsrechte im temporären E-/A-Standardordner. Sie können den temporären E-/A-Standardordner abrufen, indem Sie die Methode `System.getTempDirectory` in den JavaScript-Funktionen verwenden.

### Vorgehensweise

- ◆ Beziehen Sie die folgende Codezeile in die JavaScript-Funktionen ein, um auf den Ordner `java.io.tmpdir` zuzugreifen.

```
var tempDir = System.getTempDirectory()
```

## Zugreifen auf Java-Klassen über JavaScript

Standardmäßig schränkt Orchestrator den JavaScript-Zugriff auf einen begrenzten Satz von Java-Klassen. Wenn Sie JavaScript-Zugriff auf mehr Java-Klassen benötigen, müssen Sie eine Orchestrator-Systemeigenschaft festlegen, um diesen Zugriff zuzulassen.

Standardmäßig kann die Orchestrator JavaScript-Engine nur auf die Klassen im Paket `java.util.*` zugreifen.

Der Orchestrator-Administrator kann den Zugriff auf andere Java-Klassen von den JavaScript-Funktionen aus zulassen, indem er eine Systemeigenschaft festlegt. Weitere Informationen zum Einrichten von Systemeigenschaften finden Sie unter *Installieren und Konfigurieren VMware vRealize Orchestrator*.

## Zugreifen auf Betriebssystembefehle über JavaScript

Die Orchestrator-API stellt eine Skripterstellungsklasse, `Command`, bereit, die Befehle im Orchestrator-Server-Hostbetriebssystem durchführt. Um nicht autorisierten Zugriff auf den Orchestrator-Serverhost zu verhindern, haben Orchestrator-Anwendungen standardmäßig keine Berechtigungen zum Ausführen der Klasse `Command`.

Der Orchestrator-Administrator kann den Zugriff auf die Skripterstellungsklasse `Command` zulassen, indem er die Systemeigenschaft `com.vmware.js.allow-local-process=true` festlegt.

Weitere Informationen zum Einrichten von Systemeigenschaften finden Sie unter *Installieren und Konfigurieren von VMware vCenter Orchestrator*.

Weitere Informationen zum Einrichten von Systemeigenschaften finden Sie unter *Installieren und Konfigurieren von VMware vCenter Orchestrator*.

## Verwenden von XPath-Ausdrücken mit dem vCenter Server -Plug-In

Mithilfe der Finder-Methoden im vCenter Server-Plug-In können Sie Abfragen für vCenter Server-Bestandslistenobjekte durchführen. Sie können mithilfe von XPath-Ausdrücken Suchparameter definieren.

Das vCenter Server-Plug-In enthält eine Reihe von Finder-Methoden für Objekte, beispielsweise `getAllDatastores()`, `getAllResourcePools()`, `findAllForType()`. Mithilfe dieser Methoden können Sie auf die Bestandslisten der mit Ihrem Orchestrator-Server verbundenen vCenter Server-Instanzen zugreifen und anhand von ID, Name oder anderen Eigenschaften nach Objekten suchen.

Aus Gründen der Leistung geben die Finder-Methoden keine Eigenschaften für die abgefragten Objekte zurück, es sei denn, Sie geben einen Satz an Eigenschaften in der Abfrage an.

Auf der Startseite der Dokumentation zu Orchestrator finden Sie eine Onlineversion der Skriptstellungs-API für das vCenter Server-Plug-In.

---

**WICHTIG** Die auf XPath-Ausdrücken basierenden Abfragen können die Leistung von Orchestrator beeinträchtigen, da die Finder-Methode auf vCenter Server-Seite sämtliche Objekte eines gegebenen Typs zurückgibt und die Abfragefilter auf vCenter Server-Plug-In-Seite angewendet werden.

---

## Verwenden von XPath-Ausdrücken mit dem vCenter Server -Plug-In

Wenn Sie eine Finder-Methode aufrufen, können Sie Ausdrücke auf Basis der XPath-Abfragesprache verwenden. Die Suche gibt alle Bestandslistenobjekte zurück, die mit den XPath-Ausdrücken übereinstimmen. Wenn Sie Eigenschaften abfragen möchten, können Sie diese in Form eines Zeichenfolgen-Arrays in das Suchskript aufnehmen.

Das folgende JavaScript-Beispiel nutzt das VcPlugin-Skriptobjekt und einen XPath-Ausdruck, um die Namen aller Datenspeicherobjekte zurückzugeben, die zu den verwalteten vCenter Server-Objekten gehören und deren Namen die Zeichenfolge **ds** enthalten.

```
var datastores = VcPlugin.getAllDatastores(null, "xpath:name[contains(.,'ds')]");
for each (datastore in datastores){
    System.log(datastore.name);
}
```

Derselbe XPath-Ausdruck kann mithilfe des Server-Skriptobjekts und der findAllForType-Finder-Methode aufgerufen werden.

```
var datastores = Server.findAllForType("VC:Datastore", "xpath:name[contains(.,'ds')]");
for each (datastore in datastores){
    System.log(datastore.name);
}
```

Das folgende Skriptbeispiel gibt die Namen aller Hostsystemobjekte zurück, deren ID mit der Ziffer **1** beginnt.

```
var hosts = VcPlugin.getAllHostSystems(null, "xpath:id[starts-with(.,'1')]");
for each (host in hosts){
    System.log(host.name);
}
```

Das folgende Skriptbeispiel gibt die Namen und IDs aller Datencenterobjekte zurück, deren Namen die Zeichenfolge **DC** in Groß- oder Kleinbuchstaben enthalten. Das Skript ruft außerdem die **tag**-Eigenschaft ab.

```
var datacenters = VcPlugin.getAllDatacenters(['tag'], "xpath:name[contains(translate(., 'DC', 'dc'), 'dc')]");
for each (datacenter in datacenters){
    System.log(datacenter.name + " " + datacenter.id);
}
```

## Richtlinien für Ausnahmebehandlung

Die Orchestrator-Implementierung der Mozilla Rhino-JavaScript-Engine unterstützt Ausnahmebehandlung, damit Sie Fehler verarbeiten können. Beim Schreiben von Ausnahmehandler in Skripten müssen Sie die folgenden Richtlinien beachten.

- Verwenden Sie die folgenden European Computer Manufacturers Association-Fehlertypen. Verwenden Sie Error als generische Ausnahme, die Plug-In-Funktionen zurückgeben, und die folgenden spezifischen Fehlertypen.
  - TypeError
  - RangeError
  - EvalError
  - ReferenceError
  - URIError
  - SyntaxError



Das folgende Beispiel zeigt eine `URIError`-Definition.

```
try {
    ...
    throw new URIError("VirtualMachine with ID 'vm-0056'
                        not found on 'vcenter-test-1'");
    ...
} catch ( e if e instanceof URIError ) {

}
```

- Alle Ausnahmen, die von Skripts nicht erfasst werden, müssen einfache Zeichenfolgeobjekten der Form `<type>:SPACE<human readable message>` sein, wie das folgende Beispiel zeigt.

```
throw "ValidationError: The input parameter 'myParam' of type 'string' is too short."
```

- Schreiben Sie von Menschen zu lesende Meldungen so klar wie deutlich.
- Die Ausnahmetypprüfung einfacher Zeichenfolgen muss das folgende Muster verwenden.

```
try {
    throw "VMwareNoSpaceLeftOnDatastore: Datastore 'myDatastore' has no space left" ;
} catch ( e if (typeof(e)=="string" && e.indexOf("VMwareNoSpaceLeftOnDatastore:") == 0) ) {
    System.log("No space left on device");
    // Do something useful here
}
```

- Die Ausnahmetypprüfung einfacher Zeichenfolgen muss das folgende Muster in Skriptelementen in Workflows verwenden.

```
if (typeof(errorCode)=="string"
    && errorCode.indexOf("VMwareNoSpaceLeftOnDatastore:")
    == 0) {
    // Do something useful here
}
```

## JavaScript-Beispiele für Orchestrator

Sie können die JavaScript-Beispiele für Orchestrator ausschneiden, einfügen und adaptieren, um Ihnen das Schreiben von JavaScripts für allgemeine Orchestrierungsaufgaben zu erleichtern.

- [Allgemeine Skriptbeispiele](#) auf Seite 202  
Workflow-Skriptelemente, -aktionen und -richtlinien erfordern eine grundlegende Skripts für häufige Aufgaben. Sie können diese Beispiele in Ihre(n) Skriptelementen ausschneiden, einfügen und anpassen.
- [E-Mail-Skriptbeispiele](#) auf Seite 204  
Workflow-Skriptelemente können die Skripterstellung häufiger E-Mail-bezogener Tasks umfassen. Sie können diese Beispiele ausschneiden und in Ihre Skriptelemente einfügen und anpassen.
- [Dateisystem-Skriptbeispiele](#) auf Seite 205  
Workflow-Skriptelemente, Aktionen und Richtlinien erfordern die Skripterstellung für häufige Dateisystemaufgaben. Sie können diese Beispiele ausschneiden und in Ihre Skriptelemente einfügen und anpassen.
- [LDAP-Skriptstellungsbeispiele](#) auf Seite 205  
Workflow-Skriptelemente, Aktionen und Richtlinien erfordern die Skripterstellung für häufige LDAP-Aufgaben. Sie können diese Beispiele in Ihre(n) Skriptelementen ausschneiden, einfügen und anpassen.

- [Protokollierung-Skripterstellungbeispiele](#) auf Seite 206

Workflow-Skriptelemente, Aktionen und Richtlinien erfordern die Skripterstellung für häufige Protokollierungsaufgaben. Sie können diese Beispiele in Ihre(n) Skriptelementen ausschneiden, einfügen und anpassen.

- [Netzwerk-Skripterstellungbeispiele](#) auf Seite 206

Workflow-Skriptelemente, Aktionen und Richtlinien erfordern die Skripterstellung für häufige Netzweraufgaben. Sie können diese Beispiele in Ihre(n) Skriptelementen ausschneiden, einfügen und anpassen.

- [Beispiele für Workflow-Skripterstellung](#) auf Seite 206

Workflow-Skriptelemente, Aktionen und Richtlinien erfordern die Skripterstellung für häufige Workflowaufgaben. Sie können diese Beispiele in Ihre(n) Skriptelementen ausschneiden, einfügen und anpassen.

## Allgemeine Skriptbeispiele

Workflow-Skriptelemente, -aktionen und -richtlinien erfordern eine grundlegende Skripts für häufige Aufgaben. Sie können diese Beispiele in Ihre(n) Skriptelementen ausschneiden, einfügen und anpassen.

### Zugriff auf XML-Dokumente

Anhand des folgenden JavaScript-Beispiels können Sie mit der Implementierung von ECMAScript for XML (E4X) in der JavaScript-API von Orchestrator auf JavaScript-XML-Dokumente zugreifen.

---

**HINWEIS** Zusätzlich zur Implementierung von E4X in der JavaScript-API stellt Orchestrator auch eine XML-Implementierung für ein Document Object Model (DOM) im XML-Plug-In bereit. Information zum XML-Plug-In und seinen Beispielworkflows finden Sie unter *Verwenden von vRealize Orchestrator-Plug-Ins*.

---

```
var people = <people>
    <person id="1">
        <name>Moe</name>
    </person>
    <person id="2">
        <name>Larry</name>
    </person>
</people>;

System.log("'people' = " + people);

// built-in XML type
System.log("'people' is of type : " + typeof(people));

// list-like interface System.log("which contains a list of " +
people.person.length() + " persons");
System.log("whose first element is : " + people.person[0]);

// attribute 'id' is mapped to field '@id'
people.person[0].@id='47';
// change Moe's id to 47
// also supports search by constraints
System.log("Moe's id is now : " + people.person.(name=='Moe').@id);

// suppress Moe from the list
delete people.person[0];
System.log("Moe is now removed.");
```

```
// new (sub-)document can be built from a string
people.person[1] = new XML("<person id=\"3\"><name>James</name></person>");
System.log("Added James to the list, which is now :");
for each(var person in people..person)

for each(var person in people..person){
    System.log("- " + person.name + " (id=" + person.@id + ")");
}
```

## Festlegen und Abrufen von Eigenschaften über eine Hashtabelle

Im folgenden JavaScript-Beispiel werden Eigenschaften in einer Hashtabelle festgelegt und die Eigenschaften von der Hashtabelle abgerufen. Im folgenden Beispiel handelt es sich beim Schlüssel immer um einen Zeichenfolge, und der Wert ist ein Objekt, eine Zahl, ein Boolescher Wert oder eine Zeichenfolge.

```
var table = new Properties() ;
table.put("myKey",new Date()) ;
// get the object back
var myDate= table.get("myKey") ;
System.log("Date is : "+myDate) ;
```

## Ersetzen von Zeichenfolge-Inhalten

Im folgenden JavaScript-Beispiel wird der Inhalt einer Zeichenfolge durch neuen Inhalt ersetzt.

```
var str1 = "'hello'" ;
var reg = new RegExp("'", "g");
var str2 = str1.replace(reg,"\\'") ;
System.log(""+str2) ; // result : \'hello\'
```

## Vergleichen von Typen

Im folgenden JavaScript-Beispiel wird geprüft, ob ein Objekt mit einem angegebenen Objekttyp übereinstimmt.

```
var path = 'myurl/test';
if(typeof(path, string)){
    throw("string");
} else {
    throw("other");
}
```

## Ausführen eines Befehls auf dem Orchestrator-Server

Im folgenden JavaScript-Beispiel können Sie eine Befehlszeile auf dem Orchestrator-Server ausführen. Verwenden Sie dieselben Anmeldedaten wie jene zum Starten des Servers.

---

**HINWEIS** Der Zugriff auf das Dateisystem ist standardmäßig eingeschränkt.

---

```
var cmd = new Command("ls -al") ;
cmd.execute(true) ;
System.log(cmd.output) ;
```

## E-Mail-Skriptbeispiele

Workflow-Skriptelemente können die Skripterstellung häufiger E-Mail-bezogener Tasks umfassen. Sie können diese Beispiele ausschneiden und in Ihre Skriptelemente einfügen und anpassen.

Wenn Sie einen Mail-Workflow ausführen, verwendet er die Standard-Mail-Serverkonfiguration, die Sie im Workflow „Mail konfigurieren“ festlegen. Sie können die Standardwerte durch Verwenden von Eingabeparametern oder Definieren von benutzerdefinierten Werten in Workflow-Skriptelementen außer Kraft setzen.

### E-Mail-Adresse abrufen

Das folgende JavaScript-Beispiel ruft die E-Mail-Adresse des aktuellen Besitzers eines ausgeführten Skripts ab.

```
var emailAddress = Server.getRunningUser().emailAddress ;
```

### E-Mail senden

Das folgende JavaScript-Beispiel sendet eine E-Mail an den definierten Empfänger über einen SMTP-Server, mit dem definierten Inhalt.

```
var message = new EmailMessage() ;
message.smtpHost = "smtpHost" ;
message.subject= "my subject" ;
message.toAddress = "receiver@vmware.com" ;
message.fromAddress = "sender@vmware.com" ;
message.addMimePart("This is a simple message","text/html") ;
message.sendMessage() ;
```

### E-Mail-Nachrichten abrufen

Das folgende JavaScript-Beispiel ruft die Nachrichten eines E-Mail-Kontos über die durch die Klasse Mail-Client bereitgestellte Skript-API ab, ohne sie zu löschen.

```
var myMailClient = new MailClient();

myMailClient.setProtocol(mailProtocol);
if(useSSL){
    myMailClient.enableSSL();
}

myMailClient.connect( mailServer, mailPort, mailUsername, mailPassword);
System.log("Successfully login!");

try {
    myMailClient.openFolder("Inbox");

    var messages = myMailClient.getMessages();
    System.log("Reading messages...!");
    if ( messages != null && messages.length > 0 ) {
        System.log( "You have " + messages.length + " email(s) in your inbox" );
        for (i = 0; i < messages.length; i++) {
            System.log("");
            System.log("-----MSG-----");
            System.log("Headers: ");
            var headerProp = messages[i].getHeaders();
            for each(key in headerProp.keys){
```

```

        System.log(key+": "+headerProp.get(key));
    }
    System.log("");

    System.log( "Message["+ i +"] with from: " + messages[i].from + " to: " + messages[i].to);
    System.log( "Message["+ i +"] with subject: " + messages[i].subject);
    var content = messages[i].getContent();
    System.log("Msg content as string: " + content);
}
} else {
    System.warn( "No messages found" );
}
} finally {
    myMailClient.closeFolder();
    myMailClient.close();
}
}

```

## Dateisystem-Skriptbeispiele

Workflow-Skriptelemente, Aktionen und Richtlinien erfordern die Skripterstellung für häufige Dateisystemaufgaben. Sie können diese Beispiele ausschneiden und in Ihre Skriptelemente einfügen und anpassen.

### Hinzufügen von Inhalt zu einer einfachen Textdatei

Das folgende JavaScript-Beispiel fügt Inhalte zu einer Textdatei hinzu.

```

var tempDir = System.getTempDirectory() ;
var fileWriter = new FileWriter(tempDir + "/readme.txt") ;
fileWriter.open() ;
fileWriter.writeLine("File written at : "+new Date()) ;
fileWriter.writeLine("Another line") ;
fileWriter.close() ;

```

### Abrufen von Dateiinhalten

Das folgende JavaScript-Beispiel ruft die Inhalte einer Datei von der Orchestrator-Server-Host-Maschine ab.

```

var tempDir = System.getTempDirectory() ;
var fileReader = new FileReader(tempDir + "/readme.txt") ;
fileReader.open() ;
var fileContentAsString = fileReader.readAll();
fileReader.close() ;

```

## LDAP-Skriptstellungsbeispiele

Workflow-Skriptelemente, Aktionen und Richtlinien erfordern die Skripterstellung für häufige LDAP-Aufgaben. Sie können diese Beispiele in Ihre(n) Skriptelementen ausschneiden, einfügen und anpassen.

### Konvertieren von LDAP-Objekten in Active Directory-Objekte

Im folgenden JavaScript-Beispiel werden LDAP-Gruppenelemente in Active Directory-Benutzergruppenobjekte konvertiert und umgekehrt.

```

var ldapGroup ;
// convert from ldap element to Microsoft:UserGroup object
var adGroup = ActiveDirectory.search("UserGroup",ldapGroup.commonName) ;
// convert back to LdapGroup element
var ldapElement = Server.getLdapElement(adGroup.distinguishedName) ;

```

## Protokollierung-Skripterstellungsbeispiele

Workflow-Skriptelemente, Aktionen und Richtlinien erfordern die Skripterstellung für häufige Protokollierungsaufgaben. Sie können diese Beispiele in Ihre(n) Skriptelementen ausschneiden, einfügen und anpassen.

### Dauerhafte Protokollierung

Mit dem folgenden JavaScript-Beispiel werden dauerhafte Protokolleinträge erstellt.

```
Server.log("This is a persistant message", "enter a long description here");
Server.warn("This is a persistant warning", "enter a long description here");
Server.error("This is a persistant error", "enter a long description here");
```

### Nicht dauerhafte Protokollierung

Mit dem folgenden JavaScript-Beispiel werden nicht dauerhafte Protokolleinträge erstellt.

```
System.log("This is a non-persistant log message");
System.warn("This is a non-persistant log warning");
System.error("This is a non-persistant log error");
```

## Netzwerk-Skripterstellungsbeispiele

Workflow-Skriptelemente, Aktionen und Richtlinien erfordern die Skripterstellung für häufige Netzwerkaufgaben. Sie können diese Beispiele in Ihre(n) Skriptelementen ausschneiden, einfügen und anpassen.

### Abrufen von Text aus einer URL

Das folgende JavaScript-Beispiel greift auf eine URL zu, ruft Text ab und wandelt ihn in eine Zeichenfolge um.

```
var url = new URL("http://www.vmware.com") ;
var htmlContentAsString = url.getContent() ;
```

## Beispiele für Workflow-Skripterstellung

Workflow-Skriptelemente, Aktionen und Richtlinien erfordern die Skripterstellung für häufige Workflowaufgaben. Sie können diese Beispiele in Ihre(n) Skriptelementen ausschneiden, einfügen und anpassen.

### Zurückgeben aller Workflow, die vom aktuellen Benutzer ausgeführt werden

Das folgende JavaScript-Beispiel ruft alle Workflowausführungen von diesem Server ab und prüft, ob sie dem aktuellen Benutzer gehören.

```
var allTokens = Server.findAllForType('WorkflowToken');
var currentUser = Server.getCredential().username;
var res = [];
for(var i = 0; i<res.length; i++){
    if(allTokens[i].runningUserName == currentUser){
        res.push(allTokens[i]);
    }
}
return res;
```

## Zugriff auf den aktuellen Workflowtoken

Sie können auf den aktuellen Workflowtoken mithilfe der Variablen `workflow` zugreifen. Sie ist ein Objekt des Typs `WorkflowToken` und bietet den Zugriff auf den aktuellen Workflowdurchgang. Das folgende JavaScript-Beispiel ruft den Bezeichner des Workflowtokens und sein Startdatum ab.

```
System.log("Current workflow run ID: " + workflow.id);
System.log("Current workflow run start date: "+workflow.startDate);
```

## Planen eines Workflows

Das folgende JavaScript-Beispiel startet einen Workflow mit einem gegebenen Satz von Eigenschaften und plant dann, ihn eine Stunde später zu starten.

```
var workflowToLaunch = myWorkflow ;
// create parameters
var workflowParameters = new Properties() ;
workflowParameters.put("name","John Doe") ;
// change the task name
workflowParameters.put("__taskName","Workflow for John Doe") ;

// create scheduling date one hour in the future
var workflowScheduleDate = new Date() ;
var time = workflowScheduleDate.getTime() + (60*60*1000) ;
workflowScheduleDate.setTime(time) ; var scheduledTask =
workflowToLaunch.schedule(workflowParameters,workflowScheduleDate);
```

## Ausführen eines Workflows mit einer Auswahl von Objekten in einer Schleife

Das folgende JavaScript-Beispiel übernimmt das Array von virtuellen Maschinen und führt einen Workflow für jede von ihnen in einer For-Schleife aus. `VMs` und `workflowToRun` sind Workfloweingaben.

```
var len=VMs.length;
for (var i=0; i < len; i++ )
{
    var VM = VMs[i];
    //var workflowToLaunch = Server.getWorkflowWithId("workflowId");
    var workflowToLaunch = workflowToRun;
    if (workflowToLaunch == null) {
        throw "Workflow not found";
    }
    var workflowParameters = new Properties();
    workflowParameters.put("vm",VM);
    var wfToken = workflowToLaunch.execute(workflowParameters);
    System.log ("Ran workflow on " +VM.name);
}
```





# Entwicklungsaktionen

---

Orchestrator stellt Bibliotheken vordefinierter Aktionen zur Verfügung. Aktionen stellen individuelle Funktionen dar, die Sie als Bausteine in Workflows und Skripts verwenden.

Aktionen sind JavaScript-Funktionen. Sie können mehrere Eingabeparameter besitzen und haben einen einzelnen Rückgabewert. Sie können jedes Objekt in der Orchestrator-API oder Objekte in jeder API aufrufen, die Sie in Orchestrator mithilfe eines Plug-Ins importieren.

Wenn ein Workflow ausgeführt wird, nimmt eine Aktion Eingabeparameter aus den Attributen des Workflows. Diese Attribute können entweder die ursprünglichen Eingabeparameter des Workflows sein oder Attribute, die von anderen Elementen im Workflow bei ihrer Ausführung festgelegt werden.

Dieses Kapitel behandelt die folgenden Themen:

- [„Wiederverwenden von Aktionen“](#), auf Seite 209
- [„Zugriff auf die Ansicht „Aktionen““](#), auf Seite 210
- [„Komponenten der Ansicht „Aktionen““](#), auf Seite 210
- [„Erstellen von Aktionen“](#), auf Seite 210
- [„Verwenden des Aktions-Versionsverlaufs“](#), auf Seite 213
- [„Wiederherstellen von gelöschten Aktionen“](#), auf Seite 214

## Wiederverwenden von Aktionen

Wenn Sie eine individuelle Funktion als Aktion definieren, haben Sie die Möglichkeit, sie nicht direkt in einem Workflowelement als skriptfähige Aufgabe zu programmieren, sondern in der Bibliothek einzustellen. Wenn eine Aktion in der Bibliothek sichtbar ist, können andere Workflows Sie verwenden.

Wenn Sie Aktionen unabhängig von den Workflows definieren, die sie aufrufen, können Sie die Aktionen einfacher aktualisieren oder optimieren. Durch das Definieren einzelner Aktionen wird anderen Workflows die Möglichkeit gegeben, Aktionen wiederzuverwenden. Wenn ein Workflow ausgeführt wird, setzt Orchestrator jede Aktion nur beim ersten Ausführen des Workflows in den Cache. Orchestrator kann dann die Aktion im Cache wiederverwenden. Das Speichern im Cache ist für rekursive Aufrufe in einem Workflow oder für schnelle Schleifenaufrufe sinnvoll.

Sie können Aktionen duplizieren, sie in andere Workflows oder Pakete exportieren oder zu einem anderen Modul in der hierarchischen Liste der Aktionen verschieben.

## Zugriff auf die Ansicht „Aktionen“

Die Benutzeroberfläche des Orchestrator-Clients verfügt über die Ansicht **Aktionen**, die Zugriff auf die Aktionsbibliotheken des Orchestrator-Server gewährt.

In der Ansicht **Aktionen** der Benutzeroberfläche des Orchestrator-Clients sehen Sie eine hierarchische Liste aller auf dem Orchestrator-Server verfügbaren Aktionen.

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Aktionen**.
- 3 Sie können die Aktionsbibliotheken durchsuchen, indem Sie die Knoten der hierarchischen Aktionsliste erweitern.

Mithilfe der Ansicht **Aktionen** können Sie Informationen zu den in den Bibliotheken enthaltenen Aktionen anzeigen sowie Aktionen erstellen und bearbeiten.

## Komponenten der Ansicht „Aktionen“

Wenn Sie auf eine Aktion in der hierarchischen Liste für Aktionen klicken, werden im rechten Fensterbereich des Orchestrator-Clients Informationen zu dieser Aktion angezeigt.

Die Ansicht **Aktionen** enthält vier Registerkarten.

<b>Allgemein</b>	Zeigt allgemeine Informationen zur Aktion an, u. a. den Namen, die Versionsnummer, die Berechtigungen und eine Beschreibung.
<b>Skripterstellung</b>	Zeigt Rückgabetypen, Eingabeparameter und den JavaScript-Code der Aktion an, der die Funktion der Aktion definiert.
<b>Ereignisse</b>	Zeigt alle Ereignisse an, die diese Aktion festgestellt oder ausgelöst hat.
<b>Berechtigungen</b>	Zeigt an, welche Benutzer und Benutzergruppen über die Berechtigung für den Zugriff auf diese Aktion verfügen.

## Erstellen von Aktionen

Sie können einzelne Funktionen als Aktionen definieren, die von anderen Elementen, beispielsweise Workflows, verwendet werden können. Aktionen sind JavaScript-Funktionen mit definierten Eingabe- und Ausgabeparametern sowie Berechtigungen.

- [Erstellen einer Aktion](#) auf Seite 211

Wenn Sie eine einzelne Funktion als Aktion definieren, haben Sie die Möglichkeit, sie nicht direkt in einem Workflowelement als skriptfähige Aufgabe zu programmieren, sondern in die Bibliothek einzustellen, damit sie auch von anderen Workflows benutzt werden kann.

- [Suchen nach Elementen, die eine Aktion implementieren](#) auf Seite 211

Wenn Sie eine Aktion bearbeiten und ihr Verhalten ändern, könnten Sie unabsichtlich einen Workflow oder eine Anwendung beschädigen, die diese Aktion implementiert. Orchestrator bietet eine Funktion zum Suchen nach allen Aktionen, Workflows oder Pakete, die ein bestimmtes Element implementieren. Sie können überprüfen, ob das Ändern des Elements Auswirkung auf andere Elemente hat.

- [Richtlinien für Aktionscodierung](#) auf Seite 212

Wenn Sie die Leistung von Workflows optimieren und die Möglichkeiten zur Wiederverwendung maximieren möchten, sollten Sie beim Erstellen von Aktionen einige grundlegende Codierungsrichtlinien beachten.

## Erstellen einer Aktion

Wenn Sie eine einzelne Funktion als Aktion definieren, haben Sie die Möglichkeit, sie nicht direkt in einem Workflowelement als skriptfähige Aufgabe zu programmieren, sondern in die Bibliothek einzustellen, damit sie auch von anderen Workflows benutzt werden kann.

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Aktionen**.
- 3 Erweitern Sie den Stammordner der hierarchischen Liste für die Aktionen und navigieren Sie zu dem Modul, in dem Sie die Aktion erstellen möchten.
- 4 Klicken Sie mit der rechten Maustaste auf das Modul und wählen Sie **Aktion hinzufügen**.
- 5 Geben Sie in das Textfeld einen Namen für die Aktion ein und klicken Sie auf **OK**.  
Ihre benutzerdefinierte Aktion wird der Bibliothek der Aktionen hinzugefügt.
- 6 Klicken Sie mit der rechten Maustaste auf die Aktion und wählen Sie **Bearbeiten**.
- 7 Klicken Sie auf die Registerkarte **Skripterstellung**.
- 8 Klicken Sie auf den Link **Void**, um den Standardrückgabebetyp zu ändern.
- 9 Fügen Sie die Eingabeparameter für die Aktion hinzu, indem Sie auf das Pfeilsymbol klicken.
- 10 Schreiben Sie das Skript für die Aktion.
- 11 Legen Sie die Berechtigungen für die Aktion fest.
- 12 Klicken Sie auf **Speichern und schließen**.

Sie haben eine benutzerdefinierte Aktion erstellt und die Eingabeparameter der Aktion hinzugefügt.

### Weiter

Sie können die neue benutzerdefinierte Aktion in einem Workflow verwenden.

## Suchen nach Elementen, die eine Aktion implementieren

Wenn Sie eine Aktion bearbeiten und ihr Verhalten ändern, könnten Sie unabsichtlich einen Workflow oder eine Anwendung beschädigen, die diese Aktion implementiert. Orchestrator bietet eine Funktion zum Suchen nach allen Aktionen, Workflows oder Pakete, die ein bestimmtes Element implementieren. Sie können überprüfen, ob das Ändern des Elements Auswirkung auf andere Elemente hat.

---

**WICHTIG** Die Funktion **Elemente suchen, die dieses Element verwenden** überprüft alle Pakete, Workflows und Richtlinien, sucht aber nicht in Skripten. Demzufolge kann das Ändern einer Aktion Auswirkungen auf ein Element haben, das diese Aktion in einem Skript aufruft, das von der Funktion **Elemente suchen, die dieses Element verwenden** nicht gefunden wurde.

---

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Aktionen**.
- 3 Erweitern Sie die Knoten der hierarchischen Aktionsliste, um zu einer bestimmten Aktion zu navigieren.

- 4 Klicken Sie mit der rechten Maustaste auf die Aktion und wählen Sie **Elemente suchen, die dieses Element verwenden**.

Ein Dialogfeld zeigt alle gefundenen Elemente wie Workflows oder Pakete an, die diese Aktion implementieren.

- 5 Doppelklicken Sie auf ein Element in der Ergebnisliste, um das Element im Orchestrator-Client anzuzeigen.

Sie haben alle Elemente gefunden, die eine Aktion implementieren.

### Weiter

Sie können überprüfen, ob das Ändern dieses Elements Auswirkung auf andere Elemente hat.

## Richtlinien für Aktionscodierung

Wenn Sie die Leistung von Workflows optimieren und die Möglichkeiten zur Wiederverwendung maximieren möchten, sollten Sie beim Erstellen von Aktionen einige grundlegende Codierungsrichtlinien beachten.

### Allgemeine Richtlinien für Aktionen

Wenn Sie eine Aktion erstellen, müssen Sie allgemeine Richtlinien beachten.

- Jede Aktion muss eine Beschreibung ihrer Rolle und Funktion enthalten.
- Schreiben Sie kurze, elementare Aktionen und kombinieren Sie sie in einem Workflow.
- Vermeiden Sie das Schreiben von Aktionen, die mehrere Funktionen ausführen, weil dies die Möglichkeiten zur Wiederverwendung beschränkt.
- Vermeiden Sie Aktionen, die in langen Zeiträumen ausgeführt werden. Erstellen Sie stattdessen eine Schleife im Workflow und fügen Sie nach dem Aktionselement ein Element „Warteereignis“ oder „Warte-Timer“ ein.
- Schreiben Sie keine Prüfpunkte in Aktionen. Workflows legen einen Prüfpunkt am Anfang und Ende der Ausführung jedes Elements fest.
- Vermeiden Sie das Schreiben von Schleifen in einer Aktion. Erstellen Sie stattdessen Schleifen im Workflow. Wenn der Server neu gestartet wird, wird ein laufender Workflow an seinem letzten Prüfpunkt am Anfang eines Elements fortgesetzt. Wenn Sie eine Schleife innerhalb einer Aktion schreiben und der Server neu gestartet wird, während der Workflow diese Aktion ausführt, wird der Workflow am Prüfpunkt zu Beginn dieser Aktion fortgesetzt, und die Schleife von Anfang an erneut gestartet.

### Richtlinien für das Benennen von Aktionen

Verwenden Sie allgemeine Richtlinien beim Benennen von Aktionen.

- Schreiben Sie Aktionsnamen auf Englisch.
- Beginnen Sie Aktionsnamen mit einem Kleinbuchstaben. Verwenden Sie einen Großbuchstaben am Anfang jedes verbundenen Worts im Namen. Beispiel: myAction.
- Wählen Sie möglichst explizite Aktionsnamen, damit die Funktion der Aktion klar ist. Beispiel: back-upAllVMsInPool.
- Wählen Sie möglichst explizite Modulnamen.
- Wählen Sie eindeutige Modulnamen.
- Verwenden Sie das umgekehrte Internet-Adressformat für Modulnamen. Beispiel: com.vmware.myactions.myAction.

## Richtlinien für Aktionsparameter

Verwenden Sie allgemeine Richtlinien beim Schreiben von Definitionen für Aktionsparameter.

- Schreiben Sie Parameternamen auf Englisch.
- Beginnen Sie Parameternamen mit einem Kleinbuchstaben.
- Wählen Sie möglichst explizite Parameternamen.
- Beschränken Sie Parameternamen vorzugsweise auf ein einzelnes Wort. Wenn ein Name mehr als ein Wort enthalten muss, verwenden Sie einen Großbuchstaben am Anfang jedes verbundenen Worts im Namen. Beispiel: `myParameter`.
- Verwenden Sie die Pluralform für Parameter, die für ein Array von Objekten stehen.
- Wählen Sie eindeutige Variablennamen, zum Beispiel `displayName`.
- Fügen Sie für jeden Parameter eine Beschreibung zur Erläuterung seines Zwecks hinzu.
- Verwenden Sie keine übermäßige Anzahl von Parametern in einer einzelnen Aktion.

## Verwenden des Aktions-Versionsverlaufs

Mithilfe des Versionsverlaufs können Sie eine Aktion auf eine frühere Version zurücksetzen. Sie können den Aktionszustand auf eine frühere oder spätere Aktionsversion zurücksetzen. Weiterhin können Sie die Unterschiede zwischen dem aktuellen Aktionszustand und einer gespeicherten früheren Version der Aktion vergleichen.

Orchestrator erstellt ein neues Versionsverlaufselement für eine Aktion, wenn Sie die Aktionsversionsnummer erhöhen und die Aktion speichern. Nachfolgende Änderungen an der Aktion bewirken keine Änderung am aktuellen Versionselement. Beispiel: Wenn Sie Aktionsversion 1.0.0 erstellen und speichern, wird der Zustand der Aktion in der Datenbank gespeichert. Wenn Sie Änderungen an der Aktion ausführen, können Sie den Aktionszustand im Orchestrator-Client speichern, können die Änderungen jedoch nicht auf Aktionsversion 1.0.0 anwenden. Um die Änderungen in der Datenbank zu speichern, müssen Sie eine Folgeversion der Aktion erstellen und speichern. Der Versionsverlauf wird zusammen mit der Aktion in der Datenbank gespeichert.

Wenn Sie eine Aktion löschen, kennzeichnet Orchestrator das Element in der Datenbank als gelöscht, ohne jedoch den Versionsverlauf des Elements aus der Datenbank zu löschen. Dadurch können Sie gelöschte Aktionen wiederherstellen. Weitere Informationen finden Sie unter [„Wiederherstellen von gelöschten Aktionen“](#), auf Seite 214.

### Voraussetzungen

Öffnen Sie eine Aktion zur Bearbeitung.

### Vorgehensweise

- 1 Klicken Sie im Aktions-Editor auf die Registerkarte **Allgemein**.
- 2 Klicken Sie auf **Versionsverlauf anzeigen**.  
Das Fenster mit dem Versionsverlauf wird angezeigt.
- 3 Wählen Sie eine Aktionsversion aus und klicken Sie auf **Vergleich mit aktuell**, um die Unterschiede zu sehen.  
Es wird ein Fenster eingeblendet, in dem die Unterschiede zwischen der aktuellen und der ausgewählten Aktionsversion angezeigt werden.

- 4 Wählen Sie eine Aktionsversion aus und klicken Sie auf **Zurücksetzen**, um den Zustand der Aktion wiederherzustellen.



**VORSICHT** Wenn Sie die aktuelle Aktionsversion nicht gespeichert haben, wird sie aus dem Versionsverlauf gelöscht, und Sie können die aktuelle Version nicht wiederherstellen.

Der Aktionszustand wird auf den Zustand der ausgewählten Version zurückgesetzt.

## Wiederherstellen von gelöschten Aktionen

Sie können Aktionen wiederherstellen, die aus der Bibliothek gelöscht wurden.

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Aktionen**.
- 3 Navigieren Sie zu dem Ordner, in dem Sie eine oder mehrere gelöschte Aktionen wiederherstellen möchten.
- 4 Klicken Sie mit der rechten Maustaste auf den Ordner und wählen Sie **Gelöschte Aktionen wiederherstellen**.
- 5 Wählen Sie die Aktionen aus, die Sie wiederherstellen möchten, und klicken Sie auf **Wiederherstellen**.

Die Aktionen werden im ausgewählten Ordner angezeigt.

## Erstellen von Ressourcenelementen

---

Workflows erfordern möglicherweise unabhängig von Orchestrator erstellte Objekte, die als Attribute dienen. Zur Nutzung externer Objekte als Attribute in Workflows importieren Sie diese als Ressourcenelemente in den Orchestrator Server.

Zu Objekten, die von Workflows als Ressourcenelemente verwendet werden können, zählen u.a. Bilddateien, Skripte, XML-Vorlagen und HTML-Dateien. Workflows, die auf dem Orchestrator-Server ausgeführt werden, können beliebige in Orchestrator importierte Ressourcenelemente verwenden.

Durch Importieren eines Objekts als Ressourcenelement in Orchestrator können Sie Änderungen an dem Objekt an einem einzelnen Speicherort vornehmen und diese Änderungen automatisch an alle Workflows weiterleiten, die dieses Ressourcenelement verwenden.

Sie können Ressourcenelemente in Ordnern organisieren. Die Maximalgröße eines Ressourcenelements beträgt 16 MB.

Dieses Kapitel behandelt die folgenden Themen:

- [„Anzeigen eines Ressourcenelements“](#), auf Seite 215
- [„Importieren eines externen Objekts zur Verwendung als Ressourcenelement“](#), auf Seite 216
- [„Bearbeiten der Ressourcenelementinformationen und Zugriffsrechte“](#), auf Seite 216
- [„Hinzufügen eines Ressourcenelements zu einer Datei“](#), auf Seite 217
- [„Aktualisieren von Ressourcenelementen“](#), auf Seite 218
- [„Hinzufügen eines Ressourcenelements zu einem Workflow“](#), auf Seite 218

### Anzeigen eines Ressourcenelements

Sie können vorhandene Ressourcenelemente im Orchestrator-Client anzeigen, um ihre Inhalte zu untersuchen und festzustellen, welche Workflows dieses Ressourcenelement verwenden.

#### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Ressourcen**.
- 3 Erweitern Sie die hierarchische Struktur (Viewer), um zu einem Ressourcenelement zu navigieren.
- 4 Klicken Sie auf ein Ressourcenelement, um Informationen dazu im rechten Bereich anzuzeigen.
- 5 Klicken Sie auf die Registerkarte **Viewer**, um die Inhalte des Ressourcenelements anzuzeigen.

- 6 Klicken Sie mit der rechten Maustaste auf das Ressourcenelement und wählen Sie **Elemente suchen, die dieses Element verwenden**.

Orchestrator listet alle Workflows auf, die dieses Ressourcenelement verwenden.

#### Weiter

Importieren und bearbeiten Sie ein Ressourcenelement.

## Importieren eines externen Objekts zur Verwendung als Ressourcenelement

Workflows erfordern möglicherweise unabhängig von Orchestrator erstellte Objekte, die als Attribute dienen. Zur Nutzung externer Objekte als Attribute in Workflows müssen Sie diese als Ressourcenelemente in den Orchestrator-Server importieren.

#### Voraussetzungen

Überprüfen Sie, ob Sie über eine Bilddatei, ein Skript, eine XML-Vorlage, eine HTML-Datei oder über einen anderen Objekttyp verfügen, den Sie importieren möchten.

#### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Ressourcen**.
- 3 Klicken Sie mit der rechten Maustaste in der hierarchischen Liste oder dem Stamm auf den Ordner „Ressource“ und wählen Sie **Neuer Ordner**, um einen Ordner zu erstellen, in dem Sie das Ressourcenelement speichern.
- 4 Klicken Sie mit der rechten Maustaste auf den Ordner „Ressource“, in den Sie das Ressourcenelement importieren möchten, und wählen Sie **Ressourcen importieren**.
- 5 Wählen Sie die zu importierende Ressource aus und klicken Sie auf **Öffnen**.

Orchestrator fügt das Ressourcenelement dem Ordner hinzu, den Sie ausgewählt haben.

Sie haben ein Ressourcenelement auf den Orchestrator-Server importiert.

#### Weiter

Bearbeiten Sie die allgemeinen Informationen zum Ressourcenelement und legen Sie die Zugriffsrechte für Benutzer fest.

## Bearbeiten der Ressourcenelementinformationen und Zugriffsrechte

Nach Importieren eines Objekts in den Orchestrator-Server als Ressourcenelement können Sie die Details und Berechtigungen des Ressourcenelements bearbeiten.


#### Voraussetzungen

Überprüfen Sie, ob Sie ein Image, ein Skript, eine XML- oder eine HTML-Datei oder einen beliebigen anderen Objekttyp in Orchestrator als Ressourcenelement importiert haben.

#### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Ressourcen**.
- 3 Klicken Sie mit der rechten Maustaste auf das Ressourcenelement und wählen Sie **Bearbeiten**.



- 4 Klicken Sie auf die Registerkarte **Allgemein** und legen Sie Name, Version und Beschreibung des Ressourcenelements fest.
- 5 Klicken Sie auf die Registerkarte **Berechtigungen** und dann auf das Symbol **Zugriffsrechte hinzufügen** () , um Berechtigungen für eine Benutzergruppe festzulegen.
- 6 Geben Sie im Textfeld **Filter** einen Benutzergruppennamen ein.
- 7 Wählen Sie eine Benutzergruppe aus und klicken Sie auf **OK**.
- 8 Klicken Sie mit der rechten Maustaste auf die Benutzergruppe und wählen Sie **Zugriffsrechte hinzufügen**.
- 9 Aktivieren Sie die entsprechenden Kontrollkästchen zum Festlegen der Berechtigungsstufe der Benutzergruppe und klicken Sie auf **OK**.  
  
Berechtigungen sind nicht kumulativ. Damit ein Benutzer das Ressourcenelement anzeigen, in eigenen Workflows verwenden und die Berechtigungen ändern kann, müssen alle Kontrollkästchen aktiviert sein.
- 10 Klicken Sie auf **Speichern und schließen**, um den Texteditor zu beenden.

Sie haben die allgemeinen Informationen zum Ressourcenelement bearbeitet und die Benutzerzugriffsrechte festgelegt.

#### Weiter

Speichern Sie das Ressourcenelement zum Aktualisieren in einer Datei oder fügen Sie es einem Workflow hinzu.

## Hinzufügen eines Ressourcenelements zu einer Datei

Sie können ein Ressourcenelement einer Datei auf Ihrem lokalen System hinzufügen. Das Speichern des Ressourcenelements als Datei ermöglicht seine Bearbeitung.

Ressourcenelemente können nicht in Orchestrator-Client bearbeitet werden. Beispiel: Wenn es sich bei dem Ressourcenelement um eine XML-Konfigurationsdatei oder ein Skript handelt, müssen Sie es zum Bearbeiten lokal speichern.

#### Voraussetzungen

Vergewissern Sie sich, dass der Orchestrator-Server ein Ressourcenelement enthält, das Sie in einer Datei speichern können.

#### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Ressourcen**.
- 3 Klicken Sie mit der rechten Maustaste auf das Ressourcenelement und wählen Sie **In Datei speichern**.
- 4 Nehmen Sie die erforderlichen Änderungen an der Datei vor.

Sie haben ein Ressourcenelement in einer Datei gespeichert.

#### Weiter

Aktualisieren Sie das Ressourcenelement auf dem Orchestrator-Server.

## Aktualisieren von Ressourcenelementen

Für das Aktualisieren eines Ressourcenelements müssen Sie dieses in das Dateisystem exportieren, die exportierte Datei mit einem geeigneten Werkzeug bearbeiten und das Ressourcenelement durch Importieren der bearbeiteten Datei aktualisieren.

### Voraussetzungen

Überprüfen Sie, ob Sie ein Image, ein Skript, eine XML- oder eine HTML-Datei oder einen beliebigen anderen Objekttyp in Orchestrator als Ressourcenelement importiert haben.

### Vorgehensweise

- 1 Ändern Sie die Quelldatei des Ressourcenelements auf Ihrem lokalen System.
- 2 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 3 Klicken Sie auf die Ansicht **Ressourcen**.
- 4 Navigieren Sie in der hierarchischen Liste zu dem Ressourcenelement, das Sie aktualisiert haben.
- 5 Klicken Sie mit der rechten Maustaste auf das Ressourcenelement und wählen Sie **Ressource aktualisieren**.
- 6 (Optional) Klicken Sie auf die Registerkarte **Viewer**, um sich zu vergewissern, dass Orchestrator das Ressourcenelement aktualisiert hat.

Sie haben ein Ressourcenelement auf dem Orchestrator-Server aktualisiert.

## Hinzufügen eines Ressourcenelements zu einem Workflow

Ressourcenelemente sind externe Objekte, die Sie in den Orchestrator-Server importieren und die von Workflows bei der Ausführung als Attribut verwendet werden können. Beispielsweise kann ein Workflow bei seiner Ausführung eine importierte XML-Datei verwenden, die eine Zuordnung zur Konvertierung eines Datentyps in einen anderen definiert, oder ein Skript, das eine Funktion definiert.

### Voraussetzungen

Überprüfen Sie, ob der Orchestrator-Server die folgenden Objekte aufweist:

- Ein Image, ein Skript, eine XML- oder eine HTML-Datei oder ein beliebiger anderer in Orchestrator als Ressourcenelement importierter Objekttyp.
- Ein Workflow, der das Ressourcenelement als Attribut benötigt.

### Vorgehensweise

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Entwurf** aus.
- 2 Klicken Sie auf die Ansicht **Workflows**.
- 3 Erweitern Sie die hierarchische Struktur (Viewer), um zum Workflow zu navigieren, der das Ressourcenelement als Attribut benötigt.
- 4 Klicken Sie mit der rechten Maustaste auf den Workflow und wählen Sie **Bearbeiten**.
- 5 Klicken Sie auf der Registerkarte **Allgemein** im Bereich „Attribute“ auf das Symbol **Attribut hinzufügen** (A+).
- 6 Klicken Sie auf den Attributnamen und geben Sie einen neuen Namen für das Attribut ein.
- 7 Klicken Sie auf **Typ**, um den Attributtyp festzulegen.

- 8 Im Dialogfeld **Typ auswählen** geben Sie **Ressource** im Feld **Filter** ein, um nach einem Objekttyp zu suchen.

Option	Aktion
<b>Definieren eines einzelnen Ressourcenelements als Attribut</b>	Wählen Sie ResourceElement in der Liste aus.
<b>Definieren eines Ordners, der mehrere Ressourcenelemente als Attribut enthält</b>	Wählen Sie ResourceElementCategory in der Liste aus.

- 9 Klicken Sie auf **Wert** und geben Sie den Namen des Ressourcenelements oder die Kategorie der Ressourcenelemente in das Textfeld **Filter** ein.
- 10 Wählen Sie aus der vorgeschlagenen Liste das Ressourcenelement oder einen Ordner mit Ressourcenelementen und klicken Sie auf **Auswählen**.
- 11 Klicken Sie auf **Speichern und schließen**, um den Texteditor zu beenden.

Sie haben ein Ressourcenelement oder einen Ordner von Ressourcenelementen als ein Attribut in einem Workflow hinzugefügt.



## Erstellen von Paketen

---

Pakete werden verwendet, um Inhalte von einem Orchestrator-Server zu einem anderen zu verteilen. Pakete können Workflows, Aktionen, Richtlinienvorlagen, Konfigurationen oder Ressourcen enthalten.

Wenn Sie einem Paket ein Element hinzufügen, prüft der Orchestrator, ob Abhängigkeiten vorhanden sind, und fügt eventuell abhängige Elemente dem Paket hinzu. Beispiel: Wenn Sie einen Workflow hinzufügen, der Aktionen oder andere Workflows verwendet, fügt Orchestrator diese Aktionen und Workflows dem Paket hinzu.

Wenn Sie ein Paket importieren, vergleicht der Server die Versionen der verschiedenen Elemente seines Inhalts mit übereinstimmenden lokalen Elementen. Der Vergleich zeigt Unterschiede in den Versionen zwischen den lokalen und importierten Elementen. Der Administrator kann entscheiden, ob das Paket importiert wird, oder bestimmte Elemente zum Importieren auswählen.

Pakete verwenden Digital Rights Management, um zu kontrollieren, wie der empfangende Server die Inhalte des Pakets verwenden kann. Orchestrator signiert die Pakete und verschlüsselt die Pakete zum Datenschutz. Pakete können mithilfe von X509-Zertifikaten verfolgen, welche Benutzer Elemente exportieren und weiterverteilen.

Weitere Informationen über die Verwendung von Paketen finden Sie unter *Verwenden des VMware vRealize Orchestrator-Clients*.

- [Erstellen eines Pakets](#) auf Seite 221

Sie können Workflows, Richtlinienvorlagen, Aktionen, Plug-In-Referenzen, Ressourcen und Konfigurationselemente in Paketen exportieren. Alle Elemente, die von einem Element in einem Paket implementiert werden, werden dem Paket automatisch hinzugefügt, um die Kompatibilität zwischen den Versionen zu gewährleisten. Wenn Sie die referenzierten Elemente nicht hinzufügen möchten, können Sie sie im Paketerstellungseditor löschen.

- [Setzen der Benutzerberechtigungen für ein Paket](#) auf Seite 222

Sie legen verschiedene Berechtigungsebenen für ein Paket fest, um den Zugriff verschiedener Benutzer oder Benutzergruppen auf die Inhalte dieses Pakets zu begrenzen.

## Erstellen eines Pakets

Sie können Workflows, Richtlinienvorlagen, Aktionen, Plug-In-Referenzen, Ressourcen und Konfigurationselemente in Paketen exportieren. Alle Elemente, die von einem Element in einem Paket implementiert werden, werden dem Paket automatisch hinzugefügt, um die Kompatibilität zwischen den Versionen zu gewährleisten. Wenn Sie die referenzierten Elemente nicht hinzufügen möchten, können Sie sie im Paketerstellungseditor löschen.

### Voraussetzungen

Überprüfen Sie, ob der Orchestrator-Server Elemente wie Workflows, Aktionen und Richtlinienvorlagen enthält, die Sie einem Paket hinzufügen können.

**Vorgehensweise**

- 1 Wählen Sie im Dropdown-Menü im Orchestrator-Client **Verwalten** aus.
- 2 Klicken Sie auf die Ansicht **Pakete**.
- 3 Klicken Sie mit der rechten Maustaste im linken Fensterbereich und wählen Sie **Paket hinzufügen**.
- 4 Geben Sie den Namen des neuen Pakets ein und klicken Sie auf **OK**.  
Die Syntax für Paketnamen ist *Domäne.Ihr\_Unternehmen.Ordnner.Paketname*  
Beispiel: *com.vmware.meinOrdner.meinPaket*.
- 5 Klicken Sie mit der rechten Maustaste auf das Paket und wählen Sie **Bearbeiten**.  
Der Paketeditor wird geöffnet.
- 6 Fügen Sie auf der Registerkarte **Allgemein** eine Beschreibung des Pakets hinzu.
- 7 Fügen Sie dem Paket auf der Registerkarte **Workflows** die Workflows hinzu.
  - Klicken Sie auf **Workflows einfügen (Listensuche)**, um nach Workflows zu suchen und sie in einem Auswahldialogfeld auszuwählen.
  - Klicken Sie auf **Workflows einfügen (Navigation in Baumstruktur)**, um nach Ordnern mit Workflows aus der hierarchischen Liste zu suchen und einen Ordner auszuwählen.
- 8 Auf den Registerkarten **Richtlinienvorlagen**, **Aktionen**, **Konfigurationen**, **Ressourcen** und **Verwendete Plug-Ins** fügen Sie dem Paket Richtlinien Vorlagen, Aktionen, Konfigurationselemente, Ressourcenelemente und Plug-Ins hinzu.
- 9 Klicken Sie auf **Speichern und schließen**, um den Texteditor zu beenden.

Sie haben ein Paket erstellt und ihm Elemente hinzugefügt.

**Weiter**

Richten Sie Benutzerrechte für dieses Paket ein.

**Setzen der Benutzerberechtigungen für ein Paket**

Sie legen verschiedene Berechtigungsebenen für ein Paket fest, um den Zugriff verschiedener Benutzer oder Benutzergruppen auf die Inhalte dieses Pakets zu begrenzen.


Sie können die verschiedenen Benutzer und Benutzergruppen, für die Berechtigungen erstellt werden, aus den Benutzern und Benutzergruppen im Orchestrator LDAP oder vCenter Single Sign-On-Server auswählen. Orchestrator definiert Berechtigungsebenen, die Sie auf Benutzer oder Gruppen anwenden können.

<b>Ansicht</b>	Der Benutzer kann die Elemente im Paket sehen, aber nicht die Schemas oder die Skripte.
<b>Überprüfen</b>	Der Benutzer kann die Elemente im Paket einschließlich der Schemas oder Skripte sehen.
<b>Bearbeiten</b>	Der Benutzer kann die Elemente im Paket bearbeiten.
<b>Admin</b>	Der Benutzer kann Berechtigungen für die Elemente im Paket festlegen.

**Voraussetzungen**

Erstellen Sie ein Paket, öffnen Sie es zur Bearbeitung im Paketeditor und fügen Sie die erforderlichen Elemente dem Paket hinzu.

### Vorgehensweise

- 1 Klicken Sie auf die Registerkarte **Berechtigungen** im Paketeditor
- 2 Klicken Sie auf das Symbol **Zugriffsrechte hinzufügen** () , um Berechtigungen für einen neuen Benutzer oder eine neue Benutzergruppe festzulegen.
- 3 Suchen Sie nach einem Benutzer oder einer Benutzergruppe.  
Die Suchergebnisse zeigen alle Benutzer und Benutzergruppen, die mit den Suchkriterien übereinstimmen.
- 4 Wählen Sie einen Benutzer oder eine Benutzergruppe.
- 5 Aktivieren Sie die entsprechenden Kontrollkästchen zum Festlegen der Berechtigungsstufe der Benutzergruppe und klicken Sie auf **Auswählen**.  
Um einem Benutzer zu erlauben, die Elemente anzuzeigen, das Schema und die Skripte zu überprüfen, Elemente auszuführen und zu bearbeiten sowie die Berechtigungen zu ändern, müssen Sie alle Kontrollkästchen aktivieren.
- 6 Klicken Sie auf **Speichern und schließen**, um den Texteditor zu beenden.

Sie haben ein Paket erstellt und die geeigneten Benutzerberechtigungen festgelegt.





# Entwickeln von Plug-Ins

---

Orchestrator ermöglicht dank seiner offenen Plug-In-Architektur die Integration in Management- und Verwaltungslösungen. Sie nutzen den Orchestrator-Client zum Ausführen und Erstellen von Plug-In-Workflows und zum Zugriff auf die Plug-In-API.

Dieses Kapitel behandelt die folgenden Themen:

- [„Überblick über Plug-Ins“](#), auf Seite 225
- [„Inhalt und Struktur von Plug-Ins“](#), auf Seite 233
- [„Orchestrator Plug-In-API-Referenz“](#), auf Seite 237
- [„Elemente der Plug-In-Definitionsdatei vso.xml“](#), auf Seite 248
- [„Best Practices zur Entwicklung von Orchestrator-Plug-Ins“](#), auf Seite 265

## Überblick über Plug-Ins

Orchestrator-Plug-Ins müssen einen Standardsatz von Komponenten enthalten und eine Standardarchitektur aufweisen. Diese Verfahren unterstützen Sie beim Erstellen von Plug-Ins für die weitest mögliche Varianz an externen Technologien.

- [Struktur eines Orchestrator-Plug-Ins](#) auf Seite 226  
Orchestrator-Plug-Ins haben eine grundlegende Struktur, die aus verschiedenen Typen von Ebenen besteht, die spezielle Funktionen implementieren.
- [Verfügbarmachen einer externen API in Orchestrator](#) auf Seite 228  
Die API eines externen Produkts können Sie durch Erstellen eines Orchestrator-Plug-Ins auf der Orchestrator-Plattform verfügbar machen. Ein solches Plug-In kann für jede Anwendung erstellt werden, deren API durch Zuordnung zu JavaScript-Objekten, die von Orchestrator verwendet werden können, verfügbar gemacht werden kann.
- [Komponenten eines Plug-Ins](#) auf Seite 228  
Plug-Ins bestehen aus einem Standardsatz von Komponenten, die die Objekte in der Plug-In-Technologie für die Orchestrator-Plattform verfügbar machen.
- [Rolle der Datei vso.xml](#) auf Seite 229  
Sie verwenden die Datei `vso.xml`, um die Objekte, Klassen, Methoden und Attribute der integrierten Technologie zu Bestandslistenobjekten, Skripterstellungstypen, Skripterstellungsklassen, Skripterstellungsmethoden und Attributen von Orchestrator zuzuordnen. Außerdem wird in der Datei `vso.xml` das Start- und Konfigurationsverhalten des Plug-Ins definiert.

- [Rollen des Plug-In-Adapters](#) auf Seite 230  
Der Plug-In-Adapter ist der Einstiegspunkt des Plug-Ins in den Orchestrator-Server. Der Plug-In-Adapter dient als Datenspeicher für die Plug-In-Technologie im Orchestrator-Server, erstellt eine Plug-In-Factory und verwaltet Ereignisse, die in der Plug-In-Technologie auftreten.
- [Rollen der Plug-In-Factory](#) auf Seite 231  
Die Plug-In-Factory definiert, wie Orchestrator Objekte in der Plug-In-Technologie findet und damit arbeitet.
- [Rolle von Finder-Objekten](#) auf Seite 231  
Finder-Objekte erkennen und finden bestimmte Instanzen von verwalteten Objekttypen in der integrierten Technologie. Orchestrator kann Objekte, die in der integrierten Technologie gefunden werden, ändern und mit ihnen interagieren, indem es Workflows für die Finder-Objekte ausführt.
- [Rolle von Skriptobjekten](#) auf Seite 232  
Skriptobjekte sind JavaScript-Darstellungen von Objekten aus der integrierten Technologie. Skriptobjekte aus Plug-Ins werden in der Orchestrator-JavaScript-API angezeigt. Sie können sie in Elementen mit Skript in Workflows und Aktionen verwenden.
- [Rolle von Ereignishandlern](#) auf Seite 232  
Ereignisse sind Änderungen an Zuständen oder Attributen der Objekte, die Orchestrator in der Plug-In-Technologie findet. Orchestrator überwacht Ereignisse durch Implementieren von Ereignishandlern.

## Struktur eines Orchestrator-Plug-Ins

Orchestrator-Plug-Ins haben eine grundlegende Struktur, die aus verschiedenen Typen von Ebenen besteht, die spezielle Funktionen implementieren.

Die unteren drei Ebenen eines Orchestrator-Plug-Ins, also die Infrastrukturklassen, Umschließungsklassen und Skriptobjekte, implementieren die Verbindung zwischen Plug-In-Technologie und Orchestrator.

Die für den Benutzer sichtbaren Ebenen eines Orchestrator-Plug-Ins sind die oberen drei Ebenen: Aktionen, Bausteine und hochrangige Workflows.

**Abbildung 6-1.** Struktur eines Orchestrator-Plug-Ins

<b>Infrastrukturklassen</b>	Ein Satz an Klassen, die die Verbindung zwischen Plug-In-Technologie und Orchestrator herstellen. Die Infrastrukturklassen umfassen die Klassen, die entsprechend der Plug-In-Definition implementiert werden (wie Plug-In-Factory, Plug-In-Adaptor usw.). Die Infrastrukturklassen umfassen auch die Klassen, die Funktionen für häufige Aufgaben und Objekte wie Hilfsprogramme, Zwischenspeichern, Bestandslisten usw. bereitstellen.
<b>Umschließungsklassen</b>	Ein Satz an Klassen, die das Objektmodell der Plug-In-Technologie an das Objektmodell anpassen, das Sie im Orchestrator darstellen möchten.
<b>Skriptobjekte</b>	JavaScript-Objekttypen, die Zugriff auf Umschließungsklassen, Methoden und Attribute in der Plug-In-Technologie bieten. In der Datei <code>vso.xml</code> definieren Sie Umschließungsklassen, Attribute und Methoden aus der Plug-In-Technologie, die im Orchestrator dargelegt wird.
<b>Aktionen</b>	Ein Satz aus JavaScript-Funktionen, die Sie direkt in Workflows und Skriptaufgaben verwenden können. Aktionen können mehrere Eingabeparameter besitzen und haben einen einzelnen Rückgabewert.
<b>Zusammengesetzte Workflows</b>	Ein Satz aus Workflows, die alle generischen Funktionen abdecken, die Sie mit dem Plug-In bereitstellen möchten. Normalerweise stellt ein zusammengesetzter Workflow einen Vorgang in der Benutzerschnittstelle der Orchestrator-gesteuerten Technologie dar. Die zusammengesetzten Workflows können direkt verwendet oder in hochrangige Workflows einbezogen werden.
<b>Hochrangige Workflows</b>	Ein Satz aus Workflows, der spezielle Funktionen des Plug-Ins abdeckt. Sie können hochrangige Workflows bereitstellen, um konkrete Anforderungen zu erfüllen, oder um komplexe Beispiele der Plug-In-Nutzung zu zeigen.

## Verfügbarmachen einer externen API in Orchestrator

Die API eines externen Produkts können Sie durch Erstellen eines Orchestrator-Plug-Ins auf der Orchestrator-Plattform verfügbar machen. Ein solches Plug-In kann für jede Anwendung erstellt werden, deren API durch Zuordnung zu JavaScript-Objekten, die von Orchestrator verwendet werden können, verfügbar gemacht werden kann.

Die Plug-Ins ordnen Java-Objekte und -Methoden JavaScript-Objekten zu, die sie dann der Orchestrator-Skript-API hinzufügen. Wird von einer externen Anwendung eine Java-API verfügbar gemacht, können Sie diese unmittelbar JavaScript-Objekten zuordnen, die Orchestrator dann für Workflows und Aktionen verwenden kann.

Für Anwendungen, deren API in einer anderen Sprache als Java verfügbar gemacht wird, können Sie Plug-Ins mithilfe von WSDL (Web Service Definition Language), REST (Representational State Transfer) oder eines Messaging-Diensts erstellen. Die Plug-Ins integrieren die verfügbar gemachte API dann in Java-Objekte. Diese integrierten Java-Objekte können Sie anschließend JavaScript-Objekten zuordnen, die mit Orchestrator kompatibel sind.

Die Plug-In-Technologie ist unabhängig von Orchestrator. Sie können Orchestrator-Plug-Ins für externe Produkte selbst dann erstellen, wenn Sie statt auf den Quellcode nur auf den binären Code zugreifen können. Dies gilt beispielsweise für Java-Archive (JAR-Dateien).

## Komponenten eines Plug-Ins

Plug-Ins bestehen aus einem Standardsatz von Komponenten, die die Objekte in der Plug-In-Technologie für die Orchestrator-Plattform verfügbar machen.

Die wichtigsten Komponenten eines Plug-Ins sind der Plug-In-Adapter, die Factory und Ereignisimplementierungen. Sie ordnen die Objekte und Vorgänge, die im Adapter, in der Factory und in den Ereignisimplementierungen definiert sind, Orchestrator-Objekten in einer XML-Definitionsdatei namens `vso.xml` zu. Die `vso.xml`-Datei ordnet Objekte und Funktionen aus der Plug-In-Technologie den JavaScript-Skriptobjekten zu, die in der Orchestrator JavaScript-API erscheinen. Die `vso.xml`-Datei ordnet auch Objekttypen aus der Plug-In-Technologie Findern zu, die auf der Orchestrator-Registerkarte **Bestand** erscheinen.

Plug-Ins bestehen aus den folgenden Komponenten.

### Plug-In-Modul

Das Plug-In selbst, das als eine Gruppe von Java-Klassen definiert wird, eine `vso.xml`-Datei und Pakete der Workflows und Aktionen, die mit den Objekten interagieren, auf die Sie über das Plug-In zugreifen. Das Plug-In-Modul ist unbedingt erforderlich.

### Plug-In-Adapter

Definiert die Schnittstelle zwischen der Plug-In-Technologie und dem Orchestrator-Server. Der Adapter ist der Einstiegspunkt des Plug-Ins in die Orchestrator-Plattform. Der Adapter erstellt die Plug-In-Factory, verwaltet das Laden und Entladen des Plug-Ins sowie die Ereignisse, die im Zusammenhang mit den Objekten in der Plug-In-Technologie auftreten. Der Plug-In-Adapter ist unbedingt erforderlich.

### Plug-In-Factory

Definiert, wie Orchestrator Objekte in der Plug-In-Technologie findet und damit Vorgänge vornimmt. Der Adapter erstellt eine Factory für die Client-sitzung, die sich zwischen Orchestrator und einer Plug-In-Technologie öffnet. Die Factory ermöglicht es Ihnen einerseits, eine Sitzung zwischen allen Clientverbindungen gemeinsam zu nutzen, oder andererseits eine Sitzung pro Clientverbindung zu öffnen. Die Plug-In-Factory ist unbedingt erforderlich.

<b>Konfiguration</b>	Orchestrator definiert keine Standardart, wie das Plug-In seine Konfiguration speichert. Sie können Konfigurationsinformationen mithilfe von Windows-Registern, statischen Konfigurationsdateien, einer Datenbank oder in XML-Dateien speichern. Orchestrator-Plug-Ins können durch das Ausführen von Konfigurationsworkflows im Orchestrator-Client konfiguriert werden.
<b>Finder</b>	Interaktionsregeln, die definieren, wie Orchestrator Objekte in der Plug-In-Technologie ermittelt und darstellt. Finder rufen Objekte aus der Gruppe von Objekten ab, die die Plug-In-Technologie für Orchestrator bereitstellt. Sie definieren in der <code>vso.xml</code> -Datei die Beziehungen zwischen Objekten, um die Navigation durch das Netzwerk von Objekten zu ermöglichen. Orchestrator stellt das Objektmodell der Plug-In-Technologie auf der Registerkarte <b>Bestand</b> dar. Finder sind unbedingt erforderlich, wenn Sie Objekte in der Plug-In-Technologie für Orchestrator verfügbar machen möchten.
<b>Skriptobjekte</b>	JavaScript-Objekttypen, die Zugriff auf Objekte, Vorgänge und Attribute in der Plug-In-Technologie bieten. Skriptobjekte definieren, wie Orchestrator auf das Objektmodell der Plug-In-Technologie über JavaScript zugreift. Sie ordnen die Klassen und Methoden der Plug-In-Technologie den JavaScript-Objekten in der <code>vso.xml</code> -Datei zu. Sie können auf die JavaScript-Objekte in der Orchestrator-Skripterstellungs-API zugreifen und sie in Orchestrator-Skript Aufgaben, Aktionen und Workflows integrieren. Skriptobjekte sind unbedingt erforderlich, wenn Sie Skripttypen, Klassen und Methoden der Orchestrator-JavaScript-API hinzufügen möchten.
<b>Bestandsliste</b>	Instanzen von Objekten in der Plug-In-Technologie, die Orchestrator mithilfe von Findern erkennt, erscheinen in der Ansicht <b>Bestand</b> im Orchestrator-Client. Sie können mit den Objekten im Bestand arbeiten, indem Sie an ihnen Workflows ausführen. Der Bestand ist optional. Sie können ein Plug-In erstellen, das nur Skripttypen und Klassen der Orchestrator-JavaScript-API hinzufügt und keine Instanzen von Objekten im Bestand verfügbar macht.
<b>Ereignisse</b>	Änderungen des Status eines Objekts in der Plug-In-Technologie. Orchestrator kann passiv auf Ereignisse warten, die in der Plug-In-Technologie auftreten. Orchestrator kann auch aktiv Ereignisse in der Plug-In-Technologie auslösen. Ereignisse sind optional.

## Rolle der Datei `vso.xml`

Sie verwenden die Datei `vso.xml`, um die Objekte, Klassen, Methoden und Attribute der integrierten Technologie zu Bestandslistenobjekten, Skripterstellungstypen, Skripterstellungsklassen, Skripterstellungsmethoden und Attributen von Orchestrator zuzuordnen. Außerdem wird in der Datei `vso.xml` das Start- und Konfigurationsverhalten des Plug-Ins definiert.

Die Datei `vso.xml` erfüllt hauptsächlich die folgenden Rollen.

<b>Start- und Konfigurationsverhalten</b>	Definiert, wie das Plug-In gestartet wird, und findet durch das Plug-In definierte Konfigurationsimplementierungen. Lädt den Plug-In-Adapter.
<b>Bestandslistenobjekte</b>	Definiert die Objekttypen, auf die das Plug-In in der integrierten Technologie zugreift. Mit den Finder-Methoden der Plug-In-Factory-Implementierung werden Instanzen dieser Objekte gefunden und in der Orchestrator-Bestandsliste angezeigt.
<b>Skripterstellungstypen</b>	Fügt der Orchestrator-JavaScript-API Skripterstellungstypen hinzu, um die verschiedenen Objekttypen in der Bestandsliste darzustellen. Sie können diese Skripterstellungstypen als Eingabeparameter in Workflows verwenden.

<b>Skripterstellungsklassen</b>	Fügt der Orchestrator-JavaScript-API Klassen hinzu, die Sie in Elementen mit Skript in Workflows, Aktionen, Richtlinien usw. verwenden können.
<b>Skripterstellungsmethoden</b>	Fügt der Orchestrator-JavaScript-API Methoden hinzu, die Sie in Elementen mit Skript in Workflows, Aktionen, Richtlinien usw. verwenden können.
<b>Skriptstellungsattribute</b>	Fügt der Orchestrator-JavaScript-API die Attribute der Objekte aus der integrierten Technologie hinzu, die Sie in Elementen mit Skript in Workflows, Aktionen, Richtlinien usw. verwenden können.

## Rollen des Plug-In-Adapters

Der Plug-In-Adapter ist der Einstiegspunkt des Plug-Ins in den Orchestrator-Server. Der Plug-In-Adapter dient als Datenspeicher für die Plug-In-Technologie im Orchestrator-Server, erstellt eine Plug-In-Factory und verwaltet Ereignisse, die in der Plug-In-Technologie auftreten.

Zum Erstellen eines Plug-In-Adapters erstellen Sie eine Java-Klasse, die die Schnittstelle `IPluginAdaptor` implementiert.

Die Plug-In-Adapter-Klasse, die Sie erstellen, verwaltet die Plug-In-Factory, Ereignisse und Auslöser in der Plug-In-Technologie. Die Schnittstelle `IPluginAdaptor` bietet Methoden, die Sie zum Ausführen dieser Aufgaben verwenden.

Der Plug-In-Adapter erfüllt hauptsächlich die folgenden Rollen.

<b>Erstellt eine Factory</b>	Die wichtigste Rolle für den Plug-In-Adapter besteht darin, eine Plug-In-Factory-Instanz für jede Verbindung vom Orchestrator in die Plug-In-Technologie zu laden und zu entladen. Die Plug-In-Adapter-Klasse ruft die Methode <code>IPluginAdaptor.createPluginFactory()</code> auf, um eine Instanz einer Klasse zu erstellen, die die Schnittstelle <code>IPluginFactory</code> implementiert.
<b>Verwalten von Ereignissen</b>	Der Plug-In-Adapter ist die Schnittstelle zwischen dem Orchestrator-Server und der Plug-In-Technologie. Der Plug-In-Adapter verwaltet die Ereignisse, die der Orchestrator für die Objekte in einer Plug-In-Technologie ausführt oder überwacht. Der Adapter verwaltet Ereignisse durch Ereignisherausgeber. Ereignisherausgeber sind Instanzen der Schnittstelle <code>IPluginEventPublisher</code> , die der Adapter durch den Aufruf der Methode <code>IPluginAdaptor.registerEventPublisher()</code> erstellt. Ereignisherausgeber setzen Auslöser und Kontrollen auf Objekte in der Plug-In-Technologie, damit Orchestrator definierte Aktionen starten kann, wenn bestimmte Ereignisse mit dem Objekt eintreten oder die Werte des Objekts bestimmte Schwellenwerte überschreiten. Sie können ferner <code>PluginTrigger</code> - und <code>PluginWatcher</code> -Instanzen definieren, die Ereignisse festlegen, auf die „Warteereignis“-Elemente in Workflows mit langen Ausführungszeiten warten.
<b>Legt den Plug-In-Namen fest</b>	Sie übergeben einen Namen für das Plug-In in der <code>vso.xml</code> -Datei. Der Plug-In-Adapter bezieht diesen Namen aus der <code>vso.xml</code> -Datei und veröffentlicht ihn in der Ansicht <b>Bestand</b> im Orchestrator-Client.
<b>Installiert Lizenzen</b>	Sie können Methoden aufrufen, um Lizenzdateien zu installieren, die die Plug-In-Technologie in der Adapterimplementierung erfordert.

Vollständige Details zur `IPluginAdaptor`-Schnittstelle, zu allen ihren Methoden und allen anderen Klassen der Plug-In-API finden Sie unter „[Orchestrator Plug-In-API-Referenz](#)“, auf Seite 61.

## Rollen der Plug-In-Factory

Die Plug-In-Factory definiert, wie Orchestrator Objekte in der Plug-In-Technologie findet und damit arbeitet.

Zum Erstellen einer Plug-In-Factory müssen Sie die Schnittstelle `IPluginFactory` aus der Orchestrator-Plug-In-API implementieren und erweitern. Die Plug-In-Factory-Klasse, die Sie erstellen, definiert die Finderfunktionen, die Orchestrator verwendet, um auf Objekte in der Plug-In-Technologie zuzugreifen. Die Factory ermöglicht es dem Orchestrator-Server, Objekte nach ihrem Bezeichner, ihrer Beziehung mit anderen Objekten oder durch eine Suche nach einer Abfragezeichenfolge zu finden.

Die Plug-In-Factory übernimmt hauptsächlich folgende Aufgaben.

<b>Suche nach Objekten</b>	Sie können Funktionen erstellen, die Objekte nach Namen und Typen finden. Sie finden Objekte nach Namen und Typen mithilfe der Methode <code>IPluginFactory.find()</code> .
<b>Suche nach Objekten mit Beziehungen zu anderen Objekten</b>	Sie können Funktionen erstellen, um Objekte zu finden, die mit einem bestimmten Objekt über einen bestimmten Beziehungstyp verknüpft sind. Beziehungen definieren Sie in der <code>vso.xml</code> -Datei. Sie können auch Finder erstellen, die abhängige untergeordnete Objekte finden, deren Beziehung mit allen übergeordneten Objekten durch einen bestimmten Beziehungstyp bestimmt wird. Sie implementieren die Methode <code>IPluginFactory.findRelation()</code> , um Objekte zu finden, die mit einem bestimmten übergeordneten Objekt durch einen bestimmten Beziehungstyp verknüpft sind. Sie implementieren die Methode <code>IPluginFactory.hasChildrenInRelation()</code> , um zu ermitteln, ob mindestens ein untergeordnetes Objekt für eine übergeordnete Instanz existiert.
<b>Definieren von Abfragen, um Objekte nach Ihren Kriterien zu finden</b>	Sie können Objektfinder erstellen, die von Ihnen definierte Abfragerregeln implementieren. Sie implementieren die Methode <code>IPluginFactory.findAll()</code> , um alle Objekte zu finden, die bestimmten, von Ihnen definierten Abfragerregeln entsprechen, wenn die Factory diese Methode aufruft. Sie erhalten die Ergebnisse der Methode <code>findAll()</code> in einem <code>QueryResult</code> -Objekt, das eine Liste aller gefundenen Objekte enthält, die mit den von Ihnen definierten Abfragerregeln übereinstimmen.

Weitere Informationen zur Schnittstelle `IPluginFactory`, zu allen ihren Methoden und allen anderen Klassen der Plug-In-API finden Sie unter „[Orchestrator Plug-In-API-Referenz](#)“, auf Seite 61.

## Rolle von Finder-Objekten

Finder-Objekte erkennen und finden bestimmte Instanzen von verwalteten Objekttypen in der integrierten Technologie. Orchestrator kann Objekte, die in der integrierten Technologie gefunden werden, ändern und mit ihnen interagieren, indem es Workflows für die Finder-Objekte ausführt.

Jede Instanz eines bestimmten verwalteten Objekttyps in der integrierten Technologie muss einen eindeutigen Bezeichner aufweisen, damit Orchestrator-Finder-Objekte sie finden können. Die integrierte Technologie stellt die eindeutigen Bezeichner für die Objektinstanzen als Zeichenfolgen bereit. Wenn ein Workflow ausgeführt wird, legt Orchestrator die eindeutigen Bezeichner der gefundenen Objekte als Workflow-Attributwerte fest. Workflows, die ein Objekt eines bestimmten Typs als Eingabeparameter benötigen, werden mit einer bestimmten Instanz dieses Objekttyps ausgeführt.

Finder-Objekte, die von Plug-Ins zur Orchestrator-JavaScript-API hinzugefügt werden, enthalten den Plug-In-Namen als Präfix. Der verwaltete Objekttyp `VirtualMachine` aus der vCenter Server-API wird beispielsweise in Orchestrator als JavaScript-Typ `VC:VirtualMachine` angezeigt.

Um beispielsweise über das vCenter Server-Plug-In auf eine bestimmte VC:VirtualMachine-Instanz zuzugreifen, implementiert Orchestrator ein Finder-Objekt, bei dem das Attribut `id` der virtuellen Maschine als deren eindeutiger Bezeichner verwendet wird. Sie können diese Objektinstanz als Attributwerte an Workflowelemente übergeben.

Ein Orchestrator-Plug-In ordnet die Objekte aus der integrierten Technologie äquivalenten Orchestrator-Finder-Objekten zu, die in den `<finder>`-Elementen in der Datei `vso.xml` enthalten sind. Die `<finder>`-Elemente identifizieren die Methode oder Funktion aus der integrierten Technologie, die den eindeutigen Bezeichner für eine bestimmte Instanz eines Objekts abrufen. Außerdem definieren die `<finder>`-Elemente Beziehungen zwischen Objekten, um Objekte anhand der Art ihrer Beziehungen zu anderen Objekten zu finden.

Finder-Objekte werden in der Orchestrator-Registerkarte **Bestandsliste** unter dem Plug-In angezeigt, das sie enthält.

## Rolle von Skriptobjekten

Skriptobjekte sind JavaScript-Darstellungen von Objekten aus der integrierten Technologie. Skriptobjekte aus Plug-Ins werden in der Orchestrator-JavaScript-API angezeigt. Sie können sie in Elementen mit Skript in Workflows und Aktionen verwenden.

Skriptobjekte aus Plug-Ins werden in der Orchestrator-JavaScript-API als JavaScript-Module, -Typen und -Klassen angezeigt. Die meisten Finder-Objekte verfügen über eine Skriptobjekt-Darstellung. Die JavaScript-Klassen können der Orchestrator-JavaScript-API Methoden und Attribute hinzufügen, die die Methoden und Attribute für Objekte aus der API der integrierten Technologie darstellen. Die integrierte Technologie stellt die Implementierungen der Objekte, Typen, Klassen, Attribute und Methoden unabhängig von Orchestrator bereit. Das vCenter Server-Plug-In stellt beispielsweise alle Objekte aus der vCenter Server-API als JavaScript-Objekte in der Orchestrator-JavaScript-API dar, mit JavaScript-Darstellungen aller durch die vCenter Server-API definierten Klassen, Methoden und Attribute. Sie können die Skripterstellungsklassen von vCenter Server und die durch sie definierten Methoden und Attribute in Orchestrator-Funktionen mit Skript verwenden.

Der verwaltete Objekttyp `VirtualMachine` aus der vCenter Server-API wird beispielsweise vom Finder `VC:VirtualMachine` gefunden und in der Orchestrator-JavaScript-API als JavaScript-Klasse `VcVirtualMachine` angezeigt. Die JavaScript-Klasse `VcVirtualMachine` in der Orchestrator-JavaScript-API definiert genau die gleichen Methoden und Attribute wie das verwaltete Objekt `VirtualMachine` aus der vCenter Server-API.

Ein Orchestrator-Plug-In ordnet die Objekte, Typen, Klassen, Attribute und Methoden aus der integrierten Technologie äquivalenten Orchestrator-JavaScript-Objekten, -Typen, -Klassen, -Attributen und -Methoden zu, die im Element `<scripting-objects>` in der Datei `vso.xml` enthalten sind.

## Rolle von Ereignishandlern

Ereignisse sind Änderungen an Zuständen oder Attributen der Objekte, die Orchestrator in der Plug-In-Technologie findet. Orchestrator überwacht Ereignisse durch Implementieren von Ereignishandlern.

Mithilfe von Orchestrator-Plug-Ins können Sie Ereignisse in einer Plug-In-Technologie auf verschiedene Arten überwachen. Mithilfe der Orchestrator-Plug-In-API können Sie Ereignishandler der folgenden Typen erstellen, um Ereignisse in einer integrierten Technologie zu überwachen.

### Listener

Überwachen Objekte in der integrierten Technologie passiv auf Änderungen an ihrem Zustand. Durch die integrierte Technologie oder die Plug-In-Implementierung werden die Ereignisse definiert, die Listener überwachen. Listener initiieren keine Ereignisse, sondern benachrichtigen Orchestrator, wenn die Ereignisse auftreten. Listener erkennen Ereignisse entweder durch Abfragen der integrierten Technologie oder durch Benachrichtigungen von der in-



tegrierten Technologie. Wenn Ereignisse auftreten, können Orchestrator-Richtlinien oder -Workflows, die auf das betreffende Ereignis warten, darauf reagieren, indem sie Vorgänge im Orchestrator-Server starten. Listener-Komponenten sind optional.

### Richtlinien

Überwachen bestimmte Ereignisse in der integrierten Technologie und starten Vorgänge im Orchestrator-Server, wenn die Ereignisse auftreten. Richtlinien können Richtlinienauslöser und Richtlinienkontrollen überwachen. Richtlinienauslöser definieren ein Ereignis in der integrierten Technologie, dessen Auftreten dazu führt, dass eine laufende Richtlinie einen Vorgang im Orchestrator-Server startet, z. B. die Ausführung eines Workflows. Richtlinienkontrollen definieren Wertebereiche für die Attribute eines Objekts in der integrierten Technologie, deren Überschreitung dazu führt, dass Orchestrator einen Vorgang startet. Richtlinien sind optional.

### Workflowauslöser

Wenn ein laufender Workflow ein Warteeignis-Element enthält, wird die Ausführung bei Erreichen dieses Elements angehalten und der Workflow wartet darauf, dass ein Ereignis in der integrierten Technologie auftritt. Workflowauslöser definieren die Ereignisse in der integrierten Technologie, auf die bei Warteeignis-Elementen in Workflows gewartet wird. Sie registrieren Workflowauslöser bei Watchern. Workflowauslöser sind optional.

### Watcher

Überwachen Workflowauslöser im Auftrag eines Warteeignis-Elements in einem Workflow auf ein bestimmtes Ereignis in der integrierten Technologie. Bei Auftreten des Ereignisses benachrichtigen die Watcher alle Workflows, die auf dieses Ereignis warten. Watcher sind optional.

## Inhalt und Struktur von Plug-Ins

Orchestrator-Plug-Ins müssen einen Standardsatz von Komponenten enthalten und eine Standarddateistruktur aufweisen. Damit ein Plug-In der Standarddateistruktur entspricht, muss es bestimmte Ordner und Dateien umfassen.

Zum Erstellen eines Orchestrator-Plug-Ins definieren Sie, wie Orchestrator auf Objekte der entsprechenden Plug-In-Technologie zugreift und mit diesen interagiert. Und Sie ordnen diese Objekte und Funktionen der Plug-In-Technologie in der Datei `vso.xml` den entsprechenden Objekten und Funktionen von Orchestrator zu.

Die Datei `vso.xml` muss Verweise auf alle Objekttypen und Vorgänge enthalten, die in Orchestrator genutzt werden. Jedes Objekt, das vom Plug-In in der Plug-In-Technologie gefunden wird, muss eine eindeutige, von Ihnen zugewiesene Kennung besitzen. Sie definieren die Objektnamen in den `finder`-Elementen und in den Objektelelementen in der Datei `vso.xml`.

Ein Plug-In kann als Standard-Java-Archivdatei (JAR) oder als ZIP-Datei bereitgestellt werden, muss jedoch stets unter Verwendung der Dateierweiterung `.dar` umbenannt werden.

---

**HINWEIS** Über das Orchestrator Control Center können Sie DAR-Dateien auf den Orchestrator Server importieren.

---

- [Definieren der Anwendungszuweisung in der Datei `vso.xml`](#) auf Seite 234

Objekte in der Datei `vso.xml` erscheinen als Skriptobjekte in der Orchestrator-Skript-API oder als Finder-Objekte auf der Orchestrator-Registerkarte **Bestandsliste**.

- [Format der Plug-In-Definitionsdatei `vso.xml`](#) auf Seite 234

Die Datei `vso.xml` definiert, wie der Orchestrator-Server mit der integrierten Technologie interagiert. Sie müssen in die Datei `vso.xml` einen Verweis auf alle Objekttypen und Vorgänge für Orchestrator einschließen.

- [Benennen von Plug-In-Objekten](#) auf Seite 235

Jedem Objekt, das vom Plug-In in der Plug-In-Technologie gefunden wird, muss von Ihnen eine eindeutige Kennung zugewiesen werden. Sie definieren die Objektnamen in den <finder>-Elementen und in den <object>-Elementen in der Datei `vso.xml`.

- [Benennungskonventionen für Plug-In-Objekte](#) auf Seite 236

Sie müssen die Benennungskonventionen der Java-Klasse beachten, wenn Sie Objekte in Plug-Ins benennen.

- [Dateistruktur des Plug-Ins](#) auf Seite 237

Ein Plug-In muss mit der Standarddateistruktur kompatibel sein und bestimmte spezifische Ordner und Dateien enthalten. Sie stellen ein Plug-In als Standard-Java-Archiv (JAR) oder als ZIP-Datei bereit, die Sie auf die Dateierweiterung `.dar` umbenennen müssen.

## Definieren der Anwendungszuweisung in der Datei `vso.xml`

Objekte in der Datei `vso.xml` erscheinen als Skriptobjekte in der Orchestrator-Skript-API oder als Finder-Objekte auf der Orchestrator-Registerkarte **Bestandsliste**.

Die `vso.xml`-Datei gibt dem Orchestrator-Server die folgenden Informationen an:

- Version, Name und Beschreibung für das Plug-In
- Verweist auf die Klassen der integrierten Technologie und den verknüpften Plug-In-Adapter
- Initialisiert das Plug-In beim Start des Orchestrator-Servers
- Skripterstellungstypen, die die verschiedenen Objekttypen in der integrierten Technologie darstellen
- Beziehungen zwischen Objekttypen, um zu definieren, wie die Objekte in der Orchestrator-Bestandsliste angezeigt werden
- Skripterstellungsklassen, die die Objekte und Vorgänge in der integrierten Technologie Funktionen und Objekttypen in der Orchestrator-JavaScript-API zuweisen
- Enumerationen, um eine Liste konstanter Werte zu definieren, die für alle Objekte eines bestimmten Typs gelten
- Ereignisse, die Orchestrator in der integrierten Technologie überwacht

Die Datei `vso.xml` muss der XML-Schemadefinition von Orchestrator-Plug-Ins entsprechen. Diese können Sie auf der VMware-Support-Website aufrufen.

<http://www.vmware.com/support/orchestrator/plugin-4-1.xsd>

Beschreibungen aller Elemente der Datei `vso.xml` finden Sie unter „[Elemente der Plug-In-Definitionsdatei vso.xml](#)“, auf Seite 72.

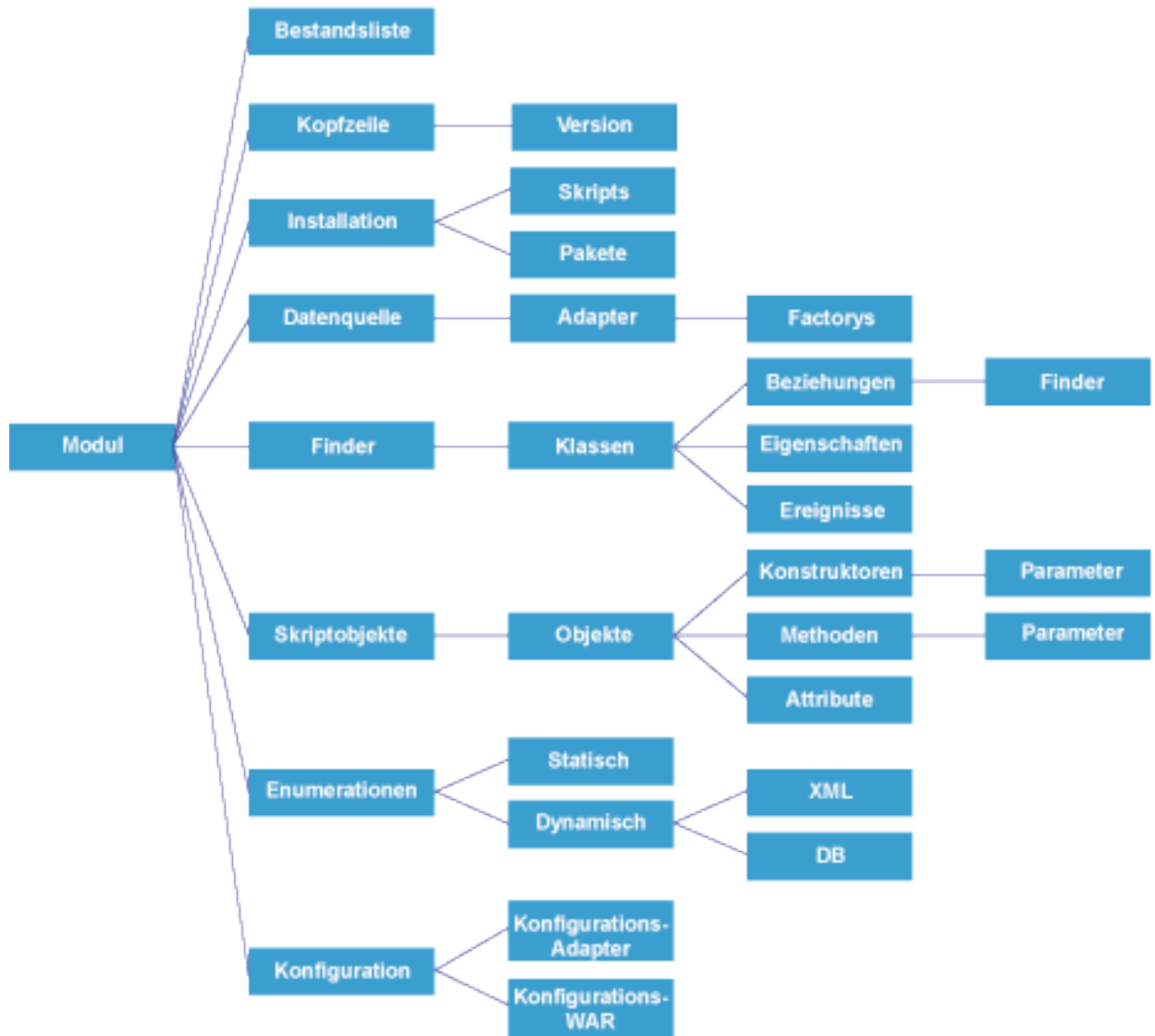
## Format der Plug-In-Definitionsdatei `vso.xml`

Die Datei `vso.xml` definiert, wie der Orchestrator-Server mit der integrierten Technologie interagiert. Sie müssen in die Datei `vso.xml` einen Verweis auf alle Objekttypen und Vorgänge für Orchestrator einschließen.

Objekte in der Datei `vso.xml` erscheinen als Skriptobjekte in der Orchestrator-Skript-API oder als Finder-Objekte auf der Orchestrator-Registerkarte **Bestandsliste**.

Als Teil der offenen Architektur und standardisierten Implementierung von Plug-Ins muss die Datei `vso.xml` einem Standardformat entsprechen.

Das folgende Diagramm zeigt das Format der Plug-In-Definitionsdatei `vso.xml` und wie die Elemente ineinander verschachtelt sind.

**Abbildung 6-2.** Format der Plug-In-Definitionsdatei vso.xml

## Benennen von Plug-In-Objekten

Jedem Objekt, das vom Plug-In in der Plug-In-Technologie gefunden wird, muss von Ihnen eine eindeutige Kennung zugewiesen werden. Sie definieren die Objektnamen in den <finder>-Elementen und in den <object>-Elementen in der Datei vso.xml.

Die Finder-Vorgänge, die Sie in der Factory-Implementierung definieren, finden Objekte in der Plug-In-Technologie. Wenn das Plug-In Objekte findet, können Sie sie in Orchestrator-Workflows verwenden und von einem Workflowelement zu einem anderen weiterreichen. Die eindeutigen Bezeichner, die Sie für die Objekte bereitstellen, ermöglichen es Ihnen, zu den Elementen in einem Workflow zu wandern.

Der Orchestrator-Server speichert nur den Typ und den Bezeichner jedes von ihm verarbeiteten Objekts. Informationen über die Herkunft oder die Art, wie Orchestrator das Objekt erhalten hat, werden nicht gespeichert. Sie müssen Objekte in der Plug-In-Implementierung konsistent benennen, damit Sie die Objekte verfolgen können, die Sie aus den Plug-Ins erhalten.

Wenn der Orchestrator-Server stoppt, während Workflows ausgeführt werden, werden die Workflows beim Neustart des Servers an dem Workflowelement wieder aufgenommen, das beim Stoppen des Servers aktiv war. Der Workflow nutzt die Bezeichner, um Objekte abzurufen, die das Element verarbeitet hat, als der Server gestoppt wurde.

## Benennungskonventionen für Plug-In-Objekte

Sie müssen die Benennungskonventionen der Java-Klasse beachten, wenn Sie Objekte in Plug-Ins benennen.

**WICHTIG** Aufgrund der Art und Weise, wie die Workflow-Engine die Daten serialisiert, dürfen Sie nicht die folgenden Zeichenfolgen in Objektnamen verwenden. Das Verwenden dieser Zeichenfolgen in Objektnamen führt dazu, dass die Workflow-Engine Workflows falsch analysiert, wodurch unerwartetes Verhalten bei der Workflowausführung auftreten kann.

- #;#
- #,#
- #=#

Verwenden Sie diese Richtlinien, wenn Sie Objekte in Plug-Ins benennen.

- Verwenden Sie einen Anfangsgroßbuchstaben für jedes Wort im Namen.
- Verwenden Sie keine Leerzeichen zum Trennen von Wörtern.
- Verwenden Sie bei Buchstaben nur die Standardzeichen A bis Z und a bis z.
- Verwenden Sie keine Sonderzeichen wie Akzente.
- Verwenden Sie keine Zahl als erstes Zeichen eines Namen.
- Verwenden Sie nach Möglichkeit weniger als 10 Zeichen.

In [Tabelle 6-1](#) sind die Regeln aufgeführt, die für die einzelnen Objekttypen gelten.

**Tabelle 6-1.** Benennungsregeln für Plug-In-Objekte

Objekttyp	Benennungsregeln
Plug-In	<ul style="list-style-type: none"> <li>■ Definiert im Element <code>&lt;module&gt;</code> in der Datei <code>vso.xml</code>.</li> <li>■ Muss den Java-Klassen-Benennungskonventionen entsprechen.</li> <li>■ Muss eindeutig sein. Sie können nicht zwei Plug-Ins mit demselben Namen im Orchestrator-Server ausführen.</li> </ul>
Finder-Objekt	<ul style="list-style-type: none"> <li>■ Definiert in den Elementen <code>&lt;finder&gt;</code> in der Datei <code>vso.xml</code>.</li> <li>■ Muss den Java-Klassen-Benennungskonventionen entsprechen.</li> <li>■ Muss im Plug-In eindeutig sein.</li> </ul> <p>Orchestrator fügt Finder-Objektnamen in den Finder-Objekttypen in der Orchestrator-Skript-API den Plug-In-Namen und einen Doppelpunkt hinzu. Beispiel: Der Objekttyp <code>VirtualMachine</code> aus dem vCenter Server-Plug-In wird in der Orchestrator-Skript-API als <code>VC:VirtualMachine</code> angezeigt.</p>
Skriptobjekt	<ul style="list-style-type: none"> <li>■ Definiert in den Elementen <code>&lt;scripting-object&gt;</code> in der Datei <code>vso.xml</code>.</li> <li>■ Muss den Java-Klassen-Benennungskonventionen entsprechen.</li> <li>■ Muss im Orchestrator-Server eindeutig sein.</li> <li>■ Fügen Sie dem Skriptobjektnamen immer ein Präfix mit dem Namen des Plug-Ins hinzu, um eine Verwechslung von Skriptobjekten mit Finder-Objekten mit demselben Namen oder mit Skriptobjekten aus anderen Plug-Ins zu vermeiden. Fügen Sie aber keinen Doppelpunkt hinzu. Beispiel: Die Klasse <code>VirtualMachine</code> aus dem vCenter Server-Plug-In wird in der Orchestrator-Skript-API als Klasse <code>VcVirtualMachine</code> angezeigt.</li> </ul>

## Dateistruktur des Plug-Ins

Ein Plug-In muss mit der Standarddateistruktur kompatibel sein und bestimmte spezifische Ordner und Dateien enthalten. Sie stellen ein Plug-In als Standard-Java-Archiv (JAR) oder als ZIP-Datei bereit, die Sie auf die Dateierweiterung `.dar` umbenennen müssen.

Die Inhalte des DAR-Archivs müssen die folgende Ordnerstruktur und die folgenden Namenskonventionen verwenden.

**Tabelle 6-2.** Struktur des DAR-Archivs

Ordner	Beschreibung
<code>plug-in_name\VS0-INF\</code>	Enthält die <code>vso.xml</code> -Datei, die die Zuordnung der Objekte in der Plug-In-Technologie zu Orchestrator-Objekten definiert. Der <code>VS0-INF</code> -Ordner und die <code>vso.xml</code> -Datei sind vorgeschrieben.
<code>plug-in_name\lib\</code>	Enthält die JAR-Dateien, in denen die Binärdateien der Plug-In-Technologie zusammengefasst sind. Der Ordner enthält auch JAR-Dateien, die die Implementierungen von Adapter, Factory, Benachrichtigungshandles und anderen Schnittstellen im Plug-In umfassen. Der <code>lib</code> -Ordner und die JAR-Dateien sind vorgeschrieben.
<code>plug-in_name\resources\</code>	Enthält Ressourcendateien, die das Plug-In erfordert. Der Ordner <code>resources</code> kann folgende Elementtypen enthalten: <ul style="list-style-type: none"> <li>■ Bilddateien zur Darstellung der Objekte des Plug-Ins auf der Registerkarte <b>Bestand</b> in Orchestrator</li> <li>■ Skripte zur Definition des Initialisierungsverhaltens beim Start des Plug-Ins</li> <li>■ Orchestrator-Pakete, die benutzerdefinierte Workflows, Aktionen und andere Ressourcen enthalten können, die mit den Objekten interagieren, auf die Sie über die Plug-Ins zugreifen</li> </ul> Sie können Ressourcen in Unterordnern organisieren. Beispiele dafür: <code>resources\images\</code> , <code>resources\scripts\</code> oder <code>resources\packages\</code> . Der Ordner <code>resources</code> ist optional.

Über das Orchestrator Control Center können Sie DAR-Dateien auf den Orchestrator Server importieren.

## Orchestrator Plug-In-API-Referenz

Die Orchestrator-Plug-In-API definiert Java-Schnittstellen und -Klassen, um sie zu implementieren und zu erweitern, wenn Sie die `IPluginAdaptor`- und `IPluginFactory`-Implementierungen entwickeln, um ein Plug-In zu erstellen.

Alle Klassen sind im Paket `ch.dunes.vso.sdk.api` enthalten, sofern nicht etwas anderes festgelegt ist.

### IAop-Schnittstelle

Die `IAop`-Schnittstelle bietet Methoden zum Abrufen und Einrichten von Eigenschaften von Objekten in Plug-In-Anwendungen.

```
public interface IAop
```

Die `IAop`-Schnittstelle definiert die folgenden Methoden:

Methode	Gibt Folgendes zurück	Beschreibung
<code>get(java.lang.String propertyName, java.lang.Object object, java.lang.Object sdkObject)</code>	<code>java.lang.Object</code>	Ruft eine Eigenschaft eines bestimmten Objekts des Plug-Ins ab.
<code>set(java.lang.String propertyName, java.lang.String propertyValue, java.lang.Object object)</code>	<code>Void</code>	Richtet eine Eigenschaft für ein bestimmtes Objekt des Plug-Ins ein.

## IDynamicFinder-Schnittstelle

Die IDynamicFinder-Schnittstelle gibt die ID und die Eigenschaften eines Finders programmgesteuert zurück, anstatt die ID und die Eigenschaften in der Datei `vso.xml` zu definieren.

Die IDynamicFinder-Schnittstelle definiert die folgenden Methoden.

Methode	Gibt Folgendes zurück	Beschreibung
<code>getIdAccessor(java.lang.String type)</code>	<code>java.lang.String</code>	Stellt einen OGNL-Ausdruck bereit, um programmgesteuert eine Objekt-ID zu erhalten.
<code>getProperties(java.lang.String type)</code>	<code>java.util.List&lt;SDKFinderProperty&gt;</code>	Stellt programmgesteuert eine Liste von Objekteigenschaften bereit.

## IPluginAdaptor-Schnittstelle

Sie implementieren die Schnittstelle IPluginAdaptor, um Plug-In-Factorys, Ereignisse und Watcher zu verwalten. Die Schnittstelle IPluginAdaptor definiert einen Adapter zwischen einem Plug-In und dem Orchestrator-Server.

IPluginAdaptor-Instanzen sind für die Sitzungsverwaltung zuständig. Die Schnittstelle IPluginAdaptor definiert die folgenden Methoden.

Methode	Gibt Folgendes zurück	Beschreibung
<code>addWatcher(PluginWatcher watcher)</code>	<code>Void</code>	Fügt einen Watcher zur Überwachung eines bestimmten Ereignisses hinzu
<code>createPluginFactory(java.lang.String sessionId, java.lang.String username, java.lang.String password, IPluginNotificationHandler notificationHandler)</code>	<code>IPluginFactory</code>	Erstellt eine IPluginFactory-Instanz. Der Orchestrator-Server verwendet die Factory, um Objekte von der durch Plug-In-Technologie über ihre ID, über ihre Beziehung zu anderen Objekten usw. anzufordern.  Die Sitzungs-ID ermöglicht es Ihnen, eine laufende Sitzung zu identifizieren. Beispiel: Ein Benutzer kann sich bei zwei verschiedenen Orchestrator-Clients anmelden und gleichzeitig zwei Sitzungen ausführen.  Durch den Start eines Workflows wird eine Sitzung erstellt, die von dem Client unabhängig ist, auf dem der Workflow gestartet wurde. Ein Workflow wird weiter ausgeführt, auch wenn Sie den Orchestrator-Client schließen.
<code>installLicenses(PluginLicense[] licenses)</code>	<code>Void</code>	Installiert die Lizenzinformationen für Standard-Plug-Ins, die von VMware bereitgestellt werden

Methoden	Gibt Folgendes zurück	Beschreibung
<code>registerEventPublisher(java.lang.String type, java.lang.String id, IPluginEventPublisher publisher)</code>	Void	Setzt Auslöser und Kontrollen für ein Element im Bestand
<code>removeWatcher(java.lang.String watcherId)</code>	Void	Entfernt einen Watcher
<code>setPluginName(java.lang.String pluginName)</code>	Void	Ruft den Plug-In-Namen aus der <code>vso.xml</code> -Datei ab
<code>setPluginPublisher(IPluginPublisher pluginPublisher)</code>	Void	Legt den Herausgeber des Plug-Ins fest
<code>uninstallPluginFactory(IPluginFactory plugin)</code>	Void	Deinstalliert eine Plug-In-Factory
<code>unregisterEventPublisher(java.lang.String type, java.lang.String id, IPluginEventPublisher publisher)</code>	Void	Entfernt Auslöser und Kontrollen von einem Element im Bestand

## IPluginEventPublisher-Schnittstelle

Die Schnittstelle `IPluginEventPublisher` veröffentlicht Kontrollen und Auslöser auf einem Ereignisbenachrichtigungsbuss zur Überwachung von Orchestrator-Richtlinien.

Sie können `IPluginEventPublisher`-Instanzen direkt in der Plug-In-Adapterimplementierung oder in getrennten Klassen zur Ereignisgenerierung erstellen.

Sie können die `IPluginEventPublisher`-Schnittstelle verwenden, um Ereignisse in der integrierten Technologie für die Orchestrator-Richtlinien-Engine zu veröffentlichen. Sie erstellen Methoden, um Richtlinien auslöser und -kontrollen für Objekte in integrierter Technologie und Ereignislistener festzulegen, um Ereignisse in diesen Objekten zu überwachen.

Richtlinien können entweder Anzeigen oder Auslöser implementieren, um Objekte in der integrierten Technologie zu überwachen. Richtlinienkontrollen überwachen die Attribute von Objekten und übertragen mithilfe von Push ein Ereignis auf den Orchestrator-Server, wenn die Werte der Objekte bestimmte Grenzwerte überschreiten. Richtlinien auslöser überwachen Objekte und übertragen mithilfe von Push ein Ereignis auf den Orchestrator-Server, wenn ein definiertes Ereignis für das Objekt eintritt. Sie registrieren Richtlinienkontrollen und -auslöser bei `IPluginEventPublisher`-Instanzen, damit Orchestrator-Richtlinien diese überwachen können.

Die Schnittstelle `IPluginEventPublisher` definiert die folgenden Methoden.

Typ	Gibt Folgendes zurück	Beschreibung
<code>pushGauge(java.lang.String type, java.lang.String id, java.lang.String gaugeName, java.lang.String deviceName, java.lang.Double gaugeValue)</code>	Void	Veröffentlicht eine Kontrolle für zu überwachende Richtlinien. Übernimmt die folgenden Parameter: <ul style="list-style-type: none"> <li>■ <code>type</code>: Typ des zu überwachenden Objekts</li> <li>■ <code>id</code>: Bezeichner des zu überwachenden Objekts</li> <li>■ <code>gaugeName</code>: Name für diese Kontrolle</li> <li>■ <code>deviceName</code>: Name für den Attributtyp, der von der Kontrolle überwacht wird</li> <li>■ <code>gaugeValue</code>: Wert, auf den die Kontrolle das Objekt überwacht</li> </ul>
<code>pushTrigger(java.lang.String type, java.lang.String id, java.lang.String triggerName, java.util.Properties additionalProperties)</code>	Void	Veröffentlicht einen Auslöser für zu überwachende Richtlinien. Übernimmt die folgenden Parameter: <ul style="list-style-type: none"> <li>■ <code>type</code>: Typ des zu überwachenden Objekts</li> <li>■ <code>id</code>: Bezeichner des zu überwachenden Objekts</li> <li>■ <code>triggerName</code>: Name für diesen Auslöser</li> <li>■ <code>additionalProperties</code>: Gegebenfalls zusätzliche Eigenschaften für den Auslöser zur Überwachung</li> </ul>

## IPluginFactory-Schnittstelle

IPluginAdaptor gibt IPluginFactory-Instanzen zurück. IPluginFactory-Instanzen führen Befehle in der als Plug-In integrierten Anwendung aus und finden Objekte, mit denen Orchestrator-Vorgänge ausgeführt werden sollen.

Die Schnittstelle IPluginFactory definiert das folgende Feld:

```
static final java.lang.String RELATION_CHILDREN
```

Die Schnittstelle IPluginFactory definiert die folgenden Methoden.

Methode	Gibt Folgendes zurück	Beschreibung
<code>executePluginCommand(java.lang.String cmd)</code>	Void	Plug-In verwenden, um einen Befehl auszuführen. VMware empfiehlt, diese Methode nicht zu verwenden.
<code>find(java.lang.String type, java.lang.String id)</code>	java.lang.Object	Plug-In verwenden, um ein Objekt zu finden. Das Objekt wird nach ID und Typ identifiziert.
<code>findAll(java.lang.String type, java.lang.String query)</code>	QueryResult	Plug-In verwenden, um Objekte eines bestimmten Typs zu finden, die mit einer Abfragezeichenfolge übereinstimmen. Sie definieren die Syntax der Abfrage in der IPluginFactory-Implementierung des Plug-Ins. Wenn Sie die Abfragesyntax nicht definieren, gibt <code>findAll()</code> alle Objekte des angegebenen Typs zurück.



Methode	Gibt Folgendes zurück	Beschreibung
<code>findRelation(java.lang.String parentType, java.lang.String parentId, java.lang.String relationName)</code>	<code>java.util.List</code>	Ermittelt, ob ein Objekt untergeordnete Objekte hat.
<code>hasChildrenInRelation(java.lang.String parentType, java.lang.String parentId, java.lang.String relationName)</code>	<code>HasChildrenResult</code>	Findet alle untergeordneten Objekte, die eine bestimmte Beziehung mit einem bestimmten übergeordneten Objekt haben.
<code>invalidate(java.lang.String type, java.lang.String id)</code>	<code>Void</code>	Objekte nach Typ und ID als nicht gültig ermitteln.
<code>void invalidateAll()</code>	<code>Void</code>	Alle Objekte im Cache als nicht gültig ermitteln.

## IPluginNotificationHandler-Schnittstelle

`IPluginNotificationHandler` definiert Methoden, um Orchestrator über verschiedene Arten von Ereignissen zu benachrichtigen, die bei den Objekten vorkommen, auf die Orchestrator über das Plug-In zugreift.

Die Schnittstelle `IPluginNotificationHandler` definiert die folgenden Methoden.

Methode	Gibt Folgendes zurück	Beschreibung
<code>getSessionID()</code>	<code>java.lang.String</code>	Gibt die aktuelle Sitzungs-ID zurück.
<code>notifyElementDeleted(java.lang.String type, java.lang.String id)</code>	<code>Void</code>	Benachrichtigt das System, dass ein Objekt mit dem angegebenen Typ und der angegebenen ID gelöscht wurde.
<code>notifyElementInvalidate(java.lang.String type, java.lang.String id)</code>	<code>Void</code>	Benachrichtigt das System, dass sich die Beziehungen eines Objekts geändert haben. Sie können die <code>notifyElementInvalidate()</code> -Methode verwenden, um Orchestrator über alle Veränderungen in Beziehungen zwischen Objekten zu benachrichtigen, nicht nur über Änderungen von Beziehungen, die ein Objekt ungültig machen. Das Hinzufügen eines untergeordneten Objekts zu einem übergeordneten Objekt stellt beispielsweise eine Änderung in der Beziehung zwischen den beiden Objekten dar.
<code>notifyElementUpdated(java.lang.String type, java.lang.String id)</code>	<code>Void</code>	Benachrichtigt das System, dass die Attribute eines Objekts geändert wurden.
<code>notifyMessage(ch.dun.es.vso.sdk.api.ErrorLevel severity, java.lang.String type, java.lang.String id, java.lang.String message)</code>	<code>Void</code>	Veröffentlicht eine Fehlermeldung in Bezug auf das aktuelle Modul.

## IPluginPublisher-Schnittstelle

Die Schnittstelle IPluginPublisher veröffentlicht ein Watcher-Ereignis auf einem Ereignisbenachrichtigungsbus für „Warteereignis“-Elemente bei länger ausführenden Workflows, die überwacht werden sollen.

Wenn ein Workflowauslöser ein Ereignis in der integrierten Technologie startet, benachrichtigt ein Plug-In-Watcher, der diesen Auslöser überwacht und der bei einer IPluginPublisher-Instanz registriert ist, alle wartenden Workflows darüber, dass das Ereignis eingetreten ist.

Die Schnittstelle IPluginPublisher definiert die folgende Methode.

Typ	Wert	Beschreibung
pushWatcherEvent( <code>java.lang.String id, java.util.Properties properties</code> )	Void	Veröffentlicht ein Watcher-Ereignis auf dem Ereignisbenachrichtigungsbus

## WebConfigurationAdaptor-Schnittstelle

Die WebConfigurationAdaptor-Schnittstelle implementiert IConfigurationAdaptor und definiert Methoden zum Finden und Installieren einer Webanwendung auf der Konfigurationsregisterkarte für ein Plug-In.

**HINWEIS** Die WebConfigurationAdaptor-Schnittstelle ist seit Orchestrator 4.1 veraltet. Um der Konfiguration eine Webanwendung hinzuzufügen, implementieren Sie IConfigurationAdaptor und verwenden Sie das Attribut `configuration-war` in der Datei `vso.xml` zum Identifizieren der Webanwendung.

Die WebConfigurationAdaptor-Schnittstelle definiert die folgenden Methoden.

Methode	Gibt Folgendes zurück	Beschreibung
getWebAppContext()	String	Findet die WAR-Datei der Webanwendung für die Konfigurationsregisterkarte. Geben Sie den Namen und Pfad zur WAR-Datei aus dem <code>/webapps</code> -Verzeichnis in der DAR-Datei als Zeichenfolge ein.
setWebConfiguration( <code>boolean webConfiguration</code> )	Boolean	Legen Sie fest, ob die Inhalte der Konfigurationsregisterkarte durch eine Webanwendung definiert werden.

## PluginTrigger-Klasse

Die Klasse PluginTrigger erstellt ein Auslösermodul, das Informationen über Objekte und Ereignisse erhält, die in der Plug-In-Technologie für ein „Warteereignis“-Element in einem Workflow zu überwachen sind.

Die PluginTrigger-Klasse definiert Methoden zum Abrufen oder Festlegen des Typs und Namens des zu überwachenden Objekts, der Art des Ereignisses und einer Zeitüberschreitungsdauer.

Sie erstellen Implementierungen der PluginTrigger-Klasse für die ausschließliche Verwendung von Warteereigniselementen in Workflows. Sie definieren Richtlinienauslöser für Orchestrator-Richtlinien in Klassen, die Ereignisse definieren und die IPluginEventPublisher.pushTrigger()-Methode implementieren.

```
public class PluginTrigger
extends java.lang.Object
implements java.io.Serializable
```

Die Klasse PluginTrigger definiert die folgenden Methoden.

Methode	Gibt Folgendes zurück	Beschreibung
<code>getModuleName()</code>	<code>java.lang.String</code>	Bezieht den Namen des Auslösermoduls.
<code>getProperties()</code>	<code>java.util.Properties</code>	Bezieht eine Liste von Eigenschaften für den Auslöser.
<code>getSdkId()</code>	<code>java.lang.String</code>	Bezieht den Bezeichner des Objekts, das in der Plug-In-Technologie überwacht werden soll.
<code>getSdkType()</code>	<code>java.lang.String</code>	Bezieht den Typ des Objekts, das in der Plug-In-Technologie überwacht werden soll.
<code>getTimeout()</code>	<code>Long</code>	Bezieht die Zeitüberschreitungsperiode für den Auslöser.
<code>setModuleName(java.lang.String moduleName)</code>	<code>Void</code>	Legt den Namen des Triggermoduls fest
<code>setProperties(java.util.Properties properties)</code>	<code>Void</code>	Legt eine Liste von Eigenschaften für den Auslöser fest.
<code>setSdkId(java.lang.String sdkId)</code>	<code>Void</code>	Legt den Bezeichner des Objekts fest, das in der Plug-In-Technologie überwacht werden soll.
<code>setSdkType(java.lang.String sdkType)</code>	<code>Void</code>	Legt den Typ des Objekts fest, das in der Plug-In-Technologie überwacht werden soll.
<code>setTimeout(long timeout)</code>	<code>Void</code>	Legt eine Zeitüberschreitungsperiode in Sekunden fest. Ein negativer Wert deaktiviert die Zeitüberschreitung.

## Konstruktoren

- `PluginTrigger()`
- `PluginTrigger(java.lang.String moduleName, long timeout, java.lang.String sdkType, java.lang.String sdkId)`

## PluginWatcher-Klasse

Die Klasse `PluginWatcher` überwacht ein Auslösermodul auf ein definiertes Ereignis in der Plug-In-Technologie für ein Warteereignis-Element eines Workflows mit langer Ausführungszeit.

Die `PluginWatcher`-Klasse definiert einen Konstruktor, den Sie verwenden können, um Plug-In-Watcher-Instanzen zu erstellen. Die `PluginWatcher`-Klasse definiert Methoden zum Abrufen oder Festlegen von Namen des zu überwachenden Workflowauslösers sowie eine Zeitüberschreitung.

```
public class PluginWatcher
extends java.lang.Object
implements java.io.Serializable
```

Die Klasse `PluginWatcher` definiert die folgenden Methoden:

Methode	Gibt Folgendes zurück	Beschreibung
<code>getId()</code>	<code>java.lang.String</code>	Bezieht die ID des Auslösers
<code>getModuleName()</code>	<code>java.lang.String</code>	Bezieht den Namen des Auslösermoduls.
<code>getTimeoutDate()</code>	<code>Long</code>	Bezieht das Zeitüberschreitungsdatum für den Auslöser

Methode	Gibt Folgendes zurück	Beschreibung
getTrigger()	Void	Bezieht einen Auslöser
setId(java.lang.String id)	Void	Legt den Bezeichner des Auslösers fest
setTimeoutDate()	Void	Legt das Zeitüberschreitungsdatum für den Auslöser fest

## Konstruktor

PluginWatcher(PluginTrigger trigger)

## QueryResult-Klasse

Die Klasse QueryResult enthält die Ergebnisse einer find-Abfrage auf Objekten, auf die Orchestrator über das Plug-In zugreift.

```
public class QueryResult
extends java.lang.Object
implements java.io.Serializable
```

Der Wert totalCount kann größer als die Anzahl von Elementen sein, die von QueryResult zurückgegeben werden, wenn die Gesamtanzahl der gefundenen Ergebnisse die Anzahl von Ergebnissen überschreitet, die die Abfrage zurückgibt. Die Anzahl von Ergebnissen, die von der Abfrage zurückgegeben werden, wird in der Abfragesyntax der vso.xml-Datei definiert.

Die Klasse QueryResult definiert die folgenden Methoden:

Methode	Gibt Folgendes zurück	Beschreibung
addElement(java.lang.Object element)	Void	Fügt QueryResult ein Element hinzu
addElements(java.util.List elements)	Void	Fügt QueryResult eine Elementliste hinzu
getElements()	java.util.List	Bezieht Elemente aus der Plug-In-Anwendung
getTotalCount()	Lang	Bezieht eine Zählung aller in der Plug-In-Technologie verfügbaren Elemente
isPartialResult()	Boolean	Ermittelt, ob das bezogene Ergebnis vollständig ist
removeElement(java.lang.Object element)	Void	Entfernt ein Element aus der Plug-In-Technologie
setElements(java.util.List elements)	Void	Setzt Elemente in der Plug-In-Technologie
setTotalCount(long totalCount)	Void	Setzt die Gesamtanzahl von verfügbaren Elementen in der Plug-In-Technologie

## Konstruktoren

- QueryResult()
- QueryResult(java.util.List ret)
- QueryResult(java.util.List elements, long totalCount)

## SDKFinderProperty-Klasse

Die Klasse SDKFinderProperty definiert Methoden, um Eigenschaften in den Objekten zu beziehen und festzulegen, die in der Plug-In-Technologie von Orchestrator-Finderobjekten gefunden werden. Die Methode IDynamicFinder.getProperties gibt SDKFinderProperty-Objekte zurück.

```
public class SDKFinderProperty
extends java.lang.Object
```

Die Klasse SDKFinderProperty definiert die folgenden Methoden:

Methoden	Gibt Folgendes zurück	Beschreibung
getAttributeName()	java.lang.String	Bezieht einen Objektattributnamen
getBeanProperty()	java.lang.String	Bezieht Eigenschaften aus einer Java Bean
getDescription()	java.lang.String	Bezieht eine Objektbeschreibung
getDisplayName()	java.lang.String	Bezieht einen Objektanzeigenamen
getPossibleResultType()	java.lang.String	Bezieht mögliche Typen von Ergebnissen, die ein Finder zurückgibt
getPropertyAccessor()	java.lang.String	Bezieht eine Zugriffsroutine auf Objekteigenschaften
getPropertyAccessorTree()	java.lang.Object	Bezieht einen Routinenbaum für den Zugriff auf Objekteigenschaften
isHidden()	Boolean	Zeigt oder verbirgt das Objekt
isShowInColumn()	Boolean	Zeigt oder verbirgt das Objekt in der Datenbankspalte
isShowInDescription()	Boolean	Zeigt oder verbirgt die Objektbeschreibung
setAttributeName(java.lang.String attributeName)	Void	Setzt einen Objektattributnamen
setBeanProperty(java.lang.String beanProperty)	Void	Setzt Eigenschaften in einer Java Bean
setDescription(java.lang.String description)	Void	Setzt eine Objektbeschreibung
setDisplayName(java.lang.String displayName)	Void	Ersetzt einen Objektanzeigenamen
setHidden(boolean hidden)	Void	Zeigt oder verbirgt das Objekt
setPossibleResultType(java.lang.String possibleResultType)	Void	Setzt mögliche Typen von Ergebnissen, die ein Finder zurückgibt
setPropertyAccessor(java.lang.String propertyAccessor)	Void	Setzt eine Zugriffsroutine auf Objekteigenschaften
setPropertyAccessorTree(java.lang.Object propertyAccessorTree)	Void	Setzt einen Routinenbaum für den Zugriff auf Objekteigenschaften
setShowInColumn(boolean showInTable)	Void	Zeigt oder verbirgt das Objekt in der Datenbankspalte
setShowInDescription(boolean showInDescription)	Void	Zeigt oder verbirgt die Objektbeschreibung

## Konstruktor

SDKFinderProperty(java.lang.String attributeName, java.lang.String displayName, java.lang.String beanProperty, java.lang.String propertyAccessor)

## PluginExecutionException-Klasse

Die Klasse PluginExecutionException gibt eine Fehlermeldung zurück, wenn das Plug-In beim Ausführen eines Vorgangs auf eine Ausnahme trifft.

```
public class PluginExecutionException
extends java.lang.Exception
implements java.io.Serializable
```

Die Klasse PluginExecutionException erbt die folgenden Methoden von class java.lang.Throwable:

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString, fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace

## Konstruktor

PluginExecutionException(java.lang.String message)

## PluginOperationException-Klasse

Die Klasse PluginOperationException verarbeitet Fehler, die bei einem Plug-In-Vorgang aufgetreten sind.

```
public class PluginOperationException
extends java.lang.RuntimeException
implements java.io.Serializable
```

Die Klasse PluginOperationException erbt die folgenden Methoden von class java.lang.Throwable:

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

## Konstruktor

PluginOperationException(java.lang.String message)

## Enumeration „HasChildrenResult“

Die Enumeration HasChildrenResult gibt an, ob ein übergeordnetes Element untergeordnete Elemente hat. Die Methode IPluginFactory.hasChildrenInRelation gibt HasChildrenResult-Objekte zurück.

```
public enum HasChildrenResult
extends java.lang.Enum<HasChildrenResult>
implements java.io.Serializable
```

Die Enumeration HasChildrenResult definiert die folgenden Konstanten:

- public static final HasChildrenResult Yes
- public static final HasChildrenResult No
- public static final HasChildrenResult Unknown

Die Enumeration HasChildrenResult definiert die folgenden Methoden:

Methode	Gibt Folgendes zurück	Beschreibung
<code>getValue()</code>	<code>int</code>	Gibt einen der folgenden Werte zurück:  <b>1</b> Übergeordnetes Element hat untergeordnete Elemente  <b>-1</b> Übergeordnetes Element hat keine untergeordneten Elemente  <b>0</b> Unbekannter oder ungültiger Parameter
<code>valueOf(java.lang.String name)</code>	<code>static HasChildrenResult</code>	Gibt einen Enumerationskonstante dieses Typs mit dem angegebenen Namen zurück. Die Zeichenfolge muss genau mit einem Bezeichner übereinstimmen, der zum Deklarieren einer Enumerationskonstante dieses Typs verwendet wird. Verwenden Sie keine Leerzeichen im Enumerationsnamen.
<code>values()</code>	<code>static HasChildrenResult[]</code>	Gibt einen Array mit den Konstanten dieses Enumerationstyps in der Reihenfolge der Deklaration zurück. Diese Methode kann Konstanten iterativ durchlaufen:  <pre>for (HasChildrenResult c : HasChildrenResult.values()) System.out.println(c);</pre>

Die Enumeration `HasChildrenResult` erbt die folgenden Methoden von class `java.lang.Enum`:

`clone`, `compareTo`, `equals`, `finalize`, `getDeclaringClass`, `hashCode`, `name`, `ordinal`, `toString`, `valueOf`

## ScriptingAttribute-Anmerkungstyp

Der Anmerkungstyp `ScriptingAttribute` versieht ein Attribut von einem Objekt in der Plug-In-Technologie mit Anmerkungen, um es bei der Skripterstellung als Eigenschaft wenden zu können.

```
@Retention(value=RUNTIME)
@Target(value={METHOD, FIELD})
public @interface ScriptingAttribute
```

Der Anmerkungstyp `ScriptingAttribute` hat folgenden Wert:

```
public abstract java.lang.String value
```

## ScriptingFunction-Anmerkungstyp

Der Anmerkungstyp `ScriptingFunction` versieht eine Methode mit einer Anmerkung, die als Eigenschaft bei der Skripterstellung verwendet wird.

```
@Retention(value=RUNTIME)
@Target(value={METHOD, CONSTRUCTOR})
public @interface ScriptingFunction
```

Der Anmerkungstyp `ScriptingFunction` hat folgenden Wert:

```
public abstract java.lang.String value
```

## ScriptingParameter-Anmerkungstyp

Der Anmerkungstyp `ScriptingParameter` versteht einen Parameter mit einer Anmerkung, die als Eigenschaft bei der Skripterstellung verwendet wird.

```
@Retention(value=RUNTIME)
@Target(value=PARAMETER)
public @interface ScriptingParameter
```

Der Anmerkungstyp `ScriptingParameter` hat folgenden Wert:

```
public abstract java.lang.String value
```

## Elemente der Plug-In-Definitionsdatei vso.xml

Die Datei `vso.xml` umfasst einen Satz an Standardelementen. Einige Elemente sind obligatorisch, andere optional. Jedes Element hat Attribute, die Werte für die Objekte und Vorgänge definieren, die Sie Orchestrator-Objekten und -Vorgängen zuordnen.

Außerdem können Elemente null oder mehr untergeordnete Elemente haben. Eine untergeordnete Element definiert das übergeordnete Element noch weiter. Das gleiche untergeordnete Element kann in mehreren übergeordneten Elementen auftreten. Das Element `description` hat beispielsweise keine untergeordneten Elemente, wird aber als untergeordnetes Element für viele übergeordneten Elemente angezeigt: `module`, `example`, `trigger`, `gauge`, `finder`, `constructor`, `method`, `object` und `enumeration`.

Jede der folgenden Elementdefinitionen listet seine über- und untergeordneten Attribute auf.

### Modulelement

Ein Modul beschreibt eine Gruppe von Plug-In-Objekten, die für Orchestrator verfügbar gemacht werden.

Das Modul enthält Informationen darüber, wie Daten aus der Plug-In-Technologie den Java-Klassen zugeordnet sind, über die Versionierung, die Bereitstellung des Moduls und die Art, wie das Plug-In im Orchestrator-Bestand erscheint.

Das Element `<module>` ist optional. Das Element `<module>` besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
<code>name</code>	Zeichenfolge	Definiert den Typ aller <code>&lt;finder&gt;</code> -Elemente im Plug-In. Erforderliches Attribut.
<code>version</code>	Zahl	Die Plug-In-Versionsnummer, die beim erneuten Laden von Paketen in einer neuen Version des Plug-Ins verwendet wird. Erforderliches Attribut.
<code>build-number</code>	Zahl	Die Plug-In-Build-Nummer, die beim erneuten Laden von Paketen in einer neuen Version des Plug-Ins verwendet wird. Erforderliches Attribut.
<code>image</code>	Bilddatei	Das Symbol, das im Orchestrator-Bestand angezeigt wird. Erforderliches Attribut.
<code>display-name</code>	Zeichenfolge	Der Name, der im Orchestrator-Bestand angezeigt wird. Optionales Attribut.
<code>interface-mapping-allowed</code>	True oder False	VMware rät von der Schnittstellenzuordnung ab. Optionales Attribut.



**Tabelle 6-3.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
Keine	<ul style="list-style-type: none"> <li>■ &lt;description&gt;</li> <li>■ &lt;installation&gt;</li> <li>■ &lt;configuration&gt;</li> <li>■ &lt;finder-datasources&gt;</li> <li>■ &lt;inventory&gt;</li> <li>■ &lt;finders&gt;</li> <li>■ &lt;scripting-objects&gt;</li> <li>■ &lt;enumerations&gt;</li> </ul>

## Element „description“

Die Elemente <description> bieten Beschreibungen der Elemente des Plug-Ins, das in der API-Explorer-Dokumentation angezeigt wird.

Der Text, der in der API-Explorer-Dokumentation erscheint, wird zwischen den Tags <description> und </description> eingefügt.

Das Element <description> ist optional. Das Element <description> besitzt keine Attribute.

**Tabelle 6-4.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<ul style="list-style-type: none"> <li>■ &lt;module&gt;</li> <li>■ &lt;example&gt;</li> <li>■ &lt;trigger&gt;</li> <li>■ &lt;gauge&gt;</li> <li>■ &lt;finder&gt;</li> <li>■ &lt;constructor&gt;</li> <li>■ &lt;method&gt;</li> <li>■ &lt;object&gt;</li> <li>■ &lt;enumeration&gt;</li> </ul>	Keine

## Element „deprecated“

Mit dem Element <deprecated> werden veraltete Objekte und Methoden in der API-Explorer-Dokumentation markiert.

Der Text, der in der API-Explorer-Dokumentation erscheint, wird zwischen den Tags <deprecated> und </deprecated> eingefügt.

Das Element <deprecated> ist optional. Das Element <deprecated> besitzt keine Attribute.

**Tabelle 6-5.** Elementhierarchie

Übergeordnete Elemente	Untergeordnete Elemente
<ul style="list-style-type: none"> <li>■ &lt;method&gt;</li> <li>■ &lt;object&gt;</li> </ul>	Keine

## URL-Element

Das Element `<url>` stellt eine URL bereit, die auf eine externe Dokumentation über ein Objekt oder eine Aufzählung verweist.

Sie geben eine URL zwischen den Tags `<url>` und `</url>` an.

Das Element `<url>` ist optional. Das Element `<url>` besitzt keine Attribute.

**Tabelle 6-6.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<ul style="list-style-type: none"> <li>■ <code>&lt;enumeration&gt;</code></li> <li>■ <code>&lt;object&gt;</code></li> </ul>	Keine

## installation-Element

Das Element `<installation>` ermöglicht Ihnen die Installation eines Pakets oder das Ausführen eines Skripts, wenn der Server startet.

Das Element `<installation>` ist optional. Das Element `<installation>` besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
mode	always, never oder version	<p>Wenn der mode-Wert aktiv ist, wird das folgende Verhalten bewirkt, sobald der Orchestrator-Server startet:</p> <ul style="list-style-type: none"> <li>■ Die Aktion <code>always</code> wird ausgeführt</li> <li>■ Die Aktion <code>never</code> wird ausgeführt</li> <li>■ Die Aktion wird ausgeführt, wenn der Server eine neuere Version des Plug-Ins entdeckt</li> </ul> <p>Erforderliches Attribut.</p>

**Tabelle 6-7.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<code>&lt;module&gt;</code>	<code>&lt;action&gt;</code>

## Element „action“

Das Element `<action>` legt die Aktion fest, die beim Start des Orchestrator-Servers ausgeführt wird.

Die Attribute für das Element `<action>` geben den Pfad zum Orchestrator-Paket oder -Skript an, das das Verhalten des Plug-Ins beim Start definiert.

Das Element `<action>` ist optional. Ein Plug-In kann über beliebig viele `<action>`-Elemente verfügen. Das Element `<action>` besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
resource	Zeichenfolge	Der Pfad zum Java-Paket oder -Skript vom Stammordner der DAR-Datei. Erforderliches Attribut.
type	install-package oder execute-script	Entweder wird das angegebene Orchestrator-Paket auf dem Orchestrator-Server installiert oder das angegebene Skript ausgeführt. Erforderliches Attribut.

**Tabelle 6-8.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<installation>	Keine

## Element „finder-datasources“

Das Element <finder-datasources> ist der Container für die Elemente <finder-datasource>.

Das Element <finder-datasources> ist optional. Das Element <finder-datasources> besitzt keine Attribute.

**Tabelle 6-9.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<module>	<finder-datasource>

## Element „finder-datasource“

Das Element <finder-datasource> verweist auf die Java-Klassendatei der IPluginAdaptor-Implementierung, die Sie für das Plug-In erstellen.

Sie legen im <finder-datasource>-Element fest, wie Orchestrator auf die Objekte der integrierten Technologie zugreift. Das Element <finder-datasource> gibt die Java-Klasse des erstellten Plug-In-Adapters an. Die Adapterklasse des Plug-Ins instanziiert die erstellte Plug-In-Factory. Die Plug-In-Factory legt die Methoden fest, mit denen Objekte in der integrierten Technologie gefunden werden. Sie können die Zeitüberschreitungszeitwerte für die Finder-Methodenaufrufe, die die Factory durchführt, im Element <finder-datasource> festlegen. Für die unterschiedlichen Finder-Methoden von der IPluginFactory-Schnittstelle gelten unterschiedliche Zeitüberschreitungszeitwerte.

Das Element <finder-datasource> ist optional. Ein Plug-In kann über beliebig viele <finder-datasources>-Elemente verfügen. Das Element <finder-datasource> besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
name	Zeichenfolge	Kennzeichnet die Datenquelle in den datasource-Attributen des <finder>-Elements. Entspricht einer XML-id. Erforderliches Attribut.
adaptor-class	Java-Klasse	Verweist auf die IPluginAdaptor-Implementierung, die Sie für das Erstellen des Plug-In-Adapters festlegen. Beispiel: com.vmware.plugins.sample.Adaptor. Erforderliches Attribut.

Attribute	Wert	Beschreibung
concurrent-call	true (Standard) oder false	Ermöglicht mehreren Benutzern den gleichzeitigen Zugriff auf den Adapter. Sie müssen <code>concurrent-call</code> auf <code>false</code> setzen, wenn das gleichzeitige Aufrufen vom Plug-In nicht unterstützt wird. Optionales Attribut.
invoker-mode	direct (Standard) oder timeout	Legt einen Zeitüberschreitungswert für die Finder-Funktion fest. Wenn die Einstellung <code>direct</code> lautet, erfolgt keine Zeitüberschreitung. Lautet die Einstellung <code>timeout</code> , wendet der Orchestrator-Server die Zeitüberschreitungsdauer an, die der Finder-Methode entspricht. Optionales Attribut.
anonymous-login-mode	never (Standard) oder always	Hiermit wird festgelegt, ob der Benutzername und das Kennwort des Benutzers an das Plug-In übergeben werden. Optionales Attribut.
timeout-fetch-relation	Zahl; Standard 30 Sekunden	Gilt für alle Aufrufe von <code>findRelation()</code> . Optionales Attribut.
timeout-find-all	Zahl; Standard 60 Sekunden	Gilt für alle Aufrufe von <code>findAll()</code> . Optionales Attribut.
timeout-find	Zahl; Standard 60 Sekunden	Gilt für alle Aufrufe von <code>find()</code> . Optionales Attribut.
timeout-has-children-in-relation	Zahl; Standard 2 Sekunden	Gilt für alle Aufrufe von <code>findChildrenInRelation()</code> . Optionales Attribut.
timeout-execute-plugin-command	Zahl; Standard 30 Sekunden	Gilt für alle Aufrufe von <code>executePluginCommand()</code> . Optionales Attribut.

**Tabelle 6-10.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<finder-datasources>	Keine

## inventory-Element

Das Element <inventory> definiert die Stammebene der hierarchischen Liste für das Plug-In, die in der Ansicht **Bestand** und in Dialogfeldern für die Objektauswahl im Orchestrator-Client angezeigt wird.

Das Element <inventory> stellt kein Objekt in der als Plug-In integrierten Anwendung dar, sondern repräsentiert das Plug-In selbst als Objekt in der Skripterstellungs-API von Orchestrator.

Das Element <inventory> ist optional. Das Element <inventory> besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
type	Ein Orchestrator-Objektyp	Der Typ des <finder>-Elements, der die Stammebene der Hierarchie von Objekten darstellt. Erforderliches Attribut.

**Tabelle 6-11.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<module>	Keine

## Element „finders“

Das Element <finders> ist der Container für die Elemente <finder>.

Das Element <finders> ist optional. Das Element <finders> besitzt keine Attribute.

**Tabelle 6-12.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<module>	<finder>

## Element „finder“

Das Element <finder> steht im Orchestrator-Client für einen Objekttyp, der über das Plug-In gefunden wurde.

Das Element <finder> identifiziert die Java-Klasse, die das vom Objekt-Finder dargestellte Objekt definiert. Das Element <finder> definiert, wie das Objekt in der Orchestrator-Client-Schnittstelle angezeigt wird. Es identifiziert auch das Skriptobjekt, das die Orchestrator-Skript-API zum Darstellen dieses Objekts definiert.

Finder dienen als Schnittstelle zwischen Objektformaten, die von verschiedenen Typen integrierter Technologien verwendet werden.

Das Element <finder> ist optional. Ein Plug-In kann über beliebig viele <finder>-Elemente verfügen. Das Element <finder> definiert die folgenden Attribute:

Attribute	Wert	Beschreibung
type	Ein Orchestrator-Objekttyp	Vom Finder dargestellter Objekttyp. Erforderliches Attribut.
datasource	Attribut <finder-datasource name>	Identifiziert die Java-Klasse, die das Objekt definiert, anhand der Datenquelle ref.id. Erforderliches Attribut.
dynamic-finder	Java-Methode	Definieren Sie eine benutzerdefinierte Finder-Methode, die Sie in einer IDynamicFinder-Instanz implementieren, um die ID und Eigenschaften eines Finders programmatisch zurückzugeben, statt sie in der Datei vso.xml zu definieren. Optionales Attribut.
hidden	true oder false (Standard)	Bei true wird der Finder im Orchestrator-Client ausgeblendet. Optionales Attribut.
image	Pfad zu einer Grafikdatei	Ein 16x16-Symbol, das den Finder in hierarchischen Listen im Orchestrator-Client darstellt. Optionales Attribut.
java-class	Name einer Java-Klasse	Die Java-Klasse, die das vom Finder gefundene Objekt definiert und einem Skriptobjekt zuordnet. Optionales Attribut.
script-object	<scripting-object type>-Attribut	Der <scripting-object>-Typ, dem dieser Finder ggf. zuzuordnen ist. Optionales Attribut.

**Tabelle 6-13.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<finders>	<ul style="list-style-type: none"> <li>■ &lt;id&gt;</li> <li>■ &lt;description&gt;</li> <li>■ &lt;properties&gt;</li> <li>■ &lt;default-sorting&gt;</li> <li>■ &lt;inventory-children&gt;</li> <li>■ &lt;relations&gt;</li> <li>■ &lt;inventory-tabs&gt;</li> <li>■ &lt;events&gt;</li> </ul>

## Eigenschaftenelement

Das Element <properties> ist der Container für die <finder><property>-Elemente.

Das Element <properties> ist optional. Das Element <properties> besitzt keine Attribute.

**Tabelle 6-14.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<finder>	<property>

## Eigenschaftselement

Das Element <property> ordnet die Eigenschaften des gefundenen Objekts den Java-Eigenschaften oder Methoden aufrufen zu.

Sie können die Methoden der Klasse SDKFinderProperty aufrufen, wenn Sie die Plug-In-Factory implementieren, um Eigenschaften für die Verarbeitung durch die Plug-In-Factory zu erhalten.

Sie können Objekteigenschaften in den Ansichten im Orchestrator-Client anzeigen oder ausblenden. Sie können auch die Enumeration verwenden, um Objekteigenschaften zu definieren.

Das Element <property> ist optional. Ein Plug-In kann über beliebig viele <property>-Elemente verfügen. Das Element <property> besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
name	Findername	Der Name, den FinderResult verwendet, um das Element zu speichern. Erforderliches Attribut.
display-name	Findername	Der angezeigte Eigenschaftensname. Optionales Attribut.
bean-property	Eigenschaftensname	<p>Sie verwenden das Attribut bean-property, um eine Eigenschaft zu kennzeichnen, die Sie mit den Vorgängen get und set beziehen möchten. Wenn Sie eine Eigenschaft mit dem Namen MyProperty kennzeichnen, definiert das Plug-In die Vorgänge getMyProperty und setMyProperty.</p> <p>Sie können entweder bean-property oder property-accessor, aber nicht beide festlegen. Optionales Attribut.</p>

Attribute	Wert	Beschreibung
property-accessor	Die Methode, die einen Eigenschaftswert von einem Objekt erhält	Das Attribut <code>property-accessor</code> ermöglicht Ihnen die Definition eines OGNL-Ausdrucks, um die Eigenschaften eines Objekts zu validieren. Sie können entweder <code>bean-property</code> oder <code>property-accessor</code> , aber nicht beide. Optionales Attribut.
show-in-column	<code>true</code> (Standard) oder <code>false</code>	Wenn <code>true</code> , wird diese Eigenschaft in der Ergebnistabelle des Orchestrator-Clients angezeigt. Optionales Attribut.
show-in-description	<code>true</code> (Standard) oder <code>false</code>	Wenn <code>true</code> , wird die Eigenschaft in der Objektbeschreibung angezeigt. Optionales Attribut.
hidden	<code>true</code> oder <code>false</code> (Standard)	Wenn <code>true</code> , wird diese Eigenschaft in allen Fällen angezeigt. Optionales Attribut.
linked-enumeration	Name der Enumeration	Verknüpft eine Findereigenschaft mit einer Enumeration. Optionales Attribut.

**Tabelle 6-15.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;properties&gt;</code>	Untergeordnete Elemente

## relations-Element

Das Element `<relations>` ist der Container für die `<finder>``<relation>`-Elemente.

Das Element `<relations>` ist optional. Das Element `<relations>` besitzt keine Attribute.

**Tabelle 6-16.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<code>&lt;finder&gt;</code>	<code>&lt;relation&gt;</code>

## relation-Element

Das Element `<relation>` definiert, wie die Beziehungen von Objekten zu anderen Objekten aussehen.

Sie definieren den Relationsnamen im Element `<relation>`.

Das Element `<relation>` ist optional. Ein Plug-In kann über beliebig viele `<relation>`-Elemente verfügen. Das Element `<relation>` besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
name	Beziehungsname	Ein Name für diese Beziehung. Erforderliches Attribut.
type	Orchestrator-Objektyp	Der Typ des Objekts, der mit einem anderen Objekt über diese Beziehung verbunden ist. Erforderliches Attribut.
cardinality	<code>to-one</code> oder <code>to-many</code>	Definiert die Beziehung zwischen den Objekten als 1:1 oder 1:N. Optionales Attribut.

**Tabelle 6-17.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<relations>	Keine

## ID-Element

Das <id>-Element definiert eine Methode, um eine eindeutige ID für das Objekt zu erhalten, das der Finder identifiziert.

Das Element <id> ist optional. Das Element <id> besitzt die folgenden Attribute.

Attribute	Wert	Beschreibung
accessor	Methodenname	Das accessor-Attribut ermöglicht Ihnen die Definition eines OGNL-Ausdrucks, um die Eigenschaften eines Objekts zu validieren. Erforderliches Attribut.

**Tabelle 6-18.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<finder>	Keine

## inventory-children-Element

Das Element <inventory-children> definiert die Hierarchie der Listen, die in der Ansicht **Bestand** und in Dialogfeldern für die Objektauswahl im Orchestrator-Client angezeigt werden.

Das Element <inventory-children> ist optional. Das Element <inventory-children> besitzt keine Attribute.

**Tabelle 6-19.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<finder>	<relation-link>

## relation-link-Element

Das Element <relation-link> definiert die Hierarchien zwischen übergeordneten und untergeordneten Objekten auf der Registerkarte **Bestand**.

Das Element <relation-link> ist optional. Ein Plug-In kann über beliebig viele <relation-link>-Elemente verfügen. Das Element <relation-link> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
name	Beziehungsname	Ein refid zu einem Beziehungsnamen. Erforderliches Attribut.

**Tabelle 6-20.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<inventory-children>	Keine



## Element „events“

Das Element `<events>` ist der Container für die Elemente `<trigger>` und `<gauge>`.

Das Element `<events>` kann eine unbegrenzte Anzahl an Auslösern oder Kontrollen enthalten.

Das Element `<events>` ist optional. Das Element `<events>` besitzt keine Attribute.

**Tabelle 6-21.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;finder&gt;</code>	<ul style="list-style-type: none"> <li>■ <code>&lt;trigger&gt;</code></li> <li>■ <code>&lt;gauge&gt;</code></li> </ul>

## Element „trigger“

Das Element `<trigger>` kennzeichnet die Auslöser, die Sie für diesen Finder verwenden können. Zum Festlegen von Auslösern müssen Sie die Methoden `registerEventPublisher()` und `unregisterEventPublisher()` von `IPluginAdaptor` implementieren.

Das Element `<trigger>` ist optional. Das Element `<trigger>` besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
name	Auslösername	Der Name des Auslösers. Erforderliches Attribut.

**Tabelle 6-22.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;events&gt;</code>	<ul style="list-style-type: none"> <li>■ <code>&lt;description&gt;</code></li> <li>■ <code>&lt;trigger-properties&gt;</code></li> </ul>

## trigger-properties-Element

Das Element `<trigger-properties>` ist der Container für die `<trigger-property>`-Elemente.

Das Element `<trigger-properties>` ist optional. Das Element `<trigger-properties>` besitzt keine Attribute.

**Tabelle 6-23.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<code>&lt;trigger&gt;</code>	<code>&lt;trigger-property&gt;</code>

## trigger-property-Element

Das Element `<trigger-property>` definiert die Eigenschaften, die ein Triggerobjekt definieren.

Das Element `<trigger-property>` ist optional. Ein Plug-In kann über beliebig viele `<trigger-property>`-Elemente verfügen. Das Element `<trigger-property>` besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
name	Auslösername	Ein Name für den Auslöser. Optionales Attribut.
display-name	Auslösername	Der Name, der im Orchestrator-Client angezeigt wird. Optionales Attribut.
type	Auslösertyp	Der Objekttyp, der den Auslöser definiert. Erforderliches Attribut.

**Tabelle 6-24.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<Auslösereigenschaften>	Keine

## Element „gauge“

Das Element <gauge> definiert die Kontrollen, die Sie für diesen Finder verwenden können. Zum Festlegen von Kontrollen müssen Sie die Methoden `registerEventPublisher()` und `unregisterEventPublisher()` von `IPluginAdaptor` implementieren.

Das Element <gauge> ist optional. Ein Plug-In kann über beliebig viele <gauge>-Elemente verfügen. Das Element <gauge> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
name	Name der Kontrolle	Ein Name für die Kontrolle. Erforderliches Attribut.
min-value	Zahl	Mindestschwellenwert. Optionales Attribut.
max-value	Zahl	Maximalschwellenwert. Optionales Attribut.
unit	Objekttyp	Objekttyp, der die Kontrolle definiert. Erforderliches Attribut.
format	String	Das Format des überwachten Werts. Optionales Attribut.

**Tabelle 6-25.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<events>	<description>

## scripting-objects-Element

Das Element <scripting-objects> ist der Container für die <object>-Elemente.

Das Element <scripting-objects> ist optional. Das Element <scripting-objects> besitzt keine Attribute.

**Tabelle 6-26.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<module>	<object>

## Objektelement

Das Element `<object>` ordnet die Konstruktoren der Plug-In-Technologie, Attribute und Methoden den JavaScript-Objekttypen zu, die durch die Skripterstellung-API von Orchestrator bereitgestellt werden.

Unter „[Benennen von Plug-In-Objekten](#)“, auf Seite 59 finden Sie Informationen über Namenskonventionen für Objekte.

Das Element `<object>` ist optional. Ein Plug-In kann über beliebig viele `<object>`-Elemente verfügen. Das Element `<object>` besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
<code>script-name</code>	JavaScript-Name	Klassenname für die Skripterstellung. Muss global eindeutig sein. Erforderliches Attribut.
<code>java-class</code>	Java-Klasse	Die Java-Klasse, die in dieser JavaScript-Klasse enthalten ist. Erforderliches Attribut.
<code>create</code>	true (Standard) oder false	Wenn true, können Sie eine neue Instanz dieser Klasse erstellen. Optionales Attribut.
<code>strict</code>	true oder false (Standard)	Wenn true, können Sie nur Methoden aufrufen, die Sie in der <code>vso.xml</code> -Datei anmerken oder deklarieren. Optionales Attribut.
<code>is-deprecated</code>	true oder false (Standard)	Wenn true, ordnet das Objekt eine veraltete Java-Klasse zu. Optionales Attribut.
<code>since-version</code>	String	Version, seit der die Java-Klasse veraltet ist. Optionales Attribut.

**Tabelle 6-27.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;scripting-objects&gt;</code>	<ul style="list-style-type: none"> <li>■ <code>&lt;description&gt;</code></li> <li>■ <code>&lt;deprecated&gt;</code></li> <li>■ <code>&lt;url&gt;</code></li> <li>■ <code>&lt;constructors&gt;</code></li> <li>■ <code>&lt;attributes&gt;</code></li> <li>■ <code>&lt;methods&gt;</code></li> <li>■ <code>&lt;singleton&gt;</code></li> </ul>

## Element „constructors“

Das Element `<constructors>` ist der Container für die `<object><constructor>`-Elemente.

Das Element `<constructors>` ist optional. Das Element `<constructors>` besitzt keine Attribute.

**Tabelle 6-28.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<code>&lt;object&gt;</code>	<code>&lt;constructor&gt;</code>

## Element „constructor“

Das Element `<constructor>` definiert eine Konstruktormethode. Mithilfe des Elements `<constructor>` wird Dokumentation im API-Explorer generiert.

Das Element `<constructor>` ist optional. Ein Plug-In kann über beliebig viele `<constructor>`-Elemente verfügen. Das Element `<constructor>` besitzt keine Attribute.

**Tabelle 6-29.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;constructors&gt;</code>	<ul style="list-style-type: none"> <li>■ <code>&lt;description&gt;</code></li> <li>■ <code>&lt;parameters&gt;</code></li> </ul>

## Element „parameters“ des Konstruktors

Das Element `<parameters>` ist der Container für die `<constructor>``<parameter>`-Elemente.

Das Element `<parameters>` ist optional. Das Element `<parameters>` besitzt keine Attribute.

**Tabelle 6-30.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<code>&lt;constructor&gt;</code>	<code>&lt;parameter&gt;</code>

## Konstruktor-Parameterelement

Das `<parameter>`-Element definiert die Parameter des Konstruktors.

Das Element `<parameter>` ist optional. Ein Plug-In kann über beliebig viele `<parameter>`-Elemente verfügen. Das Element `<parameter>` besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
<code>name</code>	String	In API-Dokumentation zu verwendender Parametername. Erforderliches Attribut.
<code>type</code>	Orchestrator-Parametertyp	In API-Dokumentation zu verwendender Parametertyp. Erforderliches Attribut.
<code>is-optional</code>	true oder false	Bei true kann der Wert null sein. Optionales Attribut.
<code>since-version</code>	String	Methodenversion. Optionales Attribut.

**Tabelle 6-31.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<code>&lt;parameters&gt;</code>	Keine

## Element „attributes“

Das Element `<attributes>` ist der Container für die `<object>``<attribute>`-Elemente.

Das Element `<attributes>` ist optional. Das Element `<attributes>` besitzt keine Attribute.

**Tabelle 6-32.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<object>	<attribute>

## Element „attribute“

Das Element <attribute> ordnet die Attribute einer Java-Klasse von der Plug-In-Technologie zu JavaScript-Attributen zu, die das Orchestrator-JavaScript-Modul zur Verfügung stellt.

Das Element <attribute> ist optional. Ein Plug-In kann über beliebig viele <attribute>-Elemente verfügen. Das Element <attribute> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
java-name	Java-Attribut	Name des Java-Attributs. Erforderliches Attribut.
script-name	JavaScript-Objekt	Name des entsprechenden JavaScript-Objekts. Erforderliches Attribut.
return-type	Zeichenfolge	Der Objekttyp, der durch dieses Attribut zurückgegeben wird. Wird in der API-Explorer-Dokumentation angezeigt. Optionales Attribut. <b>HINWEIS</b> Wenn der JavaScript-Rückgabebetyp <code>Properties</code> ist, sind die unterstützten, zugrunde liegenden Java-Implementierungen <code>java.util.HashMap</code> und <code>java.util.Hashtable</code> .
read-only	true oder false	Wenn true, können Sie dieses Attribut nicht ändern. Optionales Attribut.
is-optional	true oder false	Wenn true, kann dieses Feld einen Nullwert aufweisen. Optionales Attribut.
show-in-api	true oder false	Wenn false, wird dieses Attribut nicht in der API-Dokumentation angezeigt. Optionales Attribut.
is-deprecated	true oder false	Wenn true, ordnet das Objekt ein veraltetes Attribut zu. Optionales Attribut.
since-version	Zahl	Die Version, ab der das Attribut veraltet war. Optionales Attribut.

**Tabelle 6-33.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<attributes>	Keine

## Methodenelement

Das Element <methods> ist der Container für die <object><method>-Elemente.

Das Element <methods> ist optional. Das Element <methods> besitzt keine Attribute.

**Tabelle 6-34.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<object>	<method>

## Methodenelement

Das Element <method> ordnet eine Java-Methode von der Plug-In-Technologie einer Methode zu, die die Orchestrator-JavaScript-Engine zur Verfügung stellt.

Das Element <method> ist optional. Ein Plug-In kann über beliebig viele <method>-Elemente verfügen. Das Element <method> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
java-name	Java-Methode	Name der Java-Methodensignatur mit Argumenttypen in Klammern, beispielsweise <code>getVms(DataStore)</code> . Erforderliches Attribut.
script-name	JavaScript-Methode	Name der entsprechenden JavaScript-Methode. Erforderliches Attribut.
return-type	Java-Objektyp	Der Typ, den diese Methode erhält. Optionales Attribut. <b>HINWEIS</b> Wenn der JavaScript-Rückgabetyt <code>Properties</code> ist, sind die unterstützten, zugrunde liegenden Java-Implementierungen <code>java.util.HashMap</code> und <code>java.util.Hashtable</code> .
static	true oder false	Wenn true, ist diese Methode statisch. Optionales Attribut.
show-in-api	true oder false	Wenn false, wird diese Methode nicht in der API-Dokumentation angezeigt. Optionales Attribut.
is-deprecated	true oder false	Wenn true, ordnet das Objekt eine veraltete Methode zu. Optionales Attribut.
since-version	Zahl	Die Version, ab der die Methode veraltet war. Optionales Attribut.

**Tabelle 6-35.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<methods>	<ul style="list-style-type: none"> <li>■ &lt;deprecated&gt;</li> <li>■ &lt;description&gt;</li> <li>■ &lt;example&gt;</li> <li>■ &lt;parameters&gt;</li> </ul>

## Element „example“

Das Element <example> ermöglicht das Hinzufügen von Beispielcode für JavaScript-Methoden, die in der API-Explorer-Dokumentation vorkommen.

Das Element <example> ist optional. Das Element <example> besitzt keine Attribute.

**Tabelle 6-36.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<method>	<ul style="list-style-type: none"> <li>■ &lt;code&gt;</li> <li>■ &lt;description&gt;</li> </ul>

## Element „code“

Das Element <code> stellt Beispielcode bereit, der in der API-Explorer-Dokumentation angezeigt wird.

Sie stellen das Codebeispiel zwischen den Tags <code> und </code> bereit. Das Element <code> ist optional. Das Element <code> besitzt keine Attribute.

**Tabelle 6-37.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<example>	Keine

## Methodenparameterelement

Das Element <parameters> ist der Container für die <method><parameter>-Elemente.

Das Element <parameters> ist optional. Das Element <parameters> besitzt keine Attribute.

**Tabelle 6-38.**

Übergeordnetes Element	Untergeordnetes Element
<method>	<parameter>

## Methodenparameterelement

Das Element <parameter> definiert die Eingabeparameter der Methode.

Das Element <parameter> ist optional. Ein Plug-In kann über beliebig viele <parameter>-Elemente verfügen. Das Element <parameter> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
name	String	Parametername. Erforderliches Attribut.
type	Orchestrator-Parametertyp	Parametertyp. Erforderliches Attribut.
is-optional	true oder false	Bei true kann der Wert null sein. Optionales Attribut.
since-version	String	Methodenversion. Optionales Attribut.

**Tabelle 6-39.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<parameters>	Keine

## Singleton-Element

Das Element `<singleton>` erstellt ein JavaScript-Skriptobjekt als Singleton-Instanz.

Ein Singleton-Objekt verhält sich wie eine statische Java-Klasse. Singleton-Objekte definieren generische Objekte zur Verwendung durch das Plug-In. Sie definieren nicht eine bestimmte Instanz eines Objekts, auf das Orchestrator in der integrierten Technologie zugreift. Beispiel: Sie können ein Singleton-Objekt verwenden, um eine Verbindung zu einer integrierten Technologie herzustellen.

Das Element `<singleton>` ist optional. Das Element `<singleton>` besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
script-name	JavaScript-Objekt	Name des entsprechenden JavaScript-Objekts. Erforderliches Attribut.
datasource	Java-Objekt	Das Java-Quellobjekt für das JavaScript-Objekt. Erforderliches Attribut.

**Tabelle 6-40.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<code>&lt;object&gt;</code>	Keine

## Element „enumerations“

Das Element `<enumerations>` ist der Container für die Elemente `<enumeration>`.

Das Element `<enumerations>` ist optional. Das Element `<enumerations>` besitzt keine Attribute.

**Tabelle 6-41.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<code>&lt;module&gt;</code>	<code>&lt;enumeration&gt;</code>

## Element „enumeration“

Das Element `<enumeration>` definiert häufige Werte, die für alle Objekte eines bestimmten Typs gelten.

Wenn alle Objekte eines bestimmten Typs ein bestimmtes Attribut erfordern und wenn der Wertebereich für dieses Attribut limitiert ist, können Sie die verschiedenen Werte als Enumerationseinträge definieren. Wenn beispielsweise ein Objekttyp das Attribut `color` erfordert und die einzig verfügbaren Farben rot, blau und grün sind, können Sie drei Enumerationseinträge zum Definieren dieser drei Farbwerte erstellen. Sie definieren Einträge als untergeordnete Elemente des Elements „enumeration“.

Das Element `<enumeration>` ist optional. Ein Plug-In kann über beliebig viele `<enumeration>`-Elemente verfügen. Das Element `<enumeration>` besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
type	Orchestrator-Objektyp	Enumerationstyp. Erforderliches Attribut.



**Tabelle 6-42.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<enumerations>	<ul style="list-style-type: none"> <li>■ &lt;url&gt;</li> <li>■ &lt;description&gt;</li> <li>■ &lt;entries&gt;</li> </ul>

## Element „entries“

Das Element <entries> ist der Container für die Elemente <enumeration><entry>.

Das Element <entries> ist optional. Das Element <entries> besitzt keine Attribute.

**Tabelle 6-43.** Elementhierarchie

Übergeordnetes Element	Untergeordnetes Element
<enumeration>	<entry>

## Element „entry“

Das Element <entry> bietet einen Wert für ein Enumerationsattribut.

Das Element <entry> ist optional. Ein Plug-In kann über beliebig viele <entry>-Elemente verfügen. Das Element <entry> besitzt die folgenden Attribute.

Typ	Wert	Beschreibung
id	Text	Der von Objekten verwendete Bezeichner, um den Enumerationseintrag als Attribut festzulegen. Erforderliches Attribut.
name	Text	Der Name des Eintrags. Erforderliches Attribut.

**Tabelle 6-44.** Elementhierarchie

Übergeordnetes Element	Untergeordnete Elemente
<entries>	Keine

## Best Practices zur Entwicklung von Orchestrator-Plug-Ins

Sie können bestimmte Aspekte der entwickelten Orchestrator Plug-Ins verbessern durch Verständnis der Struktur und des Inhalts der Plug-Ins sowie durch Verständnis darüber, wie man bestimmte Probleme vermeidet.

- [Methoden zum Erstellen eines Orchestrator-Plug-Ins](#) auf Seite 266

Zum Erstellen von Orchestrator-Plug-Ins können Sie verschiedene Methoden anwenden. Sie können entweder die Ebenen des Plug-Ins einzeln erstellen oder Sie erstellen alle Ebenen gleichzeitig.

- [Typen von Orchestrator-Plug-Ins](#) auf Seite 268

Mithilfe von Plug-Ins können Sie Bibliotheken und Dienstprogramme für allgemeine Zwecke (etwa XML oder SSH) in Orchestrator integrieren oder auch ganze Systeme (etwa vCloud Director). Je nach Technologie, die Sie in Orchestrator integrieren, fallen Plug-Ins in verschiedene Kategorien: Plug-Ins für Dienste, für allgemeine Zwecke oder für Systeme.

- **Plug-In-Implementierung** auf Seite 271

Sie können beim Strukturieren Ihrer Plug-Ins bestimmte hilfreiche Methoden und Techniken verwenden, die erforderlichen Java-Klassen und JavaScript-Objekte implementieren, Plug-In-Workflows und -Aktionen entwickeln und die Workflowdarstellung bereitstellen.

- **Empfehlungen zur Entwicklung von Orchestrator-Plug-Ins** auf Seite 275

Indem Sie sich beim Entwickeln der verschiedenen Komponenten für Ihre Orchestrator-Plug-Ins an bestimmte Grundsätze halten, können Sie die Qualität der Plug-Ins verbessern.

- **Dokumentierung von Plug-In-Benutzeroberflächen-Zeichenfolgen und -APIs** auf Seite 278

Wenn Sie Benutzeroberflächen-Zeichenfolgen für Orchestrator-Plug-Ins und die zugehörige API-Dokumentation schreiben, folgen Sie den allgemein anerkannten Stil- und Formatregeln.

## Methoden zum Erstellen eines Orchestrator-Plug-Ins

Zum Erstellen von Orchestrator-Plug-Ins können Sie verschiedene Methoden anwenden. Sie können entweder die Ebenen des Plug-Ins einzeln erstellen oder Sie erstellen alle Ebenen gleichzeitig.

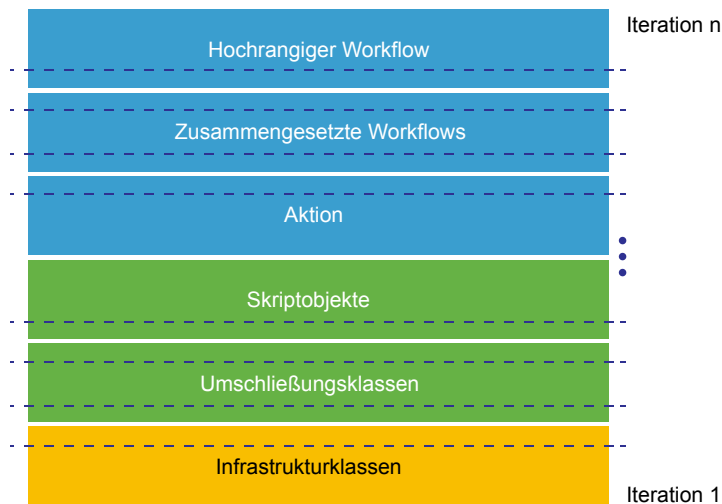
Weitere Informationen zu Plug-In-Ebenen finden Sie unter „[Struktur eines Orchestrator-Plug-Ins](#)“, auf Seite 50.

### Bottom-Up-Plug-In-Entwicklung

Ein Plug-In kann Schicht für Schicht mit dem Bottom-Up-Entwicklungsansatz erstellt werden.

Der Bottom-Up-Entwicklungsansatz erstellt das Plug-In Schicht für Schicht, beginnend mit den Schichten der untersten Ebenen und weiter mit den Schichten der höheren Ebenen. Wird dieser Ansatz mit einem interaktiven und iterativen Entwicklungsansatz kombiniert, wird bei jeder Iteration entweder ein Teil oder eine ganze Schicht fertiggestellt. Nach dem Abschluss von Iteration N ist das Plug-In vollständig.

**Abbildung 6-3.** Bottom-Up-Plug-In-Entwicklung



Ein Vorteil des Bottom-Up-Plug-In-Entwicklungsansatzes ist, dass die Entwicklung sich auf eine Schicht nach der anderen konzentriert.

Berücksichtigen Sie auch die folgenden Nachteile des Bottom-Up-Plug-In-Entwicklungsansatzes.

- Es ist schwierig den Fortschritt der Plug-In-Entwicklung zu präsentieren, bis einige Elemente eingefügt wurden.

- Das passt nicht gut in einen agilen Entwicklungsprozess.

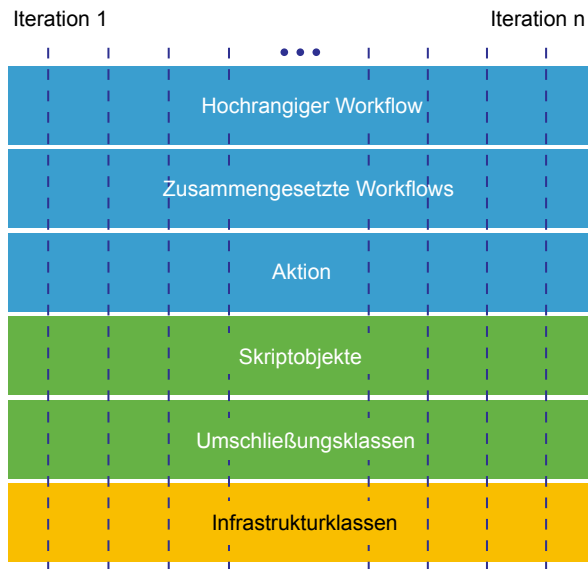
Der Bottom-Up-Entwicklungsprozess wird für einige kleine Plug-Ins mit wenigen oder keinen Umschließungsklassen, Skriptobjekten, Aktionen oder Workflows als ausreichend betrachtet.

## Top-down-Plug-In-Entwicklung

Ein Plug-In kann mithilfe eines Top-down-Ansatzes konstruiert werden, wobei eine Unterteilung der Funktionalität von der obersten bis hin zur untersten Ebene erfolgt.

Wird der Top-down-Ansatz mit einem agilen Entwicklungsprozess kombiniert, wird bei jeder Iteration neue Funktionalität bereitgestellt. Nach dem Abschluss von Iteration N ist das Plug-In vollständig implementiert.

**Abbildung 6-4.** Top-down-Plug-In-Entwicklung



Der Top-down-Ansatz bei der Entwicklung von Plug-Ins bietet die folgenden Vorteile:

- Der Fortschritt bei der Entwicklung des Plug-Ins ab der ersten Iteration ist einfach zu sehen, da bei jeder Iteration neue Funktionalität fertiggestellt wird und das Plug-In nach jeder Iteration veröffentlicht und benutzt werden kann.
- Durch die vertikale Unterteilung der Funktionalität können die Erfolgskriterien und Fortschritte eindeutig festgelegt und die Kommunikation zwischen Entwicklern, Produktmanagement und Quality Assurance (QA)-Technikern verbessert werden.
- Die QA-Techniker können von Beginn des Entwicklungsprozesses an testen und automatisieren. Dieser Ansatz ermöglicht wertvolle Rückmeldungen und verkürzt die Gesamtbereitstellungsdauer des Projekts.

Ein Vorteil der Top-down-Entwicklung von Plug-Ins ist, dass die Entwicklung auf verschiedenen Ebenen gleichzeitig erfolgt.

Die Top-down-Entwicklung wird für die meisten Plug-Ins empfohlen. Sie eignet sich für Plug-Ins mit dynamischen Anforderungen.

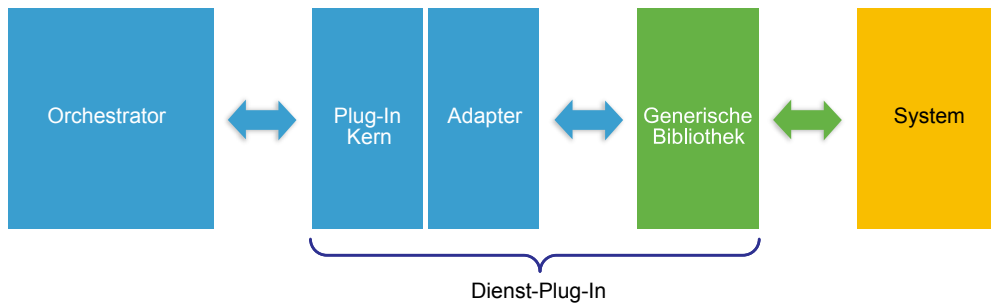
## Typen von Orchestrator-Plug-Ins

Mithilfe von Plug-Ins können Sie Bibliotheken und Dienstprogramme für allgemeine Zwecke (etwa XML oder SSH) in Orchestrator integrieren oder auch ganze Systeme (etwa vCloud Director). Je nach Technologie, die Sie in Orchestrator integrieren, fallen Plug-Ins in verschiedene Kategorien: Plug-Ins für Dienste, für allgemeine Zwecke oder für Systeme.

### Plug-Ins für Dienste

Plug-Ins für Dienste oder Plug-Ins für allgemeine Zwecke bieten Funktionen, die als Dienst innerhalb von Orchestrator angesehen werden können.

**Abbildung 6-5.** Architektur der Plug-Ins für Dienste



Plug-Ins für Dienste stellen generische Bibliotheken oder Dienstprogramme für Orchestrator bereit, wie XML, SSH oder SOAP. Die folgenden in Orchestrator verfügbaren Plug-Ins sind Beispiele für Dienst-Plug-Ins.

<b>JDBC-Plug-In</b>	Damit können Sie jede Datenbank in einem Workflow verwenden.
<b>Mail-Plug-In</b>	Damit können Sie E-Mails in einem Workflow senden.
<b>SSH-Plug-In</b>	Damit können Sie SSH-Verbindungen öffnen und Befehle in einem Workflow ausführen.
<b>XML-Plug-In</b>	Damit können Sie XML-Dokumente in einem Workflow verwalten.

Plug-Ins für Dienste haben die folgenden Merkmale:

<b>Komplexität</b>	Plug-Ins für Dienste weisen einen Komplexitätsgrad von niedrig bis mittel auf. Plug-Ins für Dienste machen eine spezifische Bibliothek oder Teile einer Bibliothek innerhalb von Orchestrator zugänglich, um konkrete Funktionen bereitzustellen. Das XML-Plug-In fügt beispielsweise der Orchestrator-JavaScript-API eine Implementierung eines DOM (Document Object Model)-XML-Parsers hinzu.
<b>Größe</b>	Plug-Ins für Dienste weisen eine relativ kleine Größe auf. Sie erfordern denselben grundlegenden Satz an Klassen wie alle Plug-Ins sowie weitere Klassen, die neue Skriptobjekte für neue Funktionen bieten.
<b>Bestandsliste</b>	Plug-Ins für Dienste benötigen eine kleine Bestandsliste von Objekten, um zu funktionieren, oder sie benötigen überhaupt keine Bestandsliste. Plug-Ins für Dienste haben ein generisches und ein kleines Objektmodell. Daher müssen Sie dieses Modell in der Orchestrator-Bestandsliste überhaupt nicht zeigen.

## Plug-Ins für Systeme

Plug-Ins für Systeme verbinden die Engine für Orchestrator-Workflows mit einem externen System, sodass Sie das externe System steuern können.

Es folgen einige Beispiele für Plug-Ins für Systeme.

<b>vCenter Server-Plug-In</b>	Damit können Sie vCenter Server-Instanzen mithilfe von Workflows verwalten.
<b>vCloud Director-Plug-In</b>	Damit können Sie mit einer vCloud Director-Installation in einem Workflow interagieren.
<b>Cisco UCSM Plug-In</b>	Damit können Sie mit Cisco-Einheiten in einem Workflow interagieren.

Es folgen die Hauptmerkmale von Plug-Ins für Systeme.

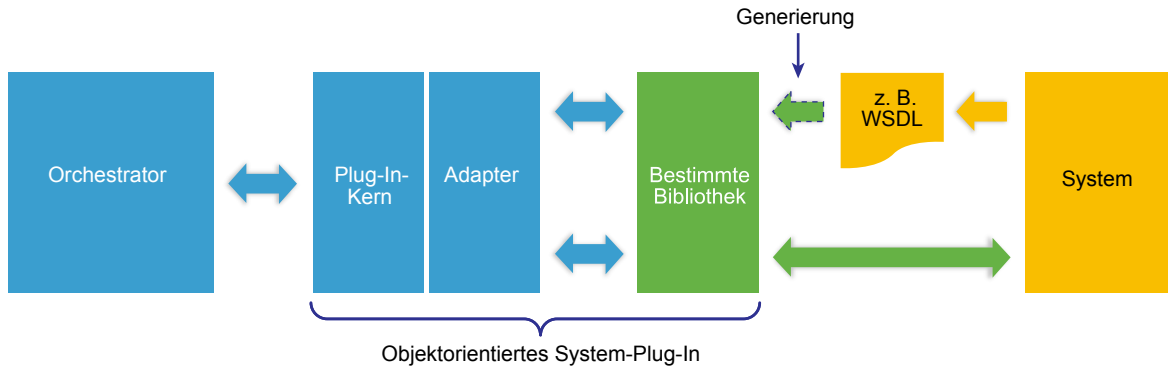
<b>Komplexität</b>	Plug-Ins für Systeme weisen höhere Komplexität als herkömmliche Plug-Ins auf, da die von ihnen eingesetzten Technologien relativ komplex sind. Plug-Ins für Systeme stellen alle Elemente des externen Systems im Orchestrator dar, die mit dem externen System interagieren und ihre Funktionalität in Orchestrator bereitstellen. Wenn das externe System einen Integrationsmechanismus bietet, können Sie diesen verwenden, um die Funktionalität des Systems leichter zugänglich zu machen. Zusätzlich zum Darstellen der Elemente des externen Systems in Orchestrator müssen Plug-Ins für Systeme ggf. auch hohe Skalierbarkeit bieten, einen Cache-Mechanismus bereitstellen, Ereignisse und Benachrichtigungen handhaben können usw.
<b>Größe</b>	Die Größe von Plug-Ins für Systeme reicht von mittel bis groß. Plug-Ins für Systeme benötigen zusätzlich zum grundlegenden Satz an Klassen zahlreiche weitere Klassen, da sie normalerweise eine große Anzahl an Skriptobjekten bieten. Plug-Ins für Systeme benötigen ggf. ein anderes Hilfsprogramm und Hilfsklassen, die mit ihnen interagieren.
<b>Bestandsliste</b>	In der Regel verfügen Plug-Ins für Systeme über eine große Anzahl an Objekten. Sie müssen diese Objekte in der Bestandsliste richtig darstellen, damit Sie sie lokalisieren und in Orchestrator mühelos mit ihnen arbeiten können. Aufgrund der großen Anzahl an Objekten, die Plug-Ins für Systeme darstellen müssen, sollten Sie ein Hilfstool oder einen Prozess erstellen, um für das Plug-In möglichst viel Code automatisch zu generieren. Das vCenter Server-Plug-In bietet ein solches Tool.

## Plug-Ins für objektorientierte Systeme

Objektorientierte Systeme bieten einen Interaktionsmechanismus auf Basis von Objekten und RPC.

Das gängigste Modell für ein objektorientiertes System ist das Webdienstmodell, das SOAP verwendet. Die Objekte in diesem Modell enthalten einen Satz an Attributen, die sich auf den Status der Objekte beziehen, und bieten einen Satz an Remotemethoden, die auf Seiten des Zielsystems aufgerufen werden.

### Abbildung 6-6. Plug-Ins für objektorientierte Systeme



Beim Implementieren von Plug-Ins für objektorientierte Systeme können Sie Folgendes berücksichtigen.

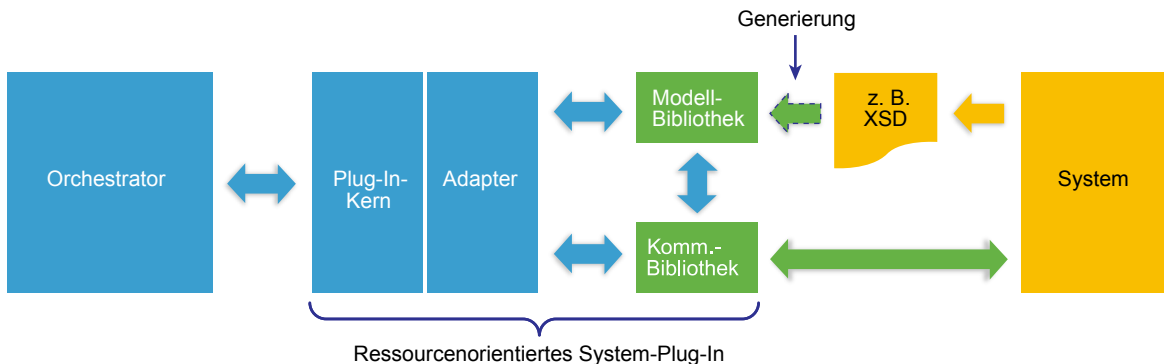
- Beim Verwenden von SOAP können Sie die WSDL-Datei verwenden, um einen Satz an Klassen zu generieren, die das Objektmodell und den Kommunikationsmechanismus kombinieren.
- Dieses Objektmodell ist im Wesentlichen alles, was Sie in Orchestrator zeigen müssen.

## Plug-Ins für ressourcenorientierte Systeme

Ressourcenorientierte Systeme bieten einen Interaktionsmechanismus, der auf Ressourcen und einfachen Vorgängen basiert, die HTTP-Methoden verwenden.

Das repräsentativste Modell für ein ressourcenorientiertes System ist das REST-Modell in Kombination beispielsweise mit XML. Die Objekte in diesem Modell haben einen Satz an Attributen, die sich auf ihren Status beziehen. Um Methoden auf dem Zielsystem (Kommunikationssystem) aufzurufen, müssen Sie standardmäßige HTTP-Methoden wie GET, POST, PUT usw. verwenden und einige Konventionen befolgen.

**Abbildung 6-7. Plug-Ins für ressourcenorientierte Systeme**



Beim Entwickeln von Plug-Ins für ressourcenorientierte Systeme können Sie Folgendes berücksichtigen.

- Wenn Sie REST oder nur HTTP in Verbindung mit XML verwenden, erhalten Sie eine oder mehrere XML-Schemadateien, um Nachrichten lesen und schreiben zu können. Über diese Schemas können Sie einen Satz an Klassen generieren, die das Objektmodell definieren. Mit diesem Satz an Klassen wird nur der Status der Objekte definiert, da die Vorgänge implizit mit den HTTP-Methoden definiert werden (beispielsweise beim vCloud Director-Plug-In), oder explizit mit einigen speziellen XML-Nachrichten (beispielsweise beim Cisco UCSM Plug-In).
- Sie müssen den Kommunikationsmechanismus in einem anderen Satz an Klassen implementieren. Mit diesem Satz an Klassen wird ein neues Objektmodell definiert, das mit dem ursprünglichen Objektmodell interagiert. Das Objektmodell für die Kommunikationsmechanismen besteht nur aus Objekten und Methoden.

- Sie können sowohl das ursprüngliche Objektmodell als auch das Objektmodell für den Kommunikationsmechanismus im Orchestrator zeigen. Auf diese Weise kann die Komplexität etwas erhöht werden, abhängig davon, auf welche Weise beide Objektmodelle angezeigt werden, und davon, ob Sie verwandte Objekte von beiden Seiten (zum Simulieren eines objektorientierten Systems) zusammenführen oder sie getrennt halten.

## Plug-In-Implementierung

Sie können beim Strukturieren Ihrer Plug-Ins bestimmte hilfreiche Methoden und Techniken verwenden, die erforderlichen Java-Klassen und JavaScript-Objekte implementieren, Plug-In-Workflows und -Aktionen entwickeln und die Workflowdarstellung bereitstellen.

- [Projektstruktur](#) auf Seite 271  
Sie können eine Standardstruktur für die Projekte der Orchestrator-Plug-Ins anwenden.
- [Projektinterne Ansätze](#) auf Seite 272  
Sie können bestimmte Ansätze beim Implementieren Ihres Plug-Ins anwenden, z. B. Objekte zwischenspeichern, Objekte in den Hintergrund setzen, Objekte klonen usw. Durch solche Ansätze können Sie die Leistung Ihrer Plug-Ins verbessern, Parallelitätsprobleme vermeiden und die Reaktionsfähigkeit des Orchestrator-Clients erhöhen.
- [Workflow-interne Ansätze](#) auf Seite 273  
Sie können einen Workflow implementieren, um langfristige Vorgänge zu überwachen, die das Orchestrator-Plug-In durchführt.
- [Workflows und Aktionen](#) auf Seite 274  
Sie können bestimmte bewährte Vorgehensweisen verwenden, um die Entwicklung und Verwendung von Workflows zu vereinfachen.
- [Workflowpräsentation](#) auf Seite 274  
Wenn Sie die Präsentation eines Workflows erstellen, müssen Sie bestimmte Strukturen und Regeln anwenden.

## Projektstruktur

Sie können eine Standardstruktur für die Projekte der Orchestrator-Plug-Ins anwenden.

Sie können eine Maven-Standardstruktur mit Modulen für Ihre Plug-In-Projekte verwenden, um klarer zu machen, wo sich die Funktionen befinden.

**Tabelle 6-45.** Struktur eines Plug-In-Projekts

Modul	Beschreibung
/myAwesomePlugin-plugin	Der Stamm des Plug-In-Projekts.
/o11nplugin-myAwesomePlugin	Das Modul, das die endgültige Plug-In-DAR-Datei erstellt.
/o11nplugin-myAwesomePlugin-config	Das Modul, das die Webanwendung für die Plug-In-Konfiguration enthält. Es wird eine standardmäßige WAR-Datei generiert.
/o11nplugin-myAwesomePlugin-core	Das Modul, das die Klassen enthält, die beliebige Orchestrator-Plug-In-Schnittstellen sowie andere Hilfsklassen, die sie verwenden, implementieren. Es wird eine standardmäßige JAR-Datei generiert.

**Tabelle 6-45.** Struktur eines Plug-In-Projekts (Fortsetzung)

Modul	Beschreibung
/o11nplugin-myAwesomePlugin-model	Das Modul, dass die Klassen enthält, mit denen Sie die Technologie von Drittanbietern über das Plug-In besser in Orchestrator integrieren können. Die Klassen dürfen keine direkte Referenz auf die standardmäßigen Orchestrator-Plug-In-APIs enthalten.
/o11nplugin-myAwesomePlugin-package	Das Modul, das eine externe Orchestrator-Paketdatei mit Aktionen und Workflows importiert, die in der endgültigen Plug-In-DAR-Datei einbezogen werden soll. Das Modul ist optional.

## Projektinterne Ansätze

Sie können bestimmte Ansätze beim Implementieren Ihres Plug-Ins anwenden, z. B. Objekte zwischenspeichern, Objekte in den Hintergrund setzen, Objekte klonen usw. Durch solche Ansätze können Sie die Leistung Ihrer Plug-Ins verbessern, Parallelitätsprobleme vermeiden und die Reaktionsfähigkeit des Orchestrator-Clients erhöhen.

### Cache-Objekte

Ihr Plug-In kann mit einem Remotedienst interagieren. Diese Interaktion wird von lokalen Objekten bereitgestellt, die Remoteobjekte auf der Dienstseite darstellen. Sie können die lokalen Objekte zwischenspeichern, statt sie jedes mal vom Remoteservice abzurufen, um eine gute Leistung des Plug-Ins und eine gute Reaktionsfähigkeit der Orchestrator-Benutzeroberfläche zu gewährleisten. Überlegen Sie sich den Umfang des Cache, z. B. ein Cache für alle Plug-In-Clients, ein Cache pro Benutzer des Plug-Ins oder ein Cache pro Benutzer des Drittanbieterdienstes. Wenn der Cachemechanismus implementiert ist, wird er in die Plug-In-Schnittstelle integriert und dient zum Suchen und zur Invalidierung von Objekten.

### Objekte in den Hintergrund setzen

Wenn Sie eine umfangreiche Liste von Objekten in der Plug-In-Bestandsliste anzeigen müssen und keine schnelle Möglichkeit zum Abrufen dieser Objekte haben, können Sie diese Objekten in den Hintergrund setzen. Sie können Objekte in den Hintergrund setzen, z. B., indem Sie Objekte mit zwei Zuständen haben: *fake* und *loaded*. Angenommen, die *fake*-Objekte sind sehr einfach zu erstellen und enthalten wenig in der Bestandsliste anzuzeigende Informationen, wie z. B. Name und ID. In diesem Fall ist es möglich, immer *fake*-Objekte zurückzugeben, und wenn wirklich alle Informationen (das reale Objekt) erforderlich sind, kann die verwendende Einheit oder das Plug-In automatisch eine *load*-Methode aufrufen, um das reale Objekt abzurufen. Sie können sogar den Objektladevorgang konfigurieren, sodass dieser automatisch startet, nachdem *Fake*-Objekte zurückgegeben werden. So können Sie den Aktionen der verwendenden Einheit zuvorkommen.

### Objekte klonen, um Parallelitätsprobleme zu vermeiden

Wenn Sie für Ihr Plug-In einen Cache verwenden, müssen Sie Objekte klonen. Das Verwenden eines Cache, der immer dieselbe Instanz eines Objekts an jede anfordernde Einheit zurückgibt, kann unerwünschte Folgen haben. Beispiel: Einheit A fordert Objekt O an, und die Einheit zeigt das Objekt in der Bestandsliste mit allen seinen Attributen an. Gleichzeitig fordert auch Einheit B das Objekt O an, und Einheit A führt einen Workflow aus, der mit dem Ändern der Attribute von Objekt O beginnt. Am Ende der Ausführung ruft der Workflow die *update*-Methode des Objekts auf, um das Objekt auf der Serverseite zu aktualisieren. Wenn Einheit A und Einheit B dieselbe Instanz von Objekt B erhalten, werden für Einheit A in der Bestandsliste alle Änderungen angezeigt, die Einheit B ausführt, und das bereits vor dem Festschreiben der Änderungen auf der Serverseite. Wenn alles ordnungsgemäß abläuft, ist dies kein Problem. Wenn die Ausführung aber fehlschlägt, werden die Attribute von Objekt O nicht für Objekt A wiederhergestellt. In diesem Fall verwenden, wenn der Cache (die *find*-Vorgänge des Plug-Ins) einen Klon des Objekts anstelle immer wieder derselben Instanz zurückgibt, beide die Einheitsansichten und ändern ihre eigene Kopie, wodurch Parallelitätsprobleme zumindest innerhalb von Orchestrator vermieden werden.



### Benachrichtigen anderer Benutzer über Änderungen

Es können Probleme auftreten, wenn Sie einen Cache verwenden und gleichzeitig Objekte klonen. Das größte Problem ist, dass das Objekt, das Einheitsansichten verwendet, möglicherweise nicht die neueste Version ist, die für das Objekt verfügbar ist. Beispiel: Wenn eine Einheit die Bestandsliste anzeigt, werden die Objekte einmal geladen. Gleichzeitig zeigt aber die erste Einheit die Änderungen nicht an, wenn eine andere Einheit einige der Objekte ändert. Verwenden Sie zum Vermeiden dieses Problems die Methoden `PluginWatcher` und `IPluginPublisher` aus der Orchestrator-Plug-in-API, um andere Benutzer über die Änderungen zu benachrichtigen, damit andere Instanzen der Orchestrator-Clients die Änderungen sehen können. Dies gilt auch für eine eindeutige Instanz des Orchestrator-Clients, wenn Änderungen aus einem Objekt aus der Bestandsliste andere Objekte in der Bestandsliste beeinflussen und diese auch benachrichtigt werden müssen. Die Vorgänge, die häufig Benachrichtigungen verwenden, sind Hinzufügen, Aktualisieren und Löschen von Objekten, wenn diese Objekte, oder einige der Eigenschaften, in der Bestandsliste angezeigt werden.

### Aktivieren der Suche nach Objekten jederzeit und überall

Sie müssen die `find`-Methode der `IPluginFactory`-Schnittstelle implementieren, um Objekte nur über Typ und ID zu finden. Die `find`-Methode kann direkt nach dem Neustart von Orchestrator und dem Fortsetzen eines Workflows aufgerufen werden.

### Simulieren eines Abfragedienstes, wenn Sie keinen haben

Für den Orchestrator-Client ist es möglicherweise erforderlich, in bestimmten Fällen einige Objekte abzurufen oder sie nicht als Baumstruktur sondern beispielsweise als Liste oder Tabelle anzuzeigen. Dies bedeutet, dass Ihr Plug-In in der Lage sein muss, zu jeder Zeit Objektsätze abzufragen. Wenn die Drittanbietersoftware einen Abfragedienst anbietet, müssen Sie diesen Dienst anpassen und verwenden. Andernfalls sollte es Ihnen möglich sein, einen Abfragedienst zu simulieren, unabhängig von der höheren Komplexität oder niedrigeren Leistung der Lösung.

### Suchmethoden dürfen keine Laufzeitausnahmen zurückgeben

Die Methoden der `IPluginFactory`-Schnittstelle, die Suchvorgänge innerhalb des Plug-Ins implementieren, dürfen keine gesteuerten oder nicht gesteuerten Laufzeitausnahmen zurückgeben. Dies kann zu unerwarteten Fehlschlägen des Typs *Validierungsfehler* führen, wenn ein Workflow ausgeführt wird. Beispiel: Zwischen zwei Knoten eines Workflows wird die Methode `find` aufgerufen, wenn eine Ausgabe aus dem ersten Knoten eine Eingabe im zweiten Knoten ist. Zu diesem Zeitpunkt erhalten Sie, falls das Objekt aufgrund einer Laufzeitausnahme nicht gefunden wird, keine weiteren Informationen als einen *Validierungsfehler* im Orchestrator-Client. Danach hängt es davon ab, wie das Plug-In die Ausnahmen protokolliert, ob mehr oder weniger Informationen in den Protokolldateien enthalten sind.

### Workflow-interne Ansätze

Sie können einen Workflow implementieren, um langfristige Vorgänge zu überwachen, die das Orchestrator-Plug-In durchführt.

Sie können einen Workflow zum Überwachen von langfristig ausgeführten Vorgängen wie der Aufgabenüberwachung implementieren. Dieser Workflow kann auf Orchestrator-Auslösern und Warteereignissen basieren. Sie müssen beachten, dass ein Workflow, der beim Warten auf eine Aufgabe blockiert ist, fortgesetzt werden kann, sobald der Orchestrator-Server gestartet wird. Das Plug-In muss alle erforderlichen Informationen abrufen können, um den Überwachungsprozess ordnungsgemäß fortzusetzen.

Der Überwachungsworkflow oder die Aufgabe, die dieser intern verwenden kann, müssen einen Mechanismus bereitstellen, um die Abrufrate und eine mögliche Zeitüberschreitung anzugeben.

Der Debugging-Vorgang eines Codes zur Skripterstellung innerhalb eines Workflows ist nicht einfach, insbesondere wenn der Code keinen Java-Code abrufen. Aus diesem Grund ist es manchmal die einzige Option, die Protokollierungsmethoden zu verwenden, die von den standardmäßigen Orchestrator-Skriptobjekten bereitgestellt werden.

## Workflows und Aktionen

Sie können bestimmte bewährte Vorgehensweisen verwenden, um die Entwicklung und Verwendung von Workflows zu vereinfachen.

### Entwickeln Sie Workflows als Bausteine

Ein Baustein kann ein einfacher Workflow sein, der wenige Eingabeparameter benötigt und eine einfache Ausgabe zurückgibt. Wenn Sie eine Vielzahl an Bausteinen haben, können Sie hochrangigere Workflows einfach erstellen und einen besseren Toolsatz zum Erstellen von komplexen Workflows bereitstellen.

### Erstellen von hochrangigen Workflows basierend auf kleineren Komponenten

Wenn Sie einen komplexen Workflow mit mehreren Eingaben und internen Schritten entwickeln müssen, können Sie diesen in kleinere und einfachere zusammengesetzte Workflows und Aktionen aufteilen.

### Erstellen Sie Aktionen, wo möglich

Sie können Aktionen erstellen, um zusätzliche Flexibilität beim Entwickeln von Workflows zu erhalten.

- Zum einfachen Erstellen komplexer Objekte oder Parameter für Skripterstellungsmethoden
- Zum Vermeiden von ständigen Wiederholungen in allgemeinen Codeabschnitten
- Zum Ausführen von Benutzeroberflächen-Validierungen

### Workflows sollten nach Möglichkeit Aktionen aufrufen

Aktionen können direkt als Knoten innerhalb des Workflowschemas aufgerufen werden. Dadurch können Workflowschemas einfach gehalten werden, da Sie keine Skript-Codeabschnitte hinzufügen müssen, um eine einzelne Aktion aufzurufen.

### Füllen Sie die erwarteten Informationen aus

Geben Sie Informationen zu allen Elementen eines Workflows oder einer Aktion an.

- Geben Sie eine Beschreibung des Workflows oder der Aktion an.
- Geben Sie eine Beschreibung der Eingabeparameter an.
- Geben Sie eine Beschreibung der Ausgaben an.
- Geben Sie eine Beschreibung der Attribute für die Workflows an.

### Halten Sie die Versionsinformationen auf dem neuesten Stand

Wenn Sie Versionen von Plug-Ins erstellen, fügen Sie aussagekräftige Kommentare mit Informationen zu wichtigen Updates am Plug-In, wichtigen Implementierungsdetails usw. hinzu.

## Workflowpräsentation

Wenn Sie die Präsentation eines Workflows erstellen, müssen Sie bestimmte Strukturen und Regeln anwenden.

Verwenden Sie die folgenden Eigenschaften für Workfloweingaben in der Workflowpräsentation.

**Tabelle 6-46.** Eigenschaften für Workfloweingaben

Eigenschaften	Nutzung
Show in Inventory	Verwenden Sie diese Eigenschaft, um dem Benutzer beim Ausführen eines Workflows aus der Bestandslistenansicht zu helfen.
Specify a root object to be shown in the chooser	Verwenden Sie diese Eigenschaft, um dem Benutzer beim Auswählen von Eingaben zu helfen. Wenn das Root-Objekt in der Präsentation aktualisiert werden kann, ein Attribut ist oder von einer Objektmethode abgerufen wird, müssen Sie eine angemessene Aktion erstellen oder festlegen, um das Objekt in der Präsentation zu aktualisieren.
Maximum string length	Verwenden Sie diese Eigenschaft für lange Zeichenfolgen wie Namen, Beschreibungen, Dateipfade usw.
Minimum string length	Verwenden Sie diese Eigenschaft, um leere Zeichenfolgen aus den Testtools zu vermeiden.
Custom validation	Implementieren Sie komplexe Validierungen mit Aktionen.

Organisieren Sie die Eingaben mit Schritten und Anzeigegruppe. Diese Organisation unterstützt den Benutzer dabei, alle Eingabeparameter eines Workflows zu bestimmen und zu unterscheiden.

## Empfehlungen zur Entwicklung von Orchestrator-Plug-Ins

Indem Sie sich beim Entwickeln der verschiedenen Komponenten für Ihre Orchestrator-Plug-Ins an bestimmte Grundsätze halten, können Sie die Qualität der Plug-Ins verbessern.

**Tabelle 6-47.** Nützliche Verfahren bei der Plug-In-Implementierung

Komponente	Element	Beschreibung
Allgemein	Zugriff auf die Drittanbieter-API	Nach Möglichkeit sollten Plug-Ins vereinfachte Methoden bereitstellen, um auf die Drittanbieter-API zuzugreifen.
	Schnittstelle	Plug-Ins sollten eine kohärente und standardisierte Schnittstelle für Benutzer bereitstellen, auch wenn die API dies nicht tut.
Aktion	Skriptobjekte	Erstellen Sie Aktionen für sämtliche Erstellungs-, Änderungs- und Lösch- und sonstigen Methoden, die für ein Skriptobjekt verfügbar sind.
	Beschreibung	In der Beschreibung einer Aktion sollte erklärt werden, was die Aktion bewirkt, und nicht, wie sie funktioniert.
	Skripterstellung	Wenn Sie Skripterstellung verwenden, um die Eigenschaften oder Methoden eines Objekts abzurufen, können Sie überprüfen, ob sich der Objektwert von <code>null</code> oder <code>undefined</code> unterscheidet.
	Veraltung	Wenn eine Aktion veraltet ist, sollte die <code>comment-</code> oder <code>throw-</code> Anweisung die Ersatzaktion enthalten oder die Aktion selbst sollte eine neue Ersatzaktion aufrufen, damit Lösungen, die auf der veralteten Version der Aktion basieren, nicht fehlschlagen.
Workflow	Benutzeroberflächenvorgänge in der Orchestrator-gesteuerten Technologie	Erstellen Sie einen Workflow für jeden Vorgang, der in der Benutzeroberfläche der Orchestrator-gesteuerten Technologie verfügbar ist.
	Beschreibung	In der Beschreibung eines Workflows sollte erklärt werden, was der Workflow bewirkt, und nicht, wie er funktioniert.
	Präsentationseigenschaft <code>mandatory input</code>	Für alle obligatorischen Workfloweingaben müssen Sie die Eigenschaft <code>mandatory input</code> festlegen.

**Tabelle 6-47.** Nützliche Verfahren bei der Plug-In-Implementierung (Fortsetzung)

Komponente	Element	Beschreibung
	Präsentationseigenschaft default value	Wenn Sie einen Workflow entwickeln, der eine Einheit konfiguriert, sollten die Standardkonfigurationswerte für diese Einheit in die Workflowpräsentation geladen werden. Wenn Sie beispielsweise einen Workflow mit dem Namen „Hostkonfiguration“ entwickeln, müssen die Standardwerte der Hostkonfiguration in die Präsentation des Workflows geladen werden.
	Präsentationseigenschaft Show in inventory	Sie müssen die Eigenschaft Show in inventory so einstellen, dass Kontextworkflows für Bestandslistenobjekte vorhanden sind.
	Präsentationseigenschaft specify a root parameter	Verwenden Sie diese Eigenschaft in Workflows, wenn es nicht notwendig ist, die Bestandsliste vom Stamm der Baumstruktur aus zu durchsuchen.
	Workflowvalidierung	Sie müssen Workflows validieren und sämtliche Fehler korrigieren.
	Objekterstellung	Alle Workflows, die ein neues Objekt erstellen, sollten das neue Objekt als Ausgabeparameter zurückgeben.
	Veraltung	Wenn ein Workflow veraltet ist, sollte die comment- oder throw-Anweisung den Ersatzworkflow enthalten oder der veraltete Workflow sollte einen neuen Ersatzworkflow aufrufen, damit Lösungen, die auf vorherigen Versionen des Workflows basieren, nicht fehlschlagen.
Bestandsliste	Trennen der Hostverbindung	Wenn Ihre Bestandsliste eine Verbindung zu einem Host enthält und dieser Host nicht mehr verfügbar ist, geben Sie an, dass die Hostverbindung getrennt wurde. Dazu können Sie entweder das Stammobjekt umbenennen, indem Sie – disconnected anhängen, oder die Baumstruktur der Objekt unterhalb dieses Objekts entfernen, was auch das vCloud Director-Plug-In tut.
	Eigenschaft Select value as list	Ein Bestandslistenobjekt muss als treeview oder list auswählbar sein.
	Hostmanager	Wenn das Plug-In ein host-Objekt für das Zielsystem implementiert, sollte ein übergeordnetes hostmanager-Stammobjekt mit Eigenschaften zum Hinzufügen, Entfernen oder Bearbeiten von Hosteigenschaften vorhanden sein.
	Abrufen oder Aktualisieren von Objekten	Wenn ein Abfragedienst in der Orchestrator-gesteuerten Technologie ausgeführt wird, sollten Sie diesen zum Abrufen mehrerer Objekte verwenden.
	Erkennung untergeordneter Objekte	Wenn Sie untergeordnete Objekte separat abrufen müssen, muss der Abrufprozess mehrere Threads umfassen und darf nicht bei einem einzelnen Fehler blockieren.
	Orchestrator-Objektänderung	Alle Workflows, die den Zustand eines Elements in der Bestandsliste ändern können, müssen die Bestandsliste aktualisieren, damit die Synchronisierung der Objekte nicht verloren geht.

**Tabelle 6-47.** Nützliche Verfahren bei der Plug-In-Implementierung (Fortsetzung)

Komponente	Element	Beschreibung
Skriptobjekt	Externe Objektänderung	Sie können mithilfe eines Benachrichtigungsmechanismus auf Änderungen in der Orchestrator-gesteuerten Technologie hinweisen, die aus außerhalb von Orchestrator ausgeführten Vorgängen resultieren. Falls solche Vorgänge dazu führen, dass Objekte aus der Orchestrator-gesteuerten Technologie entfernt werden, müssen Sie die Bestandsliste entsprechend aktualisieren, um Fehler oder Datenverlust zu vermeiden. Wenn beispielsweise eine virtuelle Maschine aus vCenter Server gelöscht wird, aktualisiert das vCenter Server-Plug-In die Bestandsliste, um das Objekt der entfernten virtuellen Maschine zu entfernen.
	Finder-Objekt	Finder-Objekte sollten über Eigenschaften verfügen, mit deren Hilfe Objekte unterschieden werden können. In der Regel sind dies die Eigenschaften, die in der Benutzeroberfläche angezeigt werden.
	Implementierung	Die Methode <code>equals</code> muss implementiert werden, um sicherzustellen, dass der Vorgang <code>==</code> auf dasselbe Objekt angewendet wird, da das Objekt möglicherweise über zwei Instanzen verfügt.
	Plug-In-Objekteigenschaften	Für Objekte mit übergeordneten Objekten sollte eine <code>parent</code> -Eigenschaft implementiert werden.
	Plug-In-Objekteigenschaften	Für Objekte mit untergeordneten Objekten sollten GET-Methoden implementiert werden, die Arrays von untergeordneten Objekten zurückgeben.
	Bestandslistenobjekte	Bestandslistenobjekte sollten sich mit <code>Server.find</code> suchen lassen.  Alle Bestandslistenobjekte sollten serialisierbar sein, damit sie in einem Workflow als Eingabe- oder Ausgabeattribute verwendet werden können.
	Konstruktor und Methoden	In den meisten Fällen sollten skriptfähige Objekte entweder über einen Konstruktor verfügen oder von anderen Objektattributen bzw. Methoden zurückgegeben werden.
	Objekt-ID	Objekte mit einer von einem externen System ausgegebenen ID sollten eine interne ID verwenden, um sicherzustellen, dass keine IDs dupliziert werden, wenn Sie mehrere Server mit Orchestrator steuern.
	Suchen nach Objekten	In <code>search</code> - oder <code>find</code> -Methoden sollte ein Filter implementiert werden, damit der angegebene Name bzw. die angegebene ID gefunden werden kann, statt einfach alle Objekte zu finden. Der Orchestrator-Server verfügt beispielsweise über die Methode <code>Server.FindById</code> , mit der ein Plug-In-Objekt anhand seiner ID gefunden werden kann. Dazu muss die Methode für jedes auffindbare Objekt im Plug-In implementiert werden.
	Auslöser	Nach Möglichkeit sollten Auslöser für Objekte, die sich ändern, verfügbar sein, damit Orchestrator bei verschiedenen Ereignissen Richtlinien auslösen lassen kann. Um beispielsweise zu ermitteln, wann eine neue virtuelle Maschine hinzugefügt, eingeschaltet, ausgeschaltet usw. wird, kann Orchestrator einen Auslöser oder ein Ereignis im vCenter-Plug-In für das Datacenter-Objekt überwachen.

**Tabelle 6-47.** Nützliche Verfahren bei der Plug-In-Implementierung (Fortsetzung)

Komponente	Element	Beschreibung
	Objekteigenschaften	Objekte in anderen Plug-Ins sollten über Eigenschaften verfügen, die eine einfache Konvertierung von einem Plug-in-Objekt in ein anderes ermöglichen. Objekte von virtuellen Maschinen müssen beispielsweise über eine <code>moref</code> (Referenz auf verwaltetes Objekt) verfügen.
	Sitzungsmanager	Wenn Sie eine Verbindung zu einem Remoteserver herstellen, der eine andere Sitzung aufweisen kann, sollten im Plug-In eine gemeinsam genutzte Sitzung und eine Sitzung pro Benutzer implementiert werden.
Auslöser	Auslöser	Es sollte möglich sein, alle langen Vorgänge und blockierenden Methoden nach einer zurückgegebenen Aufgabe zu starten und bei Abschluss ein Auslöseereignis zu generieren.
Enumerationen	Enumerationen	Enumerationen für einen bestimmten Typ sollten über ein Bestandslistenobjekt verfügen, das ein Auswählen aus den verschiedenen Werten in der Enumeration ermöglicht.
Protokollierung	Protokolle	Für Methoden sollten verschiedene Protokollierungsebenen implementiert werden.
Versionierung	Plug-In-Version	Die Plug-In-Version sollte Standards entsprechen und zusammen mit dem Plug-In-Update aktualisiert werden.
API-Dokumentation	Methoden	Methoden, die in der API-Dokumentation beschrieben werden, sollten nie die Ausnahme <code>no xyz method / property</code> für ein Objekt zurückgeben. Stattdessen sollten Methoden <code>null</code> zurückgeben, wenn keine Eigenschaften verfügbar sind, und mit Details dokumentiert werden, wenn diese Eigenschaften nicht verfügbar sind.
	<code>vso.xml</code>	Alle Objekte, Methoden und Eigenschaften müssen in der Datei <code>vso.xml</code> dokumentiert werden.

## Dokumentierung von Plug-In-Benutzeroberflächen-Zeichenfolgen und -APIs

Wenn Sie Benutzeroberflächen-Zeichenfolgen für Orchestrator-Plug-Ins und die zugehörige API-Dokumentation schreiben, folgen Sie den allgemein anerkannten Stil- und Formatregeln.

### Allgemeine Empfehlungen

- Verwenden Sie die offiziellen Namen für VMware-Produkte, die im Plug-In involviert sind. Verwenden Sie beispielsweise offizielle Namen für die folgenden Produkte sowie VMware-Terminologie.

Korrekt	Nicht verwenden
vCenter Server	VC oder vCenter
vCloud Director	vCloud

- Beenden Sie alle Workflowbeschreibungen mit einem Punkt. Beispielsweise ist `Creates a new Organization.` eine Workflowbeschreibung.
- Verwenden Sie einen Texteditor mit einer Rechtschreibprüfung, um die Beschreibungen zu schreiben, und verschieben Sie sie dann in das Plug-In.
- Achten Sie darauf, dass der Name des Plug-Ins genau mit dem genehmigten Produktnamen des Drittanbieters, mit dem es verknüpft ist, übereinstimmt.

## Workflows und Aktionen

- Verfassen Sie informative Beschreibungen. Ein oder zwei Sätze genügen für die meisten Aktionen und Workflows.
- Hochrangige Workflows beinhalten möglicherweise umfangreichere Beschreibungen und Kommentare.
- Beginnen Sie Beschreibungen mit einem Verb, z. B. `Creates...`. Verwenden Sie keine auf sich selbst verweisende Sprache wie beispielsweise `This workflow creates`.
- Setzen Sie einen Punkt am Ende der Beschreibungen, die einen ganzen Satz darstellen.
- Beschreiben Sie, was ein Workflow oder eine Aktion macht, anstatt wie er/sie implementiert wird.
- Workflows und Aktionen sind normalerweise in Ordnern und Paketen enthalten. Beschreiben Sie auch diese Ordner und Pakete kurz. Beispielsweise kann ein Workflowordner eine Beschreibung wie die folgende haben: `Set of workflows related to vApp Template management`.

## Parameter von Workflows und Aktionen

- Beginnen Sie Beschreibungen für Workflows und Aktionen mit einer beschreibenden Nominalphrase, z. B. `Name of`. Verwenden Sie keine Phrasen wie `It's the name of`.
- Setzen Sie keinen Punkt am Ende von Beschreibungen für Parameter und Aktionen. Es sind keine vollständigen Sätze.
- Eingabeparameter von Workflows müssen eine Bezeichnung mit entsprechenden Namen in der Präsentationsansicht angeben. In vielen Fällen können Sie verwandte Eingaben in einer Anzeigegruppe zusammenfassen. Anstatt zwei Eingaben mit den Bezeichnungen „Name der Organisation“ und „Vollständiger Name der Organisation“ zu erstellen, können Sie beispielsweise eine Anzeigegruppe mit der Bezeichnung „Organisation“ erstellen und die Eingaben „Name“ und „Vollständiger Name“ in die Gruppe „Organisation“ platzieren.
- Fügen Sie für Schritte und Anzeigegruppen Beschreibungen oder Kommentare hinzu, die auch in der Workflowpräsentation angezeigt werden.

## Plug-In-API

- Die Dokumentation der API verweist auf die gesamte Dokumentation in der Datei `vso.xml` und in den Java-Quelldateien.
- Verwenden Sie für die Datei `vso.xml` dieselben Regeln für die Beschreibungen der Finder-Objekte und Skripterstellung-Objekte mit ihren Methoden, die Sie für Workflows und Aktionen verwenden. Beschreibungen von Objektattributen und Methodenparametern verwenden dieselben Regeln wie die Workflow- und Aktionsparameter.
- Vermeiden Sie Sonderzeichen in der Datei `vso.xml` und beziehen Sie die Beschreibungen innerhalb eines `<![CDATA[insert your description here!]]>`-Tags mit ein.
- Verwenden Sie den Javadoc-Standardstil für die Java-Quelldateien.





# Erstellen von Plug-Ins mit Maven

Die Orchestrator Appliance bietet ein Repository mit Maven-Artefakten, mit denen Sie Plug-In-Projekte aus Archetypen erstellen können.

Das Repository wird gehostet unter `https://Orchestrator_Server:8281/vco-repo/`, oder unter `http://Orchestrator_Server:8280/vco-repo/`, falls Ihre Maven-Version das HTTPS-Protokoll nicht unterstützt. Dieser Speicherort ist in der Datei `pom.xml` des standardmäßigen Orchestrator-Maven-Plug-In-Projekts eingebettet. Sie können nur auf das Repository zugreifen, wenn die Orchestrator Appliance bereitgestellt ist.

Dieses Kapitel behandelt die folgenden Themen:

- „Erstellen eines Orchestrator Plug-Ins mit Maven aus einem Archetyp“, auf Seite 281
- „Maven-Archetypen“, auf Seite 282
- „Best Practices für die Maven-basierte Plug-In-Entwicklung“, auf Seite 282

## Erstellen eines Orchestrator Plug-Ins mit Maven aus einem Archetyp

Sie können ein standardmäßiges Orchestrator-Maven Plug-In aus einem Archetyp erstellen, indem Sie Befehle über die Befehlszeilenschnittstelle ausführen.

### Voraussetzungen

- Vergewissern Sie sich, dass Sie Orchestrator Appliance 5.5.1 oder höher installiert haben.
- Vergewissern Sie sich, dass Sie Apache Maven 3.0.4 oder 3.0.5 installiert haben.

### Vorgehensweise

- 1 Erstellen Sie im interaktiven Modus ein Projekt, indem Sie einen Archetyp auswählen.

```
mvn archetype:generate -DarchetypeCatalog=https://Orchestrator_Server:8281/vco-repo/archetype-catalog.xml -DrepoUrl=https://Orchestrator_Server:8281/vco-repo -Dmaven.repo.remote=https://Orchestrator_Server:8281/vco-repo -Dmaven.wagon.http.ssl.insecure=true -Dmaven.wagon.http.ssl.allowall=true
```

---

**HINWEIS** Sie können nur dann auf das Maven-Repository zugreifen, wenn die Orchestrator Appliance bereitgestellt wurde.

---

- 2 (Optional) Wenn Sie nicht über HTTPS auf das Repository zugreifen können, können Sie dies über HTTP tun. Wenn Sie über HTTP auf das Repository zugreifen oder ein gültiges SSL-Zertifikat besitzen, können Sie ein Projekt erstellen, ohne das Flag `-Dmaven.wagon.http.ssl.allowall=true` zu benutzen.

```
mvn archetype:generate -DarchetypeCatalog=http://Orchestrator_Server:8280/vco-repo/archetype-catalog.xml -DrepoUrl=http://Orchestrator_Server:8280/vco-repo -Dmaven.repo.remote=http://Orchestrator_Server:8280/vco-repo -Dmaven.wagon.http.ssl.insecure=true
```

- 3 Navigieren Sie zum Projektverzeichnis und erstellen Sie das Plug-In.

```
cd Projekt_Verz && mvn clean install -Dmaven.wagon.http.ssl.insecure=true -Dmaven.wagon.http.ssl.allowall=true
```

Bei erfolgreichem Ablauf wird die Datei `.dar` des Plug-Ins im Verzeichnis `Target/` des DAR Moduls erstellt.

## Maven-Archetypen

Sie können einen Satz an vordefinierten Maven-Archetypen als Vorlagen zum Entwickeln von Orchestrator-Plug-Ins verwenden.

In der folgenden Tabelle werden die in Orchestrator verfügbaren standardmäßigen Maven-Archetypen beschrieben.

**Tabelle 7-1.** Standardmäßige Maven-Archetypen

Archetyp	Beschreibung
<code>com.vmware.o11n:o11n-plugin-archetype-simple</code>	<code>com.vmware.o11n:o11n-plugin-archetype-simple</code>
<code>com.vmware.o11n:o11n-package-archetype</code>	Ein ausschließlich inhaltsbezogenes Maven-Projekt, mit dem Pakete zur besseren Interaktion mit RCS, Vergleich, Nachbearbeitung usw. unverändert aufbewahrt werden können.
<code>com.vmware.o11n:o11n-client-archetype-rest</code>	Ein einfaches Befehlszeilenprogramm, das mit der Orchestrator-REST-API kommuniziert und einen Workflow aufruft.
<code>com.vmware.o11n:o11n-plugin-archetype-inventory</code>	Ein Plug-In, das die Verwendung von Bestandslisten demonstriert. Das Plug-In implementiert ein Repository, einen Adapter und eine Factory für einen einzelnen Typ. Die Bestandsliste wird in einer Datei auf einer Festplatte gespeichert.
<code>com.vmware.o11n:o11n-archetype-inventory-annotation</code>	Ein Plug-In, dessen Deskriptor <code>vso.xml</code> zusätzlich zu den Anmerkungen generiert wird.
<code>com.vmware.o11n:o11n-archetype-spring</code>	Ein Plug-In, das Spring-basiertes SDK verwendet, eine DI-fähige Umgebung bereitstellt und im Vergleich zu standardmäßigen Plug-In-APIs hochrangigere Dienste zur Verfügung stellt.
<code>com.vmware.o11n:o11n-plugin-archetype-modeldriven</code>	Ein Archetyp, der ein Plug-In-Skelett zum Erstellen von Plug-Ins mit ModelDriven generiert.

## Best Practices für die Maven-basierte Plug-In-Entwicklung

Sie können den Vorgang der Bereitstellung von Orchestrator-Plug-Ins, die mit Maven erstellt wurden, durch Ausführen eines Aufgabensatzes verbessern.

### Verwenden eines Repository-Managers

Wenn Sie Plug-Ins in einer größeren Organisation erstellen, verwenden Sie einen Enterprise-Repository-Manager, um das standardmäßige Orchestrator Appliance-Repository einzurichten, das als Proxy-Repository hinzugefügt werden soll. Das Verwenden eines zentralen Repositories verbessert das Management und die Plug-In-Projektzusammenarbeit. Wenn Sie das erste Build im neuen Repository abgeschlossen haben, speichert der Repository-Manager die Artefakte aus dem Orchestrator Appliance-Repository im Cache, und Sie können das Standard-Repository deaktivieren.

## Sperren von Workflows

Nachdem Sie überprüft haben, dass alle Workflows in Ihrem Plug-In ordnungsgemäß funktionieren, sperren Sie sie, um nicht autorisierte Änderungen zu verhindern. Durch das Sperren von Workflows stellen Sie sicher, dass die grundlegenden Funktionen des Plug-Ins nicht beeinträchtigt werden können. Wenn Benutzer einen Standardworkflow aus einem bestimmten Grund ändern, können sie eine Kopie des ursprünglichen Workflows erstellen und diese Kopie bearbeiten.

Übergeben Sie den Parameter `-DallowedMask=vf` an Maven, um Release-Builds mit gesperrten Workflows zu erstellen.

## Verwenden eines Paketsignaturzertifikats

Verwenden Sie ein selbstsigniertes Zertifikat oder ein Zertifikat, das von einer Zertifizierungsstelle signiert wurde, um die Integrität und Authentizität des Plug-Ins zu gewährleisten. Speichern Sie das Zertifikat im Keystore unter dem Alias `_dunesrsa_alias_`, indem Sie es mit dem Keytool in das JDK importieren.

Es gibt zwei Möglichkeiten zum Angeben eines Pfads zur Keystore-Datei und des Keystore-Kennworts.

- Definieren Sie die Befehlszeilenparameter `-DkeystoreLocation` und `-DkeystorePassword` für die Variable `MAVEN_OPTS`.
- Bearbeiten Sie die Datei `pom.xml`, um die Werte manuell einzugeben. Beispiel:

```
<keystore>path to the keystore file</keystore>
<storepass>keystore password</storepass>
```

Wenn kein Keystore importiert wird, wird die Datei `.package` mit der Datei `archetype.keystore` signiert.



# Index

## A

### Aktionen

- Allgemeine Richtlinien **212**
  - Attribute **213**
  - Benennen **212**
  - Bindung **154**
  - Codierungsrichtlinien **212**
  - erstellen **211**
  - Erstellen **171, 210**
  - Hinzufügen **211**
  - Parameter **213**
  - Suchen nach implementierenden Elementen **211**
  - Versionsverlauf **213**
  - Verwenden **209**
  - Wiederherstellen von gelöschten **214**
- Aktionselement **28**
- Aktionselemente, Bindung **152**
- Ansicht „Aktionen“ **210**
- Anzeigen **210**
- API-Dokumentation **101, 278**
- API-Explorer, Zugreifen **195**
- Attribute
- Definition **21, 151**
  - read-write-Eigenschaften **164, 187**
- Ausgabeparameter **13**
- Ausnahmebehandlung **43**
- Ausnahmebindungen, erstellen **43**
- Ausnahmenbindung **163**

## B

- Befehl Skripterstellungsklasse **199**
- Beispiel für einen einfachen Workflow
- Hinweise **149**
  - Zonen **149**
- Beispiel für einen Komplex Workflow
- Hinweise **174**
  - Zonen **174**
- Benutzerdefinierte Entscheidung-Element **28**
- Benutzerinteraktion-Element **28**
- Benutzerinteraktionen
- Attribute **108, 109**
  - Definieren externer Eingaben **113**
  - Elemente **108, 109**
  - Relative Zeitüberschreitung **111, 113**

- Benutzerinteraktionen, Antworten **116**
- Benutzerinteraktionen, Attribut „security.group“ **109**
- Benutzerinteraktionen, Attribut „time-out.date“ **110, 113**
- Benutzerinteraktionen, Ausnahmen **114**
- Benutzerinteraktionen, Dialogfeld „Eingabeparameter“ **115**
- Best Practices **89, 265**
- Betriebssystembefehle, Zugreifen **199**
- Bindung
- Aktionselemente **152**
  - Entscheidungselemente **152**
  - Skriptfähige Aufgaben **155**
- Bindungen
- Aktion **154**
  - Ausnahme **43, 163**
  - Definieren **38, 176, 177**
  - Skriptfähige Aufgaben **157**
- Boolesche Auswahl **42**

## D

- Datei vso.xml
- Definition **57, 234**
  - Elemente **72, 248**
- Dateisystem, System.getTempDirectory **198**
- Dateisystemzugriff **198**
- Debuggen eines Workflows, Beispiel **138**
- Debuggen von Workflows **137**
- Dialogfeld „Eingabeparameter“, Erstellen **166, 187**
- Dokumentation **143**

## E

- Eigenschaften
- Lesen/Schreiben **164**
  - Parameter **165**
- Eingabeparameter
- Angaben während der Ausführung **108, 109**
  - Definition **176**
  - Eigenschaften **104**
  - Einstellen von Eigenschaften **105**
  - Vom Benutzer erhalten **103**
- Eingabeparameter, abgefragt beim Benutzer **102**
- Elemente, Switch-Aktivität **48**
- Ende des Workflowelements **28**

Entscheidungselement, Bindungen **152**

Entscheidungselemente

    Löschen von Pfaden **41**

    Löschen von Zweigen **41**

    Verknüpfen **40**

Entwicklung von Workflows in einem Cluster **15**

Enumeration „HasChildrenResult“ **70, 246**

Erstellen von Workflows **16**

## F

Fehlerhandler, Elemente **44**

Foreach **45**

Foreach-Element **46, 47**

Fortsetzen einer fehlgeschlagenen Workflowausführung

    Aktivieren **142**

    Einrichten des Verhaltens **141**

    Zeitüberschreitung **142**

## G

Generieren, Workflow-Dokumentation **143**

Globaler Fehlerhandler, Elemente **44, 45**

## I

IDynamicFinder-Schnittstelle **62, 238**

IN-Bindungen **38**

IPluginAdaptor-Schnittstelle **53, 62, 230, 238**

IPluginEventPublisher-Schnittstelle **63, 239**

IPluginFactory-Schnittstelle **54, 64, 231, 240**

IPluginNotificationHandler **65, 241**

IPluginPublisher-Schnittstelle **66, 242**

## J

JavaScript, Dateisystemzugriff **198**

## K

Konfigurationselemente, Erstellen **133**

## M

Maven, Archetypen **282**

Mozilla Rhino-JavaScript-Engine, Einschränkungen **192**

## O

Orchestrator-API **193, 209**

Orchestrator-Client, Zugreifen **15**

OUT-Bindungen **38**

## P

Pakete

    Berechtigungen **222**

    Digital Rights Management **221**

Erstellen **221**

    Signatur **221**

Parameter

    Definition **21, 151, 176**

    Eigenschaften **165**

    Höherstufen **26**

    read-write-Eigenschaften **164**

Parametereigenschaften

    Dynamisch **104, 105**

    Statisch **104, 105**

PDF **143**

Plug-In

    Archetyp **281**

    Entwicklung **281**

    Erstellen **281**

    Typen **91, 268**

Plug-In-Adapter, Erstellen **53, 230**

Plug-In-API

    Enumeration „HasChildrenResult“ **70, 246**

    IDynamicFinder-Schnittstelle **62, 238**

    IPluginAdaptor-Schnittstelle **62, 238**

    IPluginEventPublisher-Schnittstelle **63, 239**

    IPluginFactory-Schnittstelle **64, 240**

    IPluginNotificationHandler **65, 241**

    IPluginPublisher-Schnittstelle **66, 242**

    PluginExecutionException **70, 246**

    PluginOperationException **70, 246**

    PluginTrigger **66, 242**

    PluginWatcher **67, 243**

    QueryResult **68, 244**

    ScriptingAttribute-Anmerkung **71, 247**

    ScriptingFunction-Anmerkung **71, 247**

    ScriptingParameter-Anmerkung **72, 248**

    SDKFinderProperty-Klasse **69, 245**

Plug-In-Entwicklung

    Best Practices **89, 265, 282**

    Bottom-Up **89, 266**

    Implementierung **94, 271**

    Top-down **90, 267**

Plug-In-Factory, Erstellen **54, 231**

Plug-In-Implementierung

    Aktionen **97, 274**

    Projektinterne Ansätze **95, 272**

    Projektstruktur **94, 271**

    Workflow-interne Ansätze **96, 273**

    Workflowpräsentation **97, 274**

    Workflows **97, 274**

Plug-In-Struktur **50, 226**

Plug-In-Typen

    Plug-Ins für Dienste **91, 268**

    Plug-Ins für Systeme **92, 269**

Plug-In-Zeichenfolgen **101, 278**

## Plug-Ins

- Adapter **53, 230**
- Architektur **49, 225**
- Benennen von Objekten **59, 235**
- DAR-Archiv **60, 237**
- Ereignishandler **56, 232**
- Erstellungsmethoden **89, 266**
- Factory **54, 231**
- Finder-Objekte **55, 231, 232**
- IAop-Schnittstelle **61, 237**
- Inhalt **56, 233**
- JAR-Dateien **60, 237**
- Komponenten **51, 228**
- Listener **56, 232**
- Richtlinien **56, 232**
- Richtlinienauslöser **56, 232**
- Richtlinienkontrollen **56, 232**
- Rolle der Datei vso.xml **53, 229**
- Struktur **56, 233**
- Teile eines Plug-Ins **49, 225**
- Verfügbarmachen von externer API **51, 228**
- vso.xml-Datei **60, 237**
- WAR-Datei **60, 237**
- Watcher **56, 232**
- WebConfigurationAdaptor-Schnittstelle **66, 242**
- Workflowauslöser **56, 232**
- Plug-Ins für Systeme
  - Objektorientierte Systeme **93, 269**
  - Ressourcenorientierte Systeme **93, 270**
- Plug-Ins, Skriptobjekte **55, 232**
- PluginExecutionException **70, 246**
- PluginOperationException **70, 246**
- PluginTrigger **66, 242**
- PluginWatcher **67, 243**
- Präsentation
  - Anzeigegruppen **103**
  - Eingabeschritte **103**
  - Erstellen **166, 187**
  - Erstellen von Anzeigegruppen **187**
- Präsentationen **18**

## Q

- QueryResult **68, 244**

## R

- Registerkarte „Präsentation“ **103–105, 187**
- Relatives Datumsobjekt **111, 128**
- Remoteworkflow
  - Abrufen **123**
  - Voraussetzungen **123**

## Ressourcenelemente

- aktualisieren **218**
- Anzeigen **215**
- Bearbeiten **216**
- Hinzufügen zu Workflows **218**
- Importieren **216**
- Speichern in Datei **217**

## S

## Schema

- Ausnahmepfad **34, 36**
- benutzerdefinierte Entscheidungen **40**
- Bindungen **37, 38**
- Datenfluss **37, 38**
- Entscheidungen **36, 40**
- Links **36**
- logischer Fluss **36**
- Logischer Fluss **34**
- Standardpfad **34, 36**
- Verknüpfungen **34**

Schemaelement, Eigenschaften **31**

## Schemaelemente

- Benutzerinteraktion **108, 109**
- Bindung **176, 177**
- Bindungen **38**
- Eigenschaften **32**
- Entscheidungen **42**
- Fehlerhandler **44**
- Globaler Fehlerhandler **44, 45**
- Standard-Fehlerhandler **44, 45**
- Verknüpfen **36**

Schemas **18**

- ScriptingAttribute-Anmerkung **71, 247**
- ScriptingFunction-Anmerkung **71, 247**
- ScriptingParameter-Anmerkung **72, 248**
- SDKFinderProperty-Klasse **69, 245**

## Skripterstellung

- Allgemeine Beispiele **202**
- API-Explorer **195**
- Ausnahmebehandlung **200**
- Automatische Vervollständigung **196**
- Beispiele **201**
- Dateisystembeispiele **205**
- E-Mail-Beispiele **204**
- Farbcodierung von Skriptschlüsselwörtern **197**
- Hinzufügen von Objekten **196**
- Hinzufügen von Parametern **197**
- JavaScript-Objekttypen **195**
- LDAP-Beispiele **205**
- Mozilla Rhino-JavaScript-Engine **192**
- Netzwerkbeispiele **206**

- Protokollierungsbeispiele **206**
- Skriptelemente **192**
- Workflowbeispiele **206**
- Zugreifen auf Betriebssystembefehle **199**
- Zugriff auf Java-Klassen **199**
- Zugriff auf Skriptmodul über Aktionen **194**
- Zugriff auf Skriptmodul über Richtlinien **194**
- Zugriff auf Skriptmodul über Workflow **194**
- Skripterstellungseingabe **193**
- Skriptfähige Aufgabenelemente, Bindung **155, 157**
- Skriptfähiges Aufgabenelement **28**
- Standard-Fehlerhandler, Elemente **44, 45**
- Start des Workflowelements **28**
- Suchen, Ändern von Suchergebnissen **27**
- Suchergebnisse **24**
- Switch-Aktivität, Schemaelemente **48**
- Systemeigenschaften **199**

## T

- Token **13**

## U

- Untergeordneter Workflow, Mehrfache Ausführung **46**

## V

- Verknüpfen
  - Entscheidungselemente **40**
  - Schemaelemente **36**
- Verschachtelte Workflows **124**
- verwenden **210**
- vso.xml
  - Architektur **58, 234**
  - Bestandselement **76, 252**
  - Eigenschaftenelement **77, 254**
  - Eigenschaftselement **78, 254**
  - Element „action“ **74, 86, 250, 262**
  - Element „attribute“ **84, 261**
  - Element „attributes“ **84, 260**
  - Element „code“ **86, 263**
  - Element „constructor“ **83, 260**
  - Element „constructors“ **83, 259**
  - Element „deprecated“ **73, 249**
  - Element „description“ **73, 249**
  - Element „entries“ **88, 265**
  - Element „entry“ **88, 265**
  - Element „enumeration“ **88, 264**
  - Element „enumerations“ **87, 264**
  - Element „events“ **80, 257**
  - Element „finder-datasource“ **75, 251**
  - Element „finder-datasources“ **75, 251**

- Element „finder“ **76, 253**
- Element „finders“ **76, 253**
- Element „gauge“ **81, 258**
- Element „parameters“ **83, 260**
- Element „trigger“ **80, 257**
- ID-Element **79, 256**
- Installationselement **74, 250**
- Konstruktor-Parameterelement **83, 260**
- Methodenelement **85, 261, 262**
- Methodenparameterelement **86, 263**
- Objektelement **82, 259**
- relation-Element **79, 255**
- relation-link-Element **80, 256**
- relations-Element **79, 255**
- scripting-objects-Element **82, 258**
- Singleton-Element **87, 264**
- trigger-properties-Element **81, 257**
- trigger-property-Element **81, 257**
- Untergeordnetes Bestandselement **80, 256**
- URL-Element **73, 250**
- vso.xml-Datei, Modulelement **72, 248**

## W

- Warte-Timer-Element **28**
- Warteereignis-Element **28**
- Workflow
  - Attribute **20, 21, 151**
  - Ausführen **167, 188**
  - einfachen Workflow erstellen **144**
  - Ende **139**
  - erstellen **146, 170**
  - Hinweise **149, 174**
  - Parameter **20, 21, 151**
  - Präsentation **18, 103**
  - Schema **18**
  - Validierung **167, 188**
  - Zonen **149, 174**
- Workflow „Workflows nacheinander starten“ **126**
- Workflow „Workflows parallel starten“ **126**
- Workflow-Debugger **137**
- Workflow-Dokumentation **143**
- Workflowattribute, Benennen **22**
- Workfloweditor
  - Öffnen **17**
  - Registerkarte „Allgemein“ **19**
  - Registerkarten **18**
- Workflowentwicklung **11**
- Workflowordner **16**
- Workflowparameter, Benennen **22**
- Workflowpräsentation, erstellen **103**
- Workflows
  - Asynchron **118, 121**



- Aufrufen anderer Workflows **117**
  - Ausführen **139, 140**
  - Ausführen für eine Auswahl von Objekten **125**
  - Ausführen in Workfloweditor **139**
  - Bearbeiten **17**
  - Bearbeiten von Standardworkflows **17**
  - Berechtigungen **134, 135**
  - Dateisystemzugriff **198**
  - Debuggen **137**
  - Debugging-Beispiel **138**
  - Eigenschaften der Eingabeparameter **106**
  - Entwicklungsphasen **15**
  - Erstellen **16**
  - Fortsetzen einer fehlgeschlagenen Workflowausführung **141, 142**
  - Geplant **118, 122**
  - komplexe Workflows entwickeln **169**
  - OGNL-Ausdruckswerte **107**
  - Standardbibliothek **17**
  - Starten **118**
  - Synchron **118, 120**
  - Testen **16**
  - Übertragen von Änderungen **119**
  - Validierung **135, 136**
  - Verschachtelt **118**
  - Versionsverlauf **143**
  - Verzweigung **42**
  - Weitergeben der Präsentation **120**
  - Weitergeben von Eingabeparametern **120**
  - Wiederherstellen von gelöschten **144**
  - Workflows mit langer Ausführungsdauer
    - Auslöseobjekt **127**
    - Auslöser **130**
    - Auslöserbasiert **132**
    - Datumsobjekt **127, 128**
    - Timer-basiert **129**
  - Workflows, reservierte OGNL-Schlüsselwörter **22**
  - Workflowschema
    - Anzeigen **24**
    - bearbeiten **24**
    - Bindungen **34**
    - Eigenschaften von Schemaelementen **31**
    - Elemente **24**
    - Erstellen **24, 147, 172**
    - Kopieren von Elementen **26**
    - Registerkarten für Eigenschaften von Schemaelementen **32**
    - Verknüpfungen **34**
  - Workflowschema, Elemente **28**
  - Workflowtoken
    - Attribute **139**
    - Checkpoints **139**
  - Workflowtoken-Attribute **14**
  - Workflowvalidierungstool **135**
- X**
- XML-Skripterstellung, E4X **202**
  - XPath-Ausdrücke **199, 200**
- Z**
- Zielgruppe **9**
  - Zusammengesetzter Typ **45, 47**

