# Unica Plan V12.1 Integration-Modul

# Inhalt

# Kapitel 1. Was sind die Unica Plan Integration Services?

Unica Plan Integration Services kombiniert die Unica Plan Integration Webservice, SOAP-API-Prozeduren und Trigger, um Geschäftskompetenzen zu erweitern.

Die Unica Plan Integration Services setzen sich aus folgenden Komponenten zusammen.

- **Unica Plan Integration Webservice**

  Integration Services bietet Unica Plan-Kunden und Professional Services eine Möglichkeit, Unica Plan in andere Anwendungen in ihrer Umgebung zu integrieren.
- **Unica Plan-Prozeduren und SOAP-API**

  Benutzerdefinierte Prozeduren können in Unica Plan definiert werden, um Unica Plan-Geschäftslogiken beliebig zu erweitern. Nachdem Sie Prozeduren definiert haben, können diese Prozeduren die Ziele für Aufrufe des Web-Service Integration Services aus anderen Anwendungen sein. Prozeduren können auch zum Senden von Nachrichten an andere Anwendungen definiert werden.
- **Unica Plan Trigger**

  Trigger können Ereignissen und Prozeduren in Unica Plan zugeordnet sein. Wenn eines dieser Ereignisse auftritt, wird der zugeordnete Trigger ausgeführt.

REST-APIs verwenden Unica Plan Integration Services nicht. Informationen zur REST-API finden Sie im  Unica Plan-Administratorhandbuch.

## Versionen und Abwärtskompatibilität

Zukünftige Versionen von Integration Services werden abwärtskompatibel mit allen untergeordneten Releases und Wartungsreleases mit gemeinsamer Hauptversionsnummer sein. behält sich jedoch das Recht vor, keine Kompatibilität mit früheren Versionen für Hauptreleases (x.0) zu gewährleisten, wenn die vorliegende Geschäftssituation bzw. die technischen Umstände dies rechtfertigen.

Die Hauptversionsnummer dieser API wird inkrementell erhöht, wenn eine der folgenden Änderungen vorgenommen wird.

- Änderungen der Dateninterpretation
- Änderung der Geschäftslogik (beispielsweise Änderungen in den Funktionen der Servicemethode)
- Änderungen der Methodenparameter und/oder der Rückgabetypen

Die Nebenversionsnummer dieser API wird inkrementell erhöht, wenn eine der folgenden Änderungen vorgenommen wird. Diese Änderungen sind per Definition mit einer früheren Version kompatibel.

- Neue Methode hinzugefügt
- Neuer Datentyp hinzugefügt und dessen Verwendung auf eine neue Methode beschränkt
- Neues Element zu einem Aufzählungstyp hinzugefügt
- Eine neue Version einer Benutzeroberfläche wird mit einem Versionssuffix definiert

## Authentifizierung

Eine Authentifizierung ist nicht erforderlich. Alle Clients sind einem bekannten Unica Plan-Benutzer mit dem Namen „PlanAPIUser“ zugeordnet. Ein Systemadministrator konfiguriert die Sicherheitsfunktionen dieses Benutzers mit Sonderberechtigung entsprechend den Anforderungen aller Web-Service-Clients.

## Ländereinstellung

Als Ländereinstellung wird ausschließlich die derzeit für die Unica Plan-Systeminstanz konfigurierte Ländereinstellung unterstützt. Es wird für alle von der Ländereinstellung abhängigen Daten, wie Nachrichten und Währungen, die Ländereinstellung des Systems vorausgesetzt.

## Statusverwaltung

API und Web-Service sind statusunabhängig. Die Serviceimplementierung speichert keine clientspezifischen Informationen für nachfolgende API-Aufrufe. Mit dieser Funktion wird eine effiziente Serviceimplementierung ermöglicht und die Clusterunterstützung vereinfacht.

### Datenbanktransaktionen

Unica Plan Integration Services legt dem Client keine Datenbanktransaktionen offen, verwendet aber entsprechende Informationen, sofern sie im Ausführungskontext miteinbezogen sind. Nach dem Start einer Transaktion sind die Auswirkungen aller API-Aufrufe innerhalb einer bestimmten Prozedur atomar. Das bedeutet, dass die Datenbank bei einem fehlgeschlagenen API-Aufruf in dem Status verbleibt, den sie vor dem Aufruf der API hatte. Andere Benutzer von Unica Plan können die Änderungen erst einsehen, nachdem die Prozedur die Transaktion erfolgreich ausgeführt hat.

API-Aufrufe, die die Datenbank aktualisieren, müssen zunächst eine Bearbeitungssperre anfordern, um zu verhindern, dass andere Benutzer die zugrundeliegenden Daten während der API-Aufrufe verändern. Andere Benutzer können gesperrte Komponenten nicht aktualisieren, bis der API-Aufruf abgeschlossen ist. Ebenso müssen die nächsten Unica Plan-Benutzer oder API-Clients die Datensperre anfordern, bevor ein weiterer API-Aufruf übergeben wird.

### Ereignisverarbeitung

Wenn eine Operation auf einer Unica Plan-Komponente über die API ausgeführt wird, werden dabei die gleichen Ereignisse generiert wie bei einer Ausführung der Operation durch einen Unica Plan-Benutzer. Benutzer, die bestimmte Benachrichtigungen abonniert haben, beispielsweise bei der Statusänderung eines Projekts, werden über Statusänderungen benachrichtigt, die aus API-Aufrufen sowie aus Aktionen von Benutzern folgen.

## Welche Voraussetzungen gelten für Unica Plan Integration Services?

Für Unica Plan Integration Services bestehen die folgenden Voraussetzungen.

Unica Plan Integration Services müssen folgende Voraussetzungen erfüllen:

- Systemintegration flexibel verbinden.
- Einen Mechanismus für Kundenanwendungen bereitstellen, Unica Plan über Web-Service-Aufrufe zu steuern.

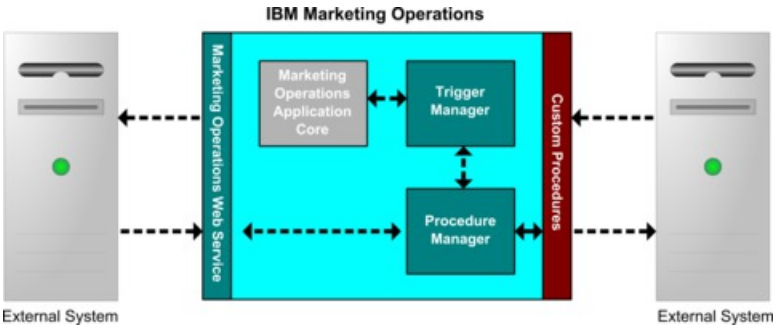
- Einen Mechanismus für Kundenanwendungen bereitstellen, über bestimmte Ereignisse in Unica Plan benachrichtigt zu werden.
- Ein einfaches Programmiermodell bereitstellen, das leicht verständlich und anwendbar ist.
- Bei der Wiederherstellung nach einer Störung stabil funktionieren.
- Datenintegrität garantieren.
- Integration mit vorhandenen grafisch orientierten Kunden von Unica Plan bei minimalen Auswirkungen.
- Differenzierten Zugriff auf Unica Plan-Komponenten bieten und dabei die zugrunde liegenden Implementierungsdetails vor Änderungen durch Programmierer schützen.

# Grundlegende Informationen zu Unica Plan Integration Services

Mithilfe von Unica Plan Integration Services erstellen Sie benutzerdefinierte Prozeduren. Durch diese Prozeduren können Sie externe Ereignisse auslösen, wenn bestimmte Ereignisse innerhalb von Unica Plan auftreten. Mit diesen Prozeduren können Sie Unica Plan-Funktionen aus externen Systemen oder Programmen ausführen.

Die API-Schnittstelle interagiert mit Unica Plan auf Programmebene wie die Benutzeroberfläche von Unica Plan auf Benutzerebene interagiert. Das Erstellen von Prozeduren erfolgt unter Verwendung der API. Mithilfe dieser Prozeduren ermöglichen Sie die Kommunikation zwischen Unica Plan und externen Systemen. Der Web-Service Unica Plan ist ein Containerobjekt für Prozeduren, API und Trigger.

Die Architektur der Unica Plan Integration Services wird hier dargestellt.

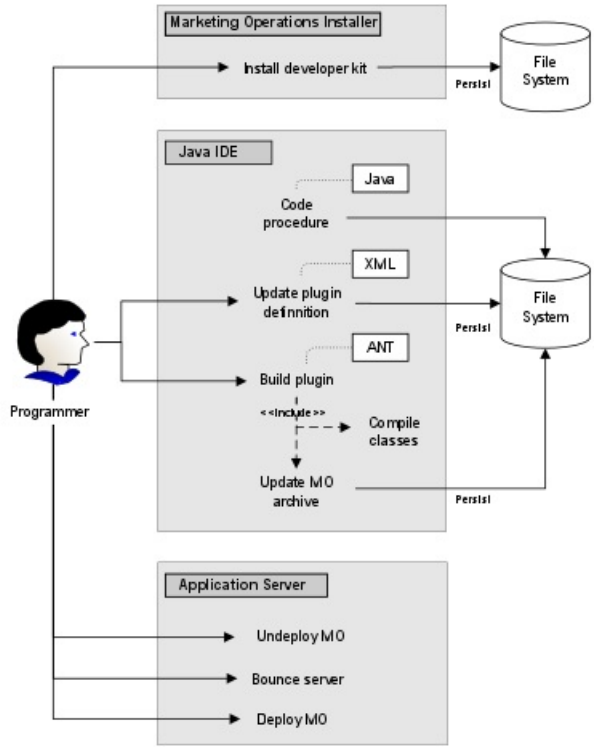



Die folgenden sind Schlüsselkomponenten der Integration Services.

- Unica Plan Prozedurmanager: erweitert die Geschäftslogik durch Interaktion mit Unica Plan über die API.
- Unica Plan Trigger-Manager: ordnet eine Bedingung (zum Beispiel die Statusänderung eines Marketingobjekts) einer Aktion zu (eine Prozedur, die ausgeführt wird, wenn die Bedingung für den Trigger erfüllt ist).

## Methoden

Die Komponenten von Unica Plan Integration Services werden, wie im folgenden Diagramm dargestellt, zur Entwicklung von benutzerdefinierten Prozeduren verwendet.




Führen Sie nach der Installation des Development-Kits die folgenden grundlegenden Schritte aus:

1. Geben Sie den Code für die benutzerdefinierte Prozedur ein.
2. Aktualisieren Sie die Plug-in-Definition in der XML-Definitionsdatei.
3. Erstellen Sie das Plug-in:
    a. Kompilieren Sie die erforderlichen Klassen.

b. Wenn Sie eine Bibliothek von Fremdanbietern verwenden, die nicht im Unica Plan-Archiv vorhanden ist, müssen Sie die Bibliothek in der Datei `plan.war` bündeln und die Bereitstellung erneut ausführen.

4. Starten Sie Unica Plan erneut. Änderungen an den Prozedurklassen werden angewendet, wenn Sie den Anwendungsserver erneut starten.

**Anmerkung:** Wenn Sie die Datei **plan.war** ändern, müssen Sie die Bereitstellung von Unica Plan zurücknehmen und die Bereitstellung mit der neuen Datei **plan.war** erneut ausführen. Wenn Sie eine Bibliothek von Fremdanbietern verwenden, die sich nicht im Unica Plan-Archiv befindet, und Sie die Datei **plan.war** ändern, müssen Sie die Bereitstellung von Unica Plan zurücknehmen und die Bereitstellung erneut ausführen.

## Einfaches Beispiel für die Kommunikation zwischen Unica Plan und der API

Im folgenden einfachen Beispiel wird die Einrichtung der Kommunikation zwischen API und Unica Plan beschrieben. Die Kommunikation in diesem Beispiel erfüllt keinen sinnvollen Zweck, es findet lediglich ein Umlauf zwischen Unica Plan und den Integration Services statt.

In diesem Beispiel werden Teile der Beispielprozeduren verwendet, die im Development-Kit für Unica Plan Integration Services enthalten sind. Sie finden die hier genannten Codebeispiele in den folgenden Dateien.

- `PlanClientFacade.java`
- `PlanWSNOOPTestCase.java`

Die `noop`-Methode ist ein Web-Service-Aufruf an Unica Plan. Sie ist in der PlanClientFacade-Klasse definiert, und übergibt Nullwerte in einem Array.

```
public ProcedureResponse noop(String jobId)
  throws RemoteException, ServiceException {
  NameValueArrays parameters =
    new NameValueArrays(null, null, null, null, null, null, null, null);
  return _serviceBinding.executeProcedure("uapNOOPProcedure", jobId,
 parameters);
```

```
}
```

Die Prozedur „testExecuteProcedure“ ruft die noop-Methode aus „PlanClientFacade“ auf, um einen Umlauf mit der Anwendung Unica Plan einzurichten.

```
public void testExecuteProcedure() throws Exception {
  // Time out after a minute
  int timeout = 60000;
  PlanClientFacade clientFacade = new PlanClientFacade(urlWebService,
 timeout);
  System.out.println("noop w/no parameters");
  long startTime = new Date().getTime();
  ProcedureResponse response = clientFacade.noop("junit-jobid");
  long duration = new Date().getTime() - startTime;

  // zero or positive status => success
  System.out.println("Status: " + response.getStatus());
  System.out.println("Duration: " + duration + " ms");
  assertTrue(response.getStatus() >= 0);
  System.out.println("Done.");
}
```

Ausführliche Informationen zu NameValueArrays, ProcedureResponse und anderen aufgelisteten Methoden und Datentypen finden Sie im Handbuch Unica PlanIntegration Module und in den JavaDocs.

## Installieren von Integration Services

Das Modul Unica Plan Integration Services ist eine separate, kostenpflichtige Komponente. Wenn Sie das Integration Services-Modul gekauft haben, müssen Sie es installieren.

1. Laden Sie das Installationsprogramm für Unica Plan Integration Services herunter.
2. Die Unica-Installationsprogramme erkennen das Integration Services-Modul.
3. Die Installationsprogramme legen Konfigurationseigenschaften auf **Einfach | umoConfiguration | integrationServices | enableIntegrationServices** fest. Sie können

die Installation durch Änderung der Konfigurationsparameter anpassen. Weitere Informationen hierzu finden Sie unter Konfigurationsparameter *(auf Seite 23)*.

## Inhalt des Software-Development-Kits

Das Software-Development-Kit enthält Dokumentationen mit allen publicapi-Klassen und -Schnittstellen sowie Mustercode.

Bei der SOAP-API werden alle Unica Plan Integration Services-Komponenten in einem Ordner mit dem Namen `devkits` installiert.

Der Mustercode wird in den folgenden Ordnern installiert.

- Der Ordner **build** enthält Scripts zum Erstellen und Bereitstellen von benutzerdefinierten Prozeduren.
- Der Ordner **Classes** enthält die kompilierten Prozedurklassen.

  Benutzer müssen die kompilierten Klassen ihrer benutzerdefinierten Prozeduren unter dem Pfad bereitstellen, der vom Konfigurationsparameter **integrationProcedureClasspathURL** angegeben wird. Anschließend werden sie vom Unica Plan-Prozedurmanager geladen, wie in der Konfigurationsdatei `procedure-plugins.xml` angegeben.
- Der Ordner **lib** enthält die erforderlichen Bibliotheken zur Entwicklung und Kompilierung von benutzerdefinierten Prozeduren.
- Der Ordner **src** enthält die Quellendateien der benutzerdefinierten Prozeduren. Benutzer können benutzerdefinierte Prozeduren, die als Trigger oder Web-Service gestartet werden sollen, hier speichern. Nur die SOAP-API unterstützt benutzerdefinierte Prozeduren.
  - Der Ordner **src/procedure** enthält die Konfigurationsdatei `procedure-plugins.xml`. Jede benutzerdefinierte Prozedur, die als ein ereignisbasierter Trigger oder ein externer Web-Service ausgeführt wird, muss einen Eintrag in dieser Datei haben. Die Einträge müssen einen vollständig qualifizierten Klassenpfad der Prozeduren und der erforderlichen Initialisierungsparameter enthalten.

- Der Ordner **src/procedure** enthält außerdem einige Beispielprozeduren, die im Lieferumfang von Unica Plan enthalten sind. Diese Prozeduren können verwendet werden, um benutzerdefinierte Prozeduren zu verstehen und zu entwickeln.

  Speichern Sie benutzerdefinierte Prozeduren im Verzeichnis **src** in einer neuen Ordnerstruktur, wie beispielsweise `com/<mycompany>/<mypackage>`. Speichern Sie benutzerdefinierte Prozeduren nicht im Ordner für Beispielprozeduren.
- Der Ordner **src/soap** enthält Beispiel-Web-Service-Clients, die in Java entwickelt werden. Verwenden Sie diese Beispiele als Ausgangspunkt, um Web-Service-basierte Clients für Integration Services zu entwickeln. Dieser Ordner enthält außerdem binäre Scripts, um einfache Clients über die Befehlszeile zu starten.

# Per Hosting bereitgestellte JavaDocs

Ausführliche Informationen über die öffentlichen API-Methoden finden Sie in der iPlanAPI-Klasse in den API-Dokumentationsdateien der JavaDocs.

Diese Dateien sind auf folgende Weise verfügbar:

- Über die Dateien im *<HCL_Unica>*`/`*<Plan_Home>*`/devkits/integration/javadocs`-Verzeichnis für die SOAP-API auf dem Server, der Unica Plan hostet.
- Indem Sie sich bei Unica Plan anmelden, auf einer beliebigen Seite **Hilfe > Produktdokumentation** auswählen und dann die Datei *<version>*`PublicAPI.zip` für die SOAP-API herunterladen.

# Dokumentation und Hilfe zu Unica Plan

Unica Plan wird in Ihrem Unternehmen von verschiedenen Personen für verschiedene Tasks verwendet. Informationen zu Unica Plan finden Sie in einer Reihe von Handbüchern, die jeweils für Teammitglieder mit bestimmten Zielen und speziellen Qualifikationsprofilen erstellt wurden.

In der folgenden Tabelle sind die Informationen, die in den einzelnen Handbüchern verfügbar sind, aufgelistet.

**Tabelle 1. Handbücher in der Gesamtdokumentation zu Unica Plan**

**In der folgenden dreispaltigen Tabelle werden in der ersten Spalte Tasks, in der zweiten Spalte Handbuchnamen und in der dritten Spalte die Zielgruppen beschrieben.**

| Wenn Sie | Siehe | Zielgruppe |
|---|---|---|
| • Projekte planen und verwalten<br>• Workflowtasks, Eckdaten und Personal erstellen<br>• Projektausgaben überwachen<br>• Prüfungen und Freigaben zum Inhalt abrufen<br>• Berichte erstellen<br>• Aufgaben und Checklisten erstellen | Unica PlanBenutzerhandbuch | • Projektleiter<br>• Designer<br>• Marketing-Manager für Direktmailing<br>• Vermarkter |
| • Vorlagen, Formulare, Attribute und Metriken entwerfen<br>• Benutzerschnittstelle anpassen<br>• Ebenen des Benutzerzugriffs und der Sicherheit definieren<br>• Zusatzfunktionen implementieren<br>• Unica Plan konfigurieren und optimieren | Unica PlanAdministratorhandbuch | • Projektleiter<br>• IT-Administratoren<br>• Implementierungsberater |
| • Marketingkampagnen erstellen<br>• Angebote planen<br>• Integration zwischen Unica Plan und | Unica Plan und Integrationshandbuch | • Projektleiter<br>• Fachleute für Marketingumsetzung<br>• Direktmarketing-Manager |

| Wenn Sie | Siehe | Zielgruppe |
| --- | --- | --- |
| Unica Campaign implementieren<br>• Integration zwischen Unica Plan und IBM Digital Recommendations implementieren<br>• Informationen zu neuen Systemfeatures<br>• Bekannte Probleme und deren Umgehung recherchieren | Unica Plan Freigeben Notes® | Alle Benutzer von Unica Plan |
| • Unica Plan installieren<br>• Unica Plan konfigurieren<br>• Upgrade auf eine neue Version von Unica Plan | Unica PlanInstallationshandbuch | • Berater für Softwareimplementierungen<br>• IT-Administratoren<br>• Datenbankadministratoren |
| Benutzerdefinierte Verfahren zur Integration von Unica Plan in andere Anwendungen erstellen | Unica Plan Integration Module und die API-JavaDocs, die verfügbar sind, nachdem Sie auf **Hilfe > Produktdokumentation** in Unica Plan klicken und dann die Datei `UnicaPlan<version>PublicAPI.zip` für die SOAP-API und `UnicaPlan<version>PublicAPI-RestClient.zip` für die REST-API herunterladen. | • IT-Administratoren<br>• Datenbankadministratoren<br>• Implementierungsberater |
| Informationen zur Struktur der Unica Plan-Datenbank | Unica Plan Systemschema | Datenbankadministratoren |

| Wenn Sie | Siehe | Zielgruppe |
| --- | --- | --- |
| Weiterführende Informationen in der Praxis | • Hilfe anfordern und das *-Benutzer- oder Administratorhandbuch* oder das -*Installationshandbuch* durchsuchen: Klicken Sie auf **Hilfe > Hilfe für diese Seite**<br>• Greifen Sie auf die Unica Plan-Handbücher zu: Klicken Sie auf **Hilfe > Produktdokumentation**<br>• Greifen Sie auf Handbücher für alle Unica-Produkte zu: Klicken Sie auf **Hilfe > Alle Unica Suite-Dokumentationen** | Alle Benutzer von Unica Plan |

# Kapitel 2. Unica Plan Integration Webservice

Der Web-Service stellt eine Clientansicht der Unica Plan Integration Services im Rahmen der Bereitstellung des Unica Plan-Servers zur Verfügung. Der Service wird gleichzeitig von Unica Plan-Webbenutzern verwendet.

Der Web-Service unterstützt einen API-Aufruf: executeProcedure.

Dieser Web-Service-Anruf wird von einem Client direkt ausgeführt.

## WSDL von Unica Plan Integration Services

Die Web Service Definition Language (WSDL) wurde manuell definiert und ist die ultimative Web-Service-Definition.

### Axis

Diese Web-Service-Version verwendet Axis2 1.5.2 zur Generierung der serverseitigen Klassen, aus denen die Web-Service-Implementierung in der WSDL-Datei besteht. Benutzer können jede beliebige Version von Axis oder auch ein anderes Verfahren als Axis verwenden, um eine clientseitige Implementierung für die Integration in die API aus der bereitgestellten WSDL zu erstellen.

### Protokollversion

Die Version des Protokolls ist wie folgt explizit an die WSDL gebunden:

- Als Teil des WSDL-Namens, zum Beispiel `PlanIntegrationService1.0.wsdl`
- Als Teil des WSDL-Zielnamensbereichs (targetNamespace), zum Beispiel `xmlns:tns="http://webservices.unica.com /MktOps/services/PlanIntegrationServices1.0?wsdl"`

### WSDL

Eine WSDL-Datei wird mit Unica Plan Integration Services bereitgestellt: `PlanIntegrationServices1.0.wsdl`. Die WSDL wird in das `integration/examples/soap/plan` Verzeichnis geliefert. Das Beispiel-Erstellungsscript verwendet

diese Datei bei der Generierung der erforderlichen clientseitigen Stubs für die Verbindung mit dem Web-Service.

# executeProcedure

executeProcedure ist der vom Web-Service unterstützte API-Aufruf.

## Syntax

```
executeProcedure(string key, string jobid, NameValueArrays paramArray)
```

## Rückgabe

```
int: status
Message[]: messages
```

## Beschreibung

Diese Methode ruft die angegebene Prozedur mit einem optionalen Array von Parametern auf. Der Aufruf wird synchron ausgeführt, das heißt, der Client wird blockiert und bei Abschluss des Aufrufs wird das Ergebnis ausgegeben.

## Parameter

**Tabelle 2. executeProcedure-Parameter**

| Name | Beschreibung |
|---|---|
| Schlüssel | Der eindeutige Schlüssel der auszuführenden Prozedur. Es wird ein *RemoteException*-Fehler zurückgegeben, wenn an **Schlüssel** keine Prozedur gebunden ist. |
| jobid | Optionale Zeichenfolge, die den Job ermittelt, der der Ausführung dieser Prozedur zugeordnet ist. Diese Zeichenfolge ist ein Durchgriffselement, sie kann jedoch verwendet werden, um Clientjobs an die Ausführung einer bestimmten Prozedur zu binden. |
| paramArray | Ein Array aus Parametern, das an die Prozedur übergeben werden soll. Es werden ein Fehlerstatus und eine Fehlernachricht zurückgegeben, wenn mindestens ein Parameter ungültig ist (beispielsweise ein falscher Typ |

| Name | Beschreibung |
|---|---|
| | oder ein unzulässiger Wert). Es ist dem Client überlassen, die Parameter, ihre Typen und die Anzahl der Werte zu bestimmen, die für die Prozedur erforderlich sind. |

### Rückgabeparameter

**Tabelle 3. Rückgabeparameter für „executeProcedure“**

| Name | Beschreibung |
|---|---|
| Status | Ein ganzzahliger Code:<br>• 0 gibt an, dass die Prozedur erfolgreich ausgeführt wurde<br>• eine ganze Zahl gibt einen Fehler an<br><br>Prozeduren können über den Status verschiedene Fehlerstufen angeben. |
| Nachrichten | Ein Array aus null oder mehr Nachrichtendatenstrukturen. Wenn **status** den Wert 0 hat, enthält dieses Array keine ERROR-Nachrichten, kann jedoch INFORMATION- und WARNING-Nachrichten enthalten.<br><br>Wenn **status** einen anderen Wert als 0 hat, können die Nachrichten aus einer beliebigen Kombination aus ERROR-, INFORMATION- und WARNING-Nachrichten bestehen. |

# Unica Plan Integration Webservice-Datentypen

Die vom Web-Service verwendeten Datentypen sind unabhängig von einer bestimmten Servicebindung oder Programmierimplementierung.

Dies erfolgt in folgender Schreibweise:

- <*type*>: <*type definition*> definiert einen einfachen Datentyp. Beispiel:

  Kennung: Zeichenfolge
- <*type*>: [ <*type definition*> ] definiert einen komplexen Datentyp oder eine Datenstruktur.
- <*type*>: { <*type definition*> } definiert einen komplexen Datentyp oder eine Datenstruktur.

Elemente, die einen komplexen Typ aufweisen, und API-Parameter können mithilfe dieser Typen Arrays deklarieren. Beispiel:

```
Handle [] handles
```

Der Typ „handles“ ist ein Array aus Handle-Typen.

## Primitive Typen

Primitive Typen sind auf Typen beschränkt, die in der nachfolgenden Tabelle definiert sind, um die Unterstützung von SOAP 1.1-Bindungen zu erleichtern. Alle Typen können als Arrays deklariert sein, zum Beispiel **String [ ]**. An sich können binäre Datentypen, wie **long**, von Protokollbindungen (z. B. SOAP) in Form von Zeichenfolgen dargestellt werden. Diese Darstellung hat jedoch keinerlei Auswirkungen darauf, wie die Semantik des Typs (z. B. gültige Werte) dem Client angezeigt wird.

**Tabelle 4. Primitive Typen**

| API-Typ | Beschreibung | SOAP-Typ | Java™ Typ |
|---|---|---|---|
| Boolesch | Boolescher Wert: **true** oder **false** | xsd:Boolean | Boolesch |
| DatumZeit | Ein Datum-Zeit-Wert | xsd:datetime | Datum |
| dezimal | Ein Dezimalwert mit beliebiger Genauigkeit | xsd:decimal | java.math.BigDecimal |
| double | Ein signierter Dezimalwert mit doppelter Genauigkeit | xsd:double | double |
| int | Ein signierter 32-Bit-Ganzzahlwert | xsd:int | int |
| Ganzzahl | Ein signierter Ganzzahlwert mit beliebiger Genauigkeit | xsd:integer | java.math.BigInteger |
| long | Ein signierter 64-Bit-Ganzzahlwert | xsd:long | long |
| Zeichenfolge | Eine Zeichenfolge aus Unicode-Zeichen | xsd:string | java.lang.String |

## MessageTypeEnum

```
MessageTypeEnum: { INFORMATION, WARNING, ERROR }
```

„MessageTypeEnum“ ist ein Aufzählungstyp, der alle möglichen Nachrichtentypen definiert.

- INFORMATION: eine Informationsnachricht
- WARNING: eine Warnnachricht
- ERROR: Eine Fehlernachricht

## Nachricht

```
Message: [MessageTypeEnum type, string code, string localizedText, string
 logDetail]
```

„Message“ ist eine Datenstruktur, die das Ergebnis eines Web-Service-API-Aufrufs definiert. Sie stellt optionale Felder für nicht lokalisierten Code, lokalisierten Text und Protokolldetails bereit. Derzeit wird in allen lokalisierten Texten die für die Unica Plan-Serverinstanz festgelegte Ländereinstellung verwendet.

**Tabelle 5. Nachrichtenparameter**

| Parameter | Beschreibung |
|---|---|
| Typ | „MessageTypeEnum“, legt den Typ der Nachricht fest. |
| Code | Ein optionaler Code für die Nachricht im Zeichenfolgeformat. |
| localizedText | Eine optionale Textzeichenfolge, die der Nachricht zugeordnet werden soll. |
| logDetail | Eine optionale Stack-Trace-Nachricht. |

## NameValue

```
NameValue: [string name, int sequence]
```

„NameValue“ ist ein komplexer Basistyp, der ein Name/Wert-Paar definiert. Dieser Typ definiert auch eine optionale Sequenz, durch die der Service nach Bedarf Werte-Arrays erstellt (die Sequenzen sind nullbasiert).

Alle „NameValues“ mit dem gleichen Namen und unterschiedlichen Folgenummern werden in ein Werte-Array konvertiert und dem gemeinsamen Namen zugeordnet.

Die Array-Größe wird durch die höchste Folgenummer bestimmt, nicht angegebene Array-Elemente haben Nullwerte. Die Array-Folgenummern müssen eindeutig sein. Der Wert und sein Typ werden durch den erweiterten Typ angegeben.

**Tabelle 6. Parameter von „NameValue“**

| Parameter | Beschreibung |
|---|---|
| Name | Eine Zeichenfolge, die den Namen eines NameValue-Typs definiert. |
| Reihenfolge | Eine nullbasierte ganze Zahl, die die Folgenummer für den durch „NameValue“ angegebenen Wert festlegt. |

Erweiterte NameValue-Typen werden wie folgt für die einzelnen primitiven Typen definiert:

**Tabelle 7. Erweiterte NameValue-Typen**

| Erweiterter Typ | Beschreibung |
|---|---|
| BigDecimalNameValue: NameValue [ Dezimalwert] | Ein NameValue-Typ, dessen Wert eine Dezimalzahl mit beliebiger Genauigkeit ist. |
| BigIntegerNameValue: NameValue [ Ganzzahlwert] | Ein NameValue-Typ, dessen Wert eine Ganzzahl beliebiger Größe ist. |
| BooleanNameValue: NameValue [ Boolescher Wert] | Ein NameValue-Typ, der einen booleschen Wert hat. |
| CurrencyNameValue: NameValue [ Zeichenfolge Ländereinstellung, Dezimalwert] | Ein NameValue-Typ, der zur Darstellung der Währung für eine Ländereinstellung geeignet ist. Die Ländereinstellung wird im ISO-Sprachencode angegeben, eine für die einzelnen Regionen in ISO-639 festgelegte Kombination aus zwei Kleinbuchstaben.<br><br>Derzeit muss die Ländereinstellung mit der für Unica Plan-Serverinstanz festgelegte Ländereinstellung übereinstimmen. |
| DateNameValue: NameValue [ Datum/Uhrzeit-Werts] | Ein NameValue-Typ, dessen Wert ein Datum ist. |

| Erweiterter Typ | Beschreibung |
|---|---|
| DecimalNameValue: NameValue [ Double-Wert] | Ein NameValue-Typ, dessen Wert eine Dezimalzahl mit doppelter Genauigkeit ist. |
| IntegerNameValue: NameValue [ langer Wert] | Ein NameValue-Typ, dessen Wert eine 64-Bit-Ganzzahl ist. |
| String NameValue: NameValue [ Zeichenfolgenwert] | Ein NameValue-Typ, dessen Wert eine Zeichenfolge ist. |

Ein Array aus erweiterten NameValue-Typen ist für die Verwendung definiert, wenn Sie eine Gruppe aus NameValue-Werten mit unterschiedlichen Typen definieren müssen.

```
     NameValueArrays: [
BooleanNameValue[]    booleanValues,
StringNameValue[]     stringValues,
IntegerNameValue[]    integerValues,
BigIntegerNameValue[] bigIntegooleanNameValue,
DecimalNameValue[]    decimalValues,
BigDecimalNameValue[] bigDecimalValues
DateNameValue[]       dateNameValues
CurrencyNameValue[]   currencyValues
     ]
```

# Kapitel 3. Unica Plan-Prozeduren

Eine „Prozedur“ ist eine benutzerdefinierte oder eine Java-Standardklasse, die von Unica Plan per Hosting bereitgestellt wird und eine Arbeitseinheit ausführt. Prozeduren bieten Kunden und Professional Services die Möglichkeit, -Geschäftslogiken auf beliebige Weise zu erweitern.

Um Auswirkung auf die von Unica Plan verwalteten Komponenten zu haben, folgen Prozeduren einem einfachen Programmiermodell mit einer klar strukturierten API. Prozeduren werden mithilfe eines einfachen Suchmechanismus und einer XML-basierten Definitionsdatei erkannt. Unica Plan führt die Prozeduren gemäß den Anforderungen der Clients aus. Beispielsweise als Reaktion auf eine Integrationsanforderung (eingehend) oder eine Initialisierung eines Triggers (intern oder ausgehend).

Prozeduren werden synchron mit dem Client ausgeführt. Ergebnisse werden dem Client direkt zur Verfügung gestellt und über einen persistenten Verfolgungsmechanismus. Die Ausführung einer Prozedur kann auch ein Auslösen weiterer Ereignisse und Trigger in Unica Plan mit sich bringen.

Prozeduren müssen in Java geschrieben sein.

## Voraussetzungen

Die Prozedurimplementierungsklassen sind als Paket in einer separaten Klassenstruktur bzw. einer JAR-Datei zusammengefasst und werden Unica Plan über einen URL-Pfad zur Verfügung gestellt.

### Prozedurimplementierung

Der Prozedurenausführungsmanager verwendet ein unabhängiges Klassenladeprogramm, um diese Klassen bei Bedarf zu laden. Unica Plan sucht standardmäßig im folgenden Verzeichnis.

`<Plan_Home>/devkits/integration/examples/classes`

Um diese Standardeinstellung zu ändern, setzen Sie den Parameter **integrationProcedureClasspathURL** unter **Einstellungen > Konfiguration > Plan > umoConfiguration > integrationServices**.

Der Name der Prozedurimplementierungsklasse entspricht der gültigen Java-Namenskonvention, um Konflikte der Paketbenennungen mit unica und Klassen anderer Anbieter zu vermeiden. Es ist wichtig, dass der Kunde Prozeduren nicht in der Paketstruktur „com.unica“ oder „com.unicacorp“ ablegt.

Die Prozedurimplementierung ist entsprechend der Java-Laufzeitversion codiert, die Unica Plan auf dem Anwendungsserver verwendet (JRE 1.8 oder höher).

Die Prozedurenimplementierungsklasse wird von der Richtlinie zum Laden von Klassen (meist **parent-last**) geladen, die normalerweise von Unica Plan verwendet wird. Der Anwendungsserver kann Entwicklungstools und Optionen zum erneuten Laden von Klassen bereitstellen, die für Unica Plan-Prozeduren gelten, dies ist jedoch nicht erforderlich.

## Bibliotheken

Unica Plan stellt eine Reihe von Open-Source-Bibliotheken und Bibliotheken von Fremdanbietern bereit. Anwendungsserver verwenden ebenfalls verschiedene Versionen dieser Bibliotheken.

Im Allgemeinen wird diese Liste von Release zu Release geändert. Folgende Bibliotheken anderer Anbieter werden unterstützt.

- `activation.jar`
- `axiom-api-1.2.15.jar`
- `axiom-compat-1.2.15.jar`
- `axiom-dom-1.2.15.jar`
- `axiom-impl-1.2.15.jar`
- `axis2-adb-1.5.2.jar`
- `axis2-adb-codegen-1.5.2.jar`
- `axis2-codegen-1.5.2.jar`
- `axis2-kernel-1.5.2.jar`
- `axis2-transport-http-1.5.2.jar`

- axis2-transport-local-1.5.2.jar
- httpcore-4.0.jar
- commons-codec.jar
- commons-httpclient-3.1.jar
- commons-lang.jar
- commons-logging.jar
- disruptor-3.4.2.jar
- geronimo-stax-api_1.0_spec-1.0.1.jar
- httpclient-4.3.6.jar
- httpcore-4.3.3.jar
- jersey-client-1.17.jar
- jersey-core-1.17.jar
- jersey-json-1.17.jar
- junit-4.4.jar
- log4j.jar
- log4j-api-2.8.2.jar
- log4j-core-2.8.2.jar
- mail.jar
- neethi-2.0.4.jar
- wsdl4j-1.6.2.jar
- xlxpScanner.jar
- xlxpScannerUtils.jar
- xlxpWASParsers.jar
- XmlSchema-1.4.3.jar
- Unica Plan APIs latest version (affinium_plan.jar)
- Unica Platform APIs latest version (unica-common.jar)

Wenn eine Prozedur oder die Sekundärklasse, die von der Prozedur importiert wird, solche Pakete verwendet, muss deren Verwendung exakt mit den Paketen übereinstimmen, die von Unica Plan oder dem Anwendungsserver bereitgestellt werden. In diesem Fall ist eine Nachbearbeitung Ihres Prozedurcodes erforderlich, wenn eine spätere Version von Unica Plan eine Bibliothek aktualisiert oder nicht weiter verwendet.

### Prozeduren und Threads

Die Prozedur muss bezüglich ihres eigenen Status threadsicher sein, das heißt, ihre Ausführungsmethode darf nicht von internen Statusänderungen zwischen einzelnen Aufrufen abhängen. Eine Prozedur kann keine eigenen Threads erstellen.

# Konfigurationsparameter

Bei der Installation von Unica Plan Integration Module legt das Installationsprogramm drei Konfigurationseigenschaften fest. Sie können die Konfigurationseigenschaften ändern, um das Verhalten von Integration Module anzupassen.

Sie finden die Konfigurationseigenschaften für Integration Module unter **Einfach | umoConfiguration | integrationServices**.

- Die Konfigurationseigenschaft **enableIntegrationServices** schaltet das Integration Services-Modul an und aus.
- Der Parameter **integrationProcedureDefinitionPath** enthält den vollständigen Dateipfad zur XML-Datei der benutzerdefinierten Prozedurdefinition.

  Der Standardwert ist `<HCL_Unica_Home><Plan_Home>/devkits/integration/examples/src/procedure/procedure-plugins.xml/`.
- Der Parameter **integrationProcedureClasspathURL** enthält die URL zum Klassenpfad für benutzerdefinierte Prozeduren.

  Der Standardwert ist `file:///<HCL_Unica_Home><Plan_Home>/devkits/integration/examples/classes/`.

  **Anmerkung:** Das Zeichen „/“ am Ende des Pfads `integrationProcedureClasspathURL` wird benötigt, um Prozedurklassen ordnungsgemäß zu laden.

# Design

Die Prozedurimplementierungsklasse verwendet die Unica Plan-API zum Lesen und Aktualisieren von Unica Plan-Komponenten, Aufrufen von Services und mehr. Andere Java-Pakete können zur Ausführung weiterer Tasks eingesetzt werden.

Konzentrieren Sie sich in Ihrem Design auf das Erstellen einer einzelnen Arbeitseinheit, die atomar bedient werden muss. Im Idealfall werden von einer Prozedur Serien von Tasks ausgeführt, die asynchron zur Ausführung zu einem späteren Zeitpunkt eingeplant werden können. Dieses „fire and forget“-Integrationsmodell sorgt für die geringste Last in beiden Systemen.

**Anmerkung:** Nur die dokumentierten Klassen und Methoden werden in zukünftigen Releases von Unica Plan unterstützt. Betrachten Sie alle anderen Klassen und Methoden in Unica Plan als nicht zulässig.

Nachdem Sie die Prozedurimplementierungsklassen codiert und kompiliert haben, stellen Sie diese Unica Plan zur Verfügung. Die Erstellungsscripts, die zusammen mit den Unica Plan Integration Services bereitgestellt werden, legen die kompilierten Prozeduren unter der Standardposition ab. Der abschließende Entwicklungsschritt besteht in der Aktualisierung der Plug-in-Definitionsdatei für die benutzerdefinierte Prozedur, die von Unica Plan zur Erkennung der benutzerdefinierten Prozeduren verwendet wird.

Die Prozedur muss die **com.unica.publicapi.plan.plugin.procedure.IProcedure**-Benutzeroberfläche implementieren und einen parameterlosen Konstruktor (normales JavaBeans-Modell) aufweisen. Die Codierung und Kompilierung der einzelnen Prozeduren erfolgt in einem beliebigen Java-Tool nach Wahl des Kunden, z. B. Eclipse, Borland JBuilder oder Idea. Beispielcode wird mit Unica Plan in Form von Developer Toolkits an folgender Speicherposition zur Verfügung gestellt:

`<Plan_Home>/devkits/integration/examples/src/procedure`

# Lebenszyklus von Prozeduren

Jede Prozedur durchläuft einen vollständigen Lebenszyklus.

Der Laufzeitlebenszyklus einer Prozedur umfasst die folgenden Schritte.

1. Erkennung und Initialisierung
2. Auswahl (optional)
3. Ausführung
4. Vernichtung

## Erkennung und Initialisierung

Für Unica Plan müssen alle standardmäßigen und benutzerdefinierten Prozeduren definiert werden, die für eine bestimmte Installationsinstanz zur Verfügung stehen. Dieser Prozess wird als Erkennung bezeichnet.

**Anmerkung:** Standardprozeduren (vom Unica Plan-Entwicklungsteam definierte Prozeduren) sind implizit bekannt, darum sind keinerlei Aktionen zur Erkennung dieser Prozeduren erforderlich.

Benutzerdefinierte Prozeduren werden in der Plug-in-Definitionsdatei der Prozedur definiert. Der Plug-in-Manager von Unica Plan liest diese Datei während der Initialisierung. Für jede erkannte Prozedur führt der Plug-in-Manager die folgenden Schritte aus.

1. Instanziieren der Prozedur, Überführung ihres Status in INSTANTIATED.
2. Erstellen eines Prüfungdatensatzes für die Prozedur.
3. Wenn die Prozedur instanziiert wurde, wird ihre **initialize()**-Methode mit allen Initialisierungsparametern aufgerufen, die in ihrer Plug-in-Beschreibungsdatei vorhanden sind. Wenn von dieser Methode eine Ausnahmebedingung ausgegeben wird, wird der Status protokolliert und die Prozedur abgebrochen. Andernfalls geht die Prozedur in den Status INITIALIZED über. Die Prozedur ist nun zur Ausführung bereit.
4. Erstellen eines Prüfungdatensatzes für die Prozedur.
5. Wenn die Prozedur initialisiert wurde, wird ihre **getKey()**-Methode aufgerufen, um den von Clients zum Verweis auf die Prozedur verwendeten Schlüssel zu bestimmen. Dieser Schlüssel wird der Instanz zugeordnet und für spätere Suchvorgänge gespeichert.

## Auswahl

Von Zeit zu Zeit zeigt Unica Plan Benutzern möglicherweise eine Liste der verfügbaren Prozeduren an, damit Administratoren beispielsweise einen Trigger einrichten können. Unica

Plan zeigt diese Liste erst an, nachdem die Prozedur mit den Methoden **getDisplayName()** und **getDescription()** der Prozedur initialisiert wurde.

## Ausführung

Nach der Initialisierung der Prozedur erhält Unica Plan eine Anforderung zur Ausführung der Prozedur. Dies kann zeitgleich mit weiteren Prozeduren (oder der gleichen Prozedur) in anderen Threads erfolgen.

Während der Laufzeit führt der Ausführungsmanager der Prozedur die folgenden Schritte aus.

1. Starten der Datenbanktransaktion.
2. Festlegen des Prozedurstatus auf EXECUTING.
3. Erstellen eines Prüfungdatensatzes für die Prozedur.
4. Aufrufen der **execute()**-Methode der Prozedur mit einem Ausführungskontext und allen Ausführungsparametern, die der Client zur Verfügung stellt. Bei der Methodenimplementierung wird bei Bedarf die Unica Plan-API verwendet, um Bearbeitungssperren anzufordern und den Ausführungskontext weiterzugeben. Wenn die Ausführungsmethode eine Ausnahme auslöst, markiert der Ausführungsmanager die Transaktion für den Rollback.
5. Entsprechend den Ausführungsergebnissen Commit oder Rollback für die Transaktion durchführen und den Prozedurstatus auf EXECUTED setzen.
6. Jegliche ausstehende Bearbeitungssperren auflösen.
7. Erstellen eines Prüfungdatensatzes für die Prozedur.

**Anmerkung:** Die **execute()**-Methode ist nicht dazu gedacht, Prozedurinstanzdaten zu ändern.

## Vernichtung

Wenn Unica Plan beendet wird, geht der Plug-in-Manager der Prozedur alle geladenen Prozeduren durch. Für jede erkannte Prozedur führt der Plug-in-Manager die folgenden Schritte aus.

1. Aufrufen der Destroy()-Methode der Prozedur, um der Prozedur eine Bereinigung zu ermöglichen, bevor die Instanz vernichtet wird.
2. Änderung des Status der Prozedur in FINALIZED (sie kann nicht ausgeführt werden).
3. Erstellen eines Prüfungdatensatzes für die Prozedur.
4. Vernichten der Instanz der Prozedur.

# Wichtige Java-Klassen

Das bereitgestellte Integration-Development-Kit enthält einen Satz Javadoc-Informationen für die öffentlichen Unica Plan-API-Klassen und unterstützende Klassen.

Die wichtigsten Java-Klassen sind hier aufgelistet.

- IProcedure (com.unica.publicapi.plan.plugin.procedure.IProcedure): Schnittstelle, die von allen Prozeduren implementiert werden muss. Prozeduren durchlaufen einen klar strukturierten Lebenszyklus und greifen auf die Unica Plan-API zu, um Arbeiten durchzuführen.
- ITriggerProcedure (com.unica.publicapi.plan.plugin.procedure.ITriggerProcedure): Schnittstelle, die von allen auslösenden Prozeduren implementiert werden muss (Markierungsschnittstelle).
- IExecutionContext (com.unica.publicapi.plan.plugin.procedure.IExecutionContext): Die Schnittstelle eines nicht transparenten Kontextobjekts, das vom Ausführungsmanager an die Prozedur übergeben wird. Dieses Objekt verfügt über öffentliche Methoden zur Protokollierung und zur Verwaltung von Bearbeitungssperren. Die Prozedur übergibt dieses Objekt auch an alle PlanAPI-Aufrufe.
- IPlanAPI (com.unica.publicapi.plan.api.IPlanAPI): Schnittstelle zur Unica Plan-API. Der Ausführungskontext stellt eine **getPlanAPI()**-Methode zum Abrufen der ordnungsgemäßen Implementierung bereit.

# Datensperre

Unica Plan verwendet ein pessimistisches Bearbeitungssperrschema, das bedeutet, es erhält immer nur ein Benutzer den Aktualisierungszugriff auf eine Komponenteninstanz.

Für den Benutzer der grafischen Benutzeroberfläche erfolgt diese Sperre auf der sichtbaren Benutzeroberfläche auf den Registerkarten. In manchen Fällen sind Daten eines Subsets einer Instanz gesperrt, beispielsweise eine Registerkarte mit Projektübersicht. In anderen Fällen sind Daten instanzenübergreifend gesperrt, beispielsweise die Registerkarte „Workflow“. Nachdem ein Benutzer eine Sperre angefordert hat, besteht für alle anderen Benutzer lediglich Lesezugriff auf die betreffenden Daten.

Um sicherzustellen, dass die von einer Prozedur an einer Komponenteninstanz oder einer Gruppe von Instanzen vorgenommenen Änderungen nicht versehentlich von einem anderen Benutzer überschrieben werden, muss eine Prozedur vor dem Aktualisieren von Komponentendaten die geeigneten Sperren anfordern. Zur Sperrung der Daten wird das an die **execute()**-Methode der Prozedur übergebene Ausführungskontextobjekt verwendet.

Bevor die Prozedur Daten aktualisiert, muss sie die **acquireLock()**-Methode des Kontexts für alle erforderlichen Sperren aufrufen. Wenn beispielsweise eine Prozedur ein Projekt und den zugehörigen Workflow aktualisiert, muss sie für beide eine Sperre anfordern.

Wenn ein anderer Benutzer bereits eine Sperre angefordert hat, löst die **acquireLock()**-Methode unverzüglich die Ausnahme **LockInUseException** aus. Um die Anzahl an Kollisionen zu minimieren, muss die Prozedur die Sperre aufheben, sobald alle Aktualisierungen vorgenommen wurden.

Der Ausführungsmanager hebt automatisch alle ausstehenden Sperren auf, nachdem die execute-Methode zurückgegeben wurde. Sperren bleiben grundsätzlich nur für die Lebensdauer der Datenbanktransaktion bestehen. Folglich verfallen Sperren, wenn das datenbankspezifische Transaktionszeitlimit überschritten ist.

**Anmerkung:** Bearbeitungssperren sind nicht das gleiche wie Datenbanktransaktionen.

# Prozedurtransaktionen

Der Prozedurausführungsmanager schließt die Ausführung der Prozedur automatisch in eine Datenbanktransaktion ein, indem er entsprechend dem Ergebnis der Prozedurausführung ein Commit oder ein Rollback durchführt.

Durch das Einschließen der Prozedurausführung in eine Datenbanktransaktion wird sichergestellt, dass Aktualisierungen der Unica Plan-Datenbank für andere Benutzer nicht sichtbar sind, bis das Commit erfolgt ist. Das Einschließen macht Aktualisierungen außerdem atomar.

Die Prozedurenschreibfunktion muss weiterhin die erforderlichen Bearbeitungssperren anfordern, um sicherzustellen, dass andere Benutzer keine Änderungen in die Datenbank schreiben können, bevor die Prozedurenausführung abgeschlossen ist.

# Prozedurkommunikation

Die Methode **execute()** einer Prozedur gibt einen Ganzzahlstatuscode an die Unica Plan-Prüfungtabelle der Prozedur zurück. Die Methode **execute()** einer Prozedur kann außerdem null oder mehrere Nachrichten an die Prüfungtabelle der Prozedur zurückgeben, die protokolliert und gespeichert werden.

Der Client kann die Statusinformationen auch auf andere Weise kommunizieren.

# Prozedurprotokollierung

Unica Plan hat eine separate Protokolldatei für Prozeduren: `<Plan_Home>\logs\system.log`

Der Prozedurenausführungsmanager protokolliert den Lebenszyklus jeder Prozedur und erstellt Prüfdatensätze.

- **logInfo()**: Schreiben einer Informationsnachricht in das Prozedurenprotokoll.
- **logWarning()**: Schreiben einer Warnnachricht in das Prozedurenprotokoll.
- **logError()**: Schreiben einer Fehlernachricht in das Prozedurenprotokoll.
- **logException()**: Erstellen eines Stack-Trace-Speicherauszugs für die Ausnahmebedingung im Prozedurenprotokoll.

# Prozedur-Plug-in-Definitiondatei

Die Plug-in-Definitionsdatei für die Prozedur definiert die Implementierungsklasse, die Metadaten und weitere Informationen zu den benutzerdefinierten Prozeduren, die in Unica Plan gehostet werden.

Die Plug-in-Definitionsdatei für eine Prozedur wird standardmäßig in folgendem Verzeichnis abgelegt:

`<Plan_Home>/devkits/integration/examples/src/procedures/procedure-plugins.xml`

Diese Datei ist ein XML-Dokument, das die nachfolgend beschriebenen Informationen enthält.

Prozeduren: eine Liste von null oder mehr **Procedure**-Elementen.

Prozedur: Ein Element, das eine Prozedur definiert. Jede Prozedur enthält die folgenden Elemente.

- **Schlüssel** (optional): Eine Zeichenfolge, die den Suchschlüssel für die Prozedur definiert. Dieser Schlüssel muss für alle Standardprozeduren sowie benutzerdefinierte Prozeduren, die von einer bestimmten Unica Plan-Instanz per Hosting bereitgestellt werden, eindeutig sein. Wenn keine Definition festgelegt wurde, dann wird standardmäßig die vollständig qualifizierte Version des Elements **className** verwendet. Namen, die mit der Zeichenfolge „uap“ beginnen, sind zur Verwendung durch Unica Plan reserviert.
- **className** (erforderlich): Der vollständig qualifizierte Paketname der Prozedurenklasse. Diese Klasse muss die IProcedure-Klasse implementieren (com.unica.public.plan.plugin.procedure.IProcedure).
- **initParameters** (optional): Eine Liste mit null oder mehr initParameter-Elementen.

  **initParameter** (optional): Parameter, der an die Methode „initialize()“ der Prozedur übergeben werden soll. Dieses Element umfasst den Namen des verschachtelten Parameters sowie seinen Typ und die Wertelemente.

  - Name: Eine Zeichenfolge, die den Parameternamen definiert.

- Typ: Der optionale Klassenname einer Java-Wrapper-Klasse, die den Typ des Parameterwerts definiert. Die folgenden Typen sind zulässig:
  - java.lang.String (Standardwert)
  - java.lang.Integer
  - java.lang.Double
  - java.lang.Calendar
  - java.lang.Boolean
- Wert: Eine Zeichenfolgeform des Attributwerts entsprechend seines Typs.

# Kapitel 4. Unica Plan-SOAP-API

Die Unica Plan-SOAP-API ist eine Anzeige, auf der eine Clientansicht einer aktiven Unica Plan-Instanz bereitgestellt wird.

Benutzern wird nur eine Untergruppe des Unica Plan-Leistungsspektrums angezeigt. Die API wird gleichzeitig von Unica Plan-Webbenutzern sowie von SOAP-Anforderungen und Triggern des Web-Service Unica Plan Integration Services verwendet. Die API unterstützt die folgenden Arten von Operationen.

- Komponentenerstellung und -löschung
- Erkennung (nach Komponententyp, Attributwert und sonstigen Werten)
- Komponentenprüfung (über zugehörige Attribute, spezialisierte Links und sonstige Werte)
- Komponentenänderung

**Anmerkung:** Unica Plan-APIs sind ausschließlich zur Verwendung durch Administratoren gedacht.

## Inhalt der Unica Plan-SOAP-API

Das Paket com.unica.publicapi.plan.api stellt die Unica Plan-SOAP-API bereit.

Dieses Paket bietet Schnittstellen und Ausnahmen und enthält folgende Typen für Klassen:

- aufgelistete Datentypen
- Kennungen zum Ermitteln von Objekt- und Komponenteninstanzen
- eine Java-Map, AttributeMap.

Sie können die vollständige Dokumentation der API einschließlich aller Methoden und möglicher Werte aufrufen, indem Sie auf **Hilfe > Produktdokumentation** in einer Instanz von Unica Plan klicken und anschließend die Datei HCL<version>PublicAPI.zip herunterladen.

# SOAP-API-Schnittstellen

Die Unica Plan-SOAP-API beinhaltet **IPlanAPI** und **IExecutionContext**.

Die Unica Plan-SOAP-API umfasst die folgenden Schnittstellen.

**IPlanAPI**

Definiert das öffentliche API für Unica Plan. Stellt Methoden zum Erstellen, Erkennen und Ändern von Objekten bereit, einschließlich Ordnern, Projekten, Programmen, Workflowtasks und Teammitgliedern.

Für Systeme, bei denen die optionale Integration in Unica Campaign aktiviert ist, werden auch Methoden zum Erstellen, Erkennen und Ändern von Angeboten zur Verfügung gestellt.

**IExecutionContext**

Definiert die Trigger und sperrt die Ausführmethoden im API.

## API-Methoden

Ausführliche Informationen über die öffentlichen API-Methoden finden Sie in der iPlanAPI-Klasse in den API-Dokumentationsdateien der JavaDocs.

Sie können auf diese Dateien zugreifen, indem Sie sich bei Unica Plan anmelden, auf einer beliebigen Seite **Hilfe > Produktdokumentation** auswählen und dann die Datei `<version>PublicAPI.zip` herunterladen.

# Allgemeine SOAP-API-Ausnahmen

Zu den allgemeinen von der SOAP-API ausgelösten Ausnahmen gehören NotFoundException, AuthorizationException, DataException, InvalidExecutionContextException und NotLockedException.

In der folgenden Liste wird erklärt, warum diese häufigen Ausnahmen auftreten können.

- <*object type*>NotFoundException: Das System kann das angegebene Element oder Objekt nicht liefern.

- AuthorizationException: Der Benutzer, der dem Ausführungskontext zugeordnet ist, ist nicht berechtigt, die angeforderte Operation auszuführen. Diese Ausnahme kann von einer beliebigen API-Methode ausgelöst werden, sie ist also nicht deklariert.
- DataException: In der zugrundeliegenden Datenbankebene in Unica Plan ist eine Ausnahme aufgetreten. Details können Sie dem SQL-Protokoll entnehmen.
- InvalidExecutionContextException: Es ist ein Problem mit einem Ausführungskontext aufgetreten, der an eine API-Methode übergeben wurde (beispielsweise wurde die Methode nicht richtig initialisiert). Diese Ausnahme kann von einem beliebigen API ausgelöst werden, sie ist also nicht deklariert.
- NotLockedException: Versuch der Aktualisierung von Komponentendaten, ohne vorher die erforderliche Sperre angefordert zu haben. Siehe acquireLock()-Methode der Schnittstelle „IExecutionContext“.

## SOAP-API-Handles

Ein Handle ist ein spezielles URL-Objekt, das innerhalb einer Unica Plan-Instanz auf eine bestimmte Objektinstanz verweist. Handles umfassen den Komponententyp, IDs von internen Daten und die Basis-URL einer Instanz.

Handles, die von dem API verwendet oder generiert werden, können zu einer vollständigen URL externalisiert werden. Sie können die resultierende URL auf verschiedene Arten verwenden. Sie können die URL verwenden, um eine Ansicht der Komponente auf der grafischen Benutzeroberfläche von Unica Plan zu öffnen, um diese Ansicht als E-Mail-Nachrichten zu senden oder um sie in einer anderen Prozedur als Parameter einzusetzen.

Handles sind lediglich für eine bestimmte Unica Plan-Serviceinstanz oder -Clusterinstanz gültig, sie sind aber immer für die Laufzeit des bereitgestellten Service gültig. Das bedeutet, Handles können zur späteren Verwendung in einer Datei gespeichert werden, sie können jedoch nicht für den Zugriff auf Komponenten einer anderen Instanz von Unica Plan verwendet werden. Diese Einschränkung gilt auch für Instanzen auf demselben physischen Hostserver. Unica Plan stellt einen Mechanismus zur Verfügung, mit dem verschiedene Basis-URLs der aktuellen Instanz zugeordnet werden können, um das Verlagern einer Instanz zu einem anderen Server zu ermöglichen (etwa bei Störungen im System).

Handles sind clientunabhängig. Beispielsweise kann ein Trigger ein Handle an eine Prozedur übergeben, die diesen als Parameter für einen SOAP-Aufruf an ein Drittanbietersystem verwendet. Dieses sendet daraufhin eine SOAP-Anforderung an Unica Plan zurück, um eine Prozedur aufzurufen, die ein Attribut aktualisiert.

Die Mitglieder der Handle-Klasse verfügen über Factory-Methoden zum Erstellen von Handles aus verschiedenen Typen von URLs. Nachfolgend sind Beispiele aufgeführt.

**Genehmigung**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=approvaldetail&
approvalid=101
```

**Asset**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=asset&
assetMode=VIEW_ASSET&assetid=101
```

**Assetordner**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=folder&id=101
```

**Assetbibliothek**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=library&id=101
```

**Anhang**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=attachmentview&
attachid=101&parentObjectId=101&parentObjectType=project
```

**Finanzkonto**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=accountdetails&
accountid=101
```

**Ordner**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=grouping_folder&
```

```
folderid=1234
```

### Rechnung

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=invoicedetails&
invoiceid=134
```

### Rechnungsposition

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=invoicedetails&
invoiceid=134&line_item_id=101
```

### Marketingobjekt

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=componenttabs&
componentid=creatives&componentinstid=1234
```

### Marketingobjektraster

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=componenttabs&
componentid=creatives&componentinstid=1234&gridid=grid
```

### Marketingobjekt-Rasterzeile

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=componenttabs&
componentid=creatives&componentinstid=1234&gridid=grid&gridrowid=101
```

### Planungsteam

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=teamdetails&
func=edit&teamid=100001
```

### Planbenutzer

```
http://mymachine:7001/plan/affiniumplan.jsp?
cat=adminuserpermissions&
func=edit&userId=101
```

### Programm

```
http://mymachine:7001/plan/affiniumplan.jsp?
cat=programtabs&programid=125
```

**Programmraster**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=programtabs&
programid=1234&gridid=grid
```

**Programmrasterzeile**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=programtabs&
programid=1234&gridid=grid&gridrowid=101
```

**Projekt**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=projecttabs&
projectid=1234
```

**Projektraster**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=projecttabs&
projectid=1234&gridid=grid
```

**Projektrasterzeile**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=projecttabs&
projectid=1234&gridid=grid&gridrowid=101
```

**Projektposition**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=projecttabs&
projectid=1234&projectlineitemid=123&projectlineitemisversionfinal=false
```

**Workflowstufe**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=projectworkflow&
projectid=1234&taskid=5678
```

**Workflowtask**

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=projectworkflow&
projectid=1234&taskid=5678
```

## SOAP-API-AttributeMap

Die Klasse AttributeMap ist eine Java-Map, die ausschließlich Attribute enthält. Das Attribut <Name> ist der Schlüssel des Zuordnungseintrags, und das Array des Attributs <values> (Plural!) ist der Wert des Zuordnungseintrags.

Die Klasse AttributeMap enthält die folgenden Felder.

- *<Name>*: der programmorientierte Name des Attributs. Dieser Name dient als eindeutiger Schlüssel für den Zugriff auf das Attribut innerhalb der Komponenteninstanz, in der es vorkommt.

  **Anmerkung:** *<Name>* entspricht nicht unbedingt dem Anzeigenamen, der auf der grafischen Benutzeroberfläche erscheint. Der Attributname von Komponenten, die mithilfe einer Vorlage erstellt wurden (z. B. Projekte oder Workflowtasks), wird in der Vorlagenelementdefinition festgelegt. Der Attributname muss eindeutig sein. Bei anderen Komponenten wird der Attributname in der Regel programmgesteuert von der serverseitigen Komponenteninstanz abgeleitet (z. B. durch Java-Introspektion).

  **Anmerkung:** Gemäß der Konvention umfassen kundenspezifische Attribute den Namen des Formats, in dem die bearbeitbare Version definiert ist: *<form_name>.<attribute_name>*.
- *Values*: ein Java-Objekt-Array, das null oder mehr Attribute enthält. Die einzelnen Werte müssen den gleichen Typ aufweisen und dem Typ des Attributs entsprechen, der in Unica Plan definiert ist. Es werden ausschließlich die folgenden Java-Wrapper und Unica Plan-Typen unterstützt:
  - AssetLibraryStateEnum ein AssetLibraryStateEnum-Aufzählungstypwert.
  - AssetStateEnum ein AssetStateEnum-Aufzählungstypwert.
  - AttachmentTypeEnum ein AttachmentTypeEnum-Aufzählungstypwert.
  - AttributeMap eine Zuweisung, die Attribute enthält.
  - BudgetPeriodEnum ein BudgetPeriodEnum-Aufzählungstypwert.

- BudgetTypeEnum ein BudgetTypeEnum-Aufzählungstypwert.
- Kennung: ein Verweis auf eine Komponenteninstanz, Rasterzeile, ein Attribut usw.
- InvoiceStateEnum ein InvoiceStateEnum-Aufzählungstypwert.
- java.io.File: Darstellung einer Datei.
- java.lang.Boolean ein boolescher Wert, entweder wahr oder falsch.
- java.lang.Double ein Dezimalzahlenwert mit doppelter Genauigkeit.
- java.lang.Float* ein Dezimalzahlenwert mit einfacher Genauigkeit.
- java.lang.Integer Ein 32-Bit-Ganzzahlwert
- java.lang.Long Ein 64-Bit-Ganzzahlwert
- java.lang.Object Generisches Java Objekt
- java.lang.String eine Zeichenfolge aus null oder mehr Unicode-Zeichen.
- java.math.BigDecimal Dezimalzahlenwert mit beliebiger Genauigkeit. Geeignet für Währung; die Interpretation des Wertes hängt von der Ländereinstellung für die Währung des Clients ab.
- java.math.BigInteger: Ganzzahlwert mit beliebiger Genauigkeit.
- java.net.URL: ein URL (Universal Resource Locator)-Objekt.
- java.util.ArrayList: Liste der Objekte.
- java.util.Calendar ein Datum/Uhrzeit-Wert für eine bestimmte Ländereinstellung.
- java.util.Date ein Datums-/Zeit-Wert. Dieser Typ wird nicht weiter unterstützt. Verwenden Sie stattdessen java.util.Calendar oder java.util.GregorianCalendar.

  **Anmerkung:** Zur Implementierung des Datums stehen dem Benutzer java.util.Calendar oder java.util.GregorianCalendar zur Verfügung.
- /java.util.GregorianCalendar: „GregorianCalendar“ ist eine konkrete Unterklasse von java.util.Calendar und stellt das Standardkalendersystem bereit, dass in den meisten Ländern verwendet wird.
- MonthEnum ein MonthEnum-Aufzählungstypwert.
- ProjectStateEnum ein ProjectStateEnum-Aufzählungstypwert.
- QuarterEnum ein QuarterEnum-Aufzählungstypwert.
- TaskStateEnum ein TaskstateEnum-Aufzählungstypwert.
- WeekEnum ein WeekEnum-Aufzählungstypwert.

Die Metadaten eines Attributs (wie übersetzter Anzeigename und Beschreibung) werden von der Vorlage, die dem Attribut zugeordnet ist, und dessen übergeordneter Objektinstanz definiert. Attribute stellen einen einfachen, dabei aber erweiterbaren Mechanismus zum Anzeigen von erforderlichen und optionalen Objektinstanzattributen bereit, beispielsweise Projektname, Code und Startdatum.

## SOAP-API - aufgelistete Datentypen

Die Unica Plan-SOAP-API unterstützt folgende aufgelistete Datentypen und Werte.

**ApprovalMethodEnum**

ApprovalMethodEnum definiert gültige Genehmigungsmethoden. Folgende Werte sind gültig:

- SEQUENTIAL
- SIMULTANEOUS

**ApprovalStateEnum**

ApprovalStateEnum definiert gültige Genehmigungsstatus. Folgende Werte sind gültig:

- CANCELLED
- ABGESCHLOSSEN
- IN_PROGRESS
- NOT_STATED
- ON_HOLD

**AssetLibraryStateEnum**

AssetLibraryStateEnum definiert gültige Assetbibliotheksstatus. Folgende Werte sind gültig:

- DEAKTIVIERT
- ENABLED

**AssetStateEnum**

AssetStateEnum definiert gültige Assetstatus. Folgende Werte sind gültig:

- ARCHIV
- DRAFT
- FINALIZE
- SPERREN

**AttachmentTypeEnum**

AttachmentTypeEnum definiert gültige Anlagentypen. Folgende Werte sind gültig:

- ASSET
- DATEI
- URL

**BudgetPeriodEnum**

BudgetPeriodEnum definiert die möglichen Planungsperioden. Folgende Werte sind gültig:

- ALLE
- MONTHLY
- QUARTERLY
- WEEKLY
- YEARLY

**BudgetTypeEnum**

BudgetTypeEnum definiert gültige Budgettypen. Folgende Werte sind gültig:

- ACTUAL
- ALLOCATED
- COMMITTED
- FORECAST

- TOTAL

**ComponentTypeEnum**

ComponentTypeEnum ermittelt die zugänglichen Unica Plan-Komponententypen. Folgende Werte sind gültig:

- GENEHMIGUNG
- ASSET
- ASSET_FOLDER
- ASSET_LIBRARY
- ATTACHMENT
- FINANCIAL_ACCOUNT
- GROUPING_FOLDER
- RECHNUNG
- MARKETING_OBJECT
- PLAN_TEAM
- PLAN_USER
- PROGRAM
- PROJECT
- PROJECT_REQUEST
- TASK
-

**InvoiceStateEnum**

InvoiceStateEnum definiert gültige Rechnungsstatus. Folgende Werte sind gültig:

- CANCELLED
- DRAFT
- PAID
- PAYABLE

**MonthEnum**

MonthEnum definiert gültige Werte für den Monat.

**OfferStateEnum**

OfferStateEnum definiert gültige Angebotsstatus. Folgende Werte sind gültig:

- STATE_OFFER_DRAFT
- STATE_OFFER_PUBLISHED
- STATE_OFFER_RETIRED

**ProjectCopyTypeEnum**

ProjectCopyTypeEnum definiert gültige Methoden zum Kopieren eines Projekts. Folgende Werte sind gültig:

- COPY_USING_PROJECT_METRICS
- COPY_USING_TEMMPLATE_METRICS

**ProjectParticipantLevelEnum**

ProjectParticipantLevelEnum gibt die Rollen an, die die Benutzer innerhalb eines Projekts haben. Folgende Werte sind gültig:

- OWNER
- PARTICIPANT
- REQUESTER

**ProjectStateEnum**

ProjectStateEnum definiert gültige Projekt- und Anforderungsstatus. Folgende Werte sind gültig:

- ACCEPTED
- CANCELLED
- COMPLETED
- DRAFT
- IN_PROGRESS

- IN_RECONCILIATION
- LATE: das Projekt wurde nicht zum geplanten Anfangsdatum gestartet.
- NOT_STARTED
- ON_HOLD
- OVERDUE: das Projekt wurde nicht vor dem geplanten Enddatum abgeschlossen.
- RETURNED
- SUBMITTED

Weitere Informationen über die Projekt- und Taskstatus finden Sie im Unica Plan-Benutzerhandbuch.

**QuarterEnum**

QuarterEnum legt die gültigen Werte für die Quartale fest: Q1, Q2, Q3 und Q4.

**TaskStateEnum**

TaskStateEnum definiert gültige Workflowtask-Status. Folgende Werte sind gültig:

- ACTIVE
- DEAKTIVIERT
- FINISHED
- PENDING
- SKIPPED

**WeekEnum**

WeekEnum definiert gültige Werte für die Wochen eines Jahres, von WEEK_1 bis WEEK_53.