IBM Unica Marketing Operations
Version 8.5.0 Publication Date: June 7, 2011

Integration Module

Copyright

Table of Contents

Preface

Contacting IBM Unica technical support

If you encounter a problem that you cannot resolve by consulting the documentation, your company's designated support contact can log a call with IBM Unica technical support. Use the information in this section to ensure that your problem is resolved efficiently and successfully.

If you are not a designated support contact at your company, contact your IBM Unica administrator for information.

Information you should gather

Before you contact IBM Unica technical support, you should gather the following information:

- A brief description of the nature of your issue.
- Detailed error messages you see when the issue occurs.
- Detailed steps to reproduce the issue.
- Related log files, session files, configuration files, and data files.
- Information about your product and system environment, which you can obtain as described in "System Information" below.

System information

When you call IBM Unica technical support, you might be asked to provide information about your environment.

If your problem does not prevent you from logging in, much of this information is available on the About page, which provides information about your installed IBM Unica applications.

You can access the About page by selecting **Help > About**. If the About page is not accessible, you can obtain the version number of any IBM Unica application by viewing the `version.txt` file located under each application's installation directory.

Contact information for IBM Unica technical support

For ways to contact IBM Unica technical support, see the IBM Unica Product Technical Support website: (http://www.unica.com/about/product-technical-support.htm).

1 What is Integration Services?

What is Marketing Operations Integration Services?

Marketing Operations Integration Services is a composite of the following.

- **Marketing Operations Integration Webservice**

  The Integration Services provide a way for IBM Unica Marketing Operations customers, partners and IBM Professional Services to integrate Marketing Operations with other applications running in their environment

- **Marketing Operations procedures and API**

  Custom procedures can be defined within Marketing Operations that extends Marketing Operations business logic in arbitrary ways. Once defined, these procedures can be the targets for the Integration Service webservice calls from other applications. Procedures also can be defined to send messages to other applications.

- **Marketing Operations triggers**

  Triggers can be associated with events and procedures in Marketing Operations. When one such event occurs, the associated trigger is executed.

What are the requirements for Marketing Operations Integration Services?

Marketing Operations Integration Services must:

- Loosely couple system integration

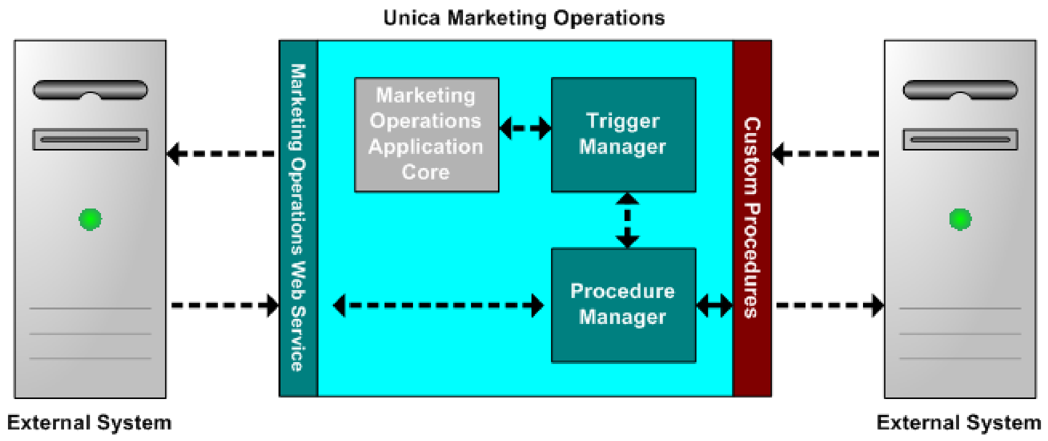
- Provide a mechanism for customer applications to affect Marketing Operations through webservice calls
- Provide a mechanism for customer applications to be notified of certain events in Marketing Operations
- Provide a simple, easy-to-understand and -use programming model
- Be robust when recovering from failure
- Guarantee data integrity
- Integrate with, and minimize the effect on, existing Marketing Operations GUI-based customers
- Provide fine-grain access to Marketing Operations components while insulating programmers from underlying implementation details

Getting started with IBM Unica Marketing Operations Integration Services

You use IBM Unica Marketing Operations Integration Services functionality to create custom procedures. You can use these procedures to trigger external events when certain events occur within Marketing Operations. You can use these procedures to perform Marketing Operations functions from external systems or programs.

You use the API to interface with IBM Unica Marketing Operations at the programmatic level, in the same way you use the GUI to interface with Marketing Operations at a user level. Using the API, you construct procedures. Using these procedures you communicate between Marketing Operations and external systems. The Marketing Operations Webservice is the container object for the procedures, API, and triggers.

The architecture of the Marketing Operations Integration Services is shown here.


Unica Marketing Operations
Marketing Operations Web Service
Marketing Operations Application Core
Trigger Manager
Procedure Manager
Custom Procedures
External System
External System

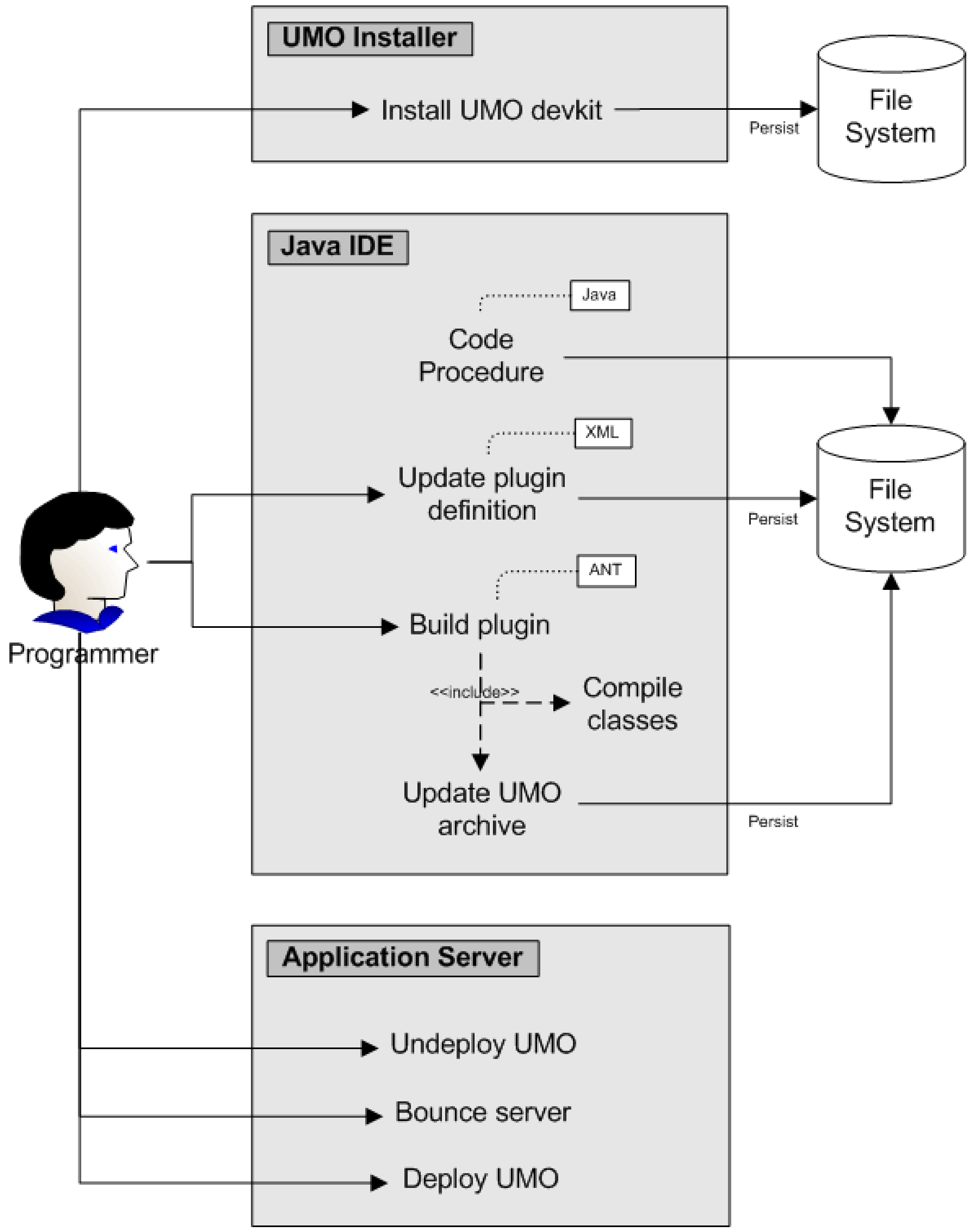
The following are key components of the Integration Services.

- Marketing Operations Procedure Manager: extends the business logic by interacting with Marketing Operations through the API.

- Marketing Operations Trigger Manager: associates a condition (for example the state change of a marketing object) with an action—a procedure to execute when the trigger's condition is met.

Methodology

You use the IBM Unica Marketing Operations Integration Service components to develop custom procedures, as shown in this diagram:



After using the Marketing Operations Installer to install the developer's kit, you follow these basic steps:

1. Code the custom procedure. Currently, you must use Java.
2. Update the plugin definition in the XML definition file.
3. Build the plugin:

   a. Compile the necessary classes.
   b. Update the Marketing Operations archive (the WAR file).

4. Undeploy and redeploy Marketing Operations, so it can recognize the new procedure.

Basic Example to communicate between IBM Unica Marketing Operations and the API

This section describes a basic example of establishing communication between the API and Marketing Operations. It does not do any useful work; it just performs a round trip betwen Marketing Operations and the Integration Services.

This section uses portions of the example procedures included with the Marketing Operations Integration Services developer's kit. Specifically, you can find the code referenced here in the following files.

- PlanClientFacade.java
- PlanWSNOOPTestCase.java

The noop method is a webservice call to Marketing Operations. It is defined in the PlanClientFacade class, and passes null values in an 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);
}
```

The procedure testExecuteProcedure calls the noop method from PlanClientFacade to establish a round-trip with the Marketing Operations application.

```
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.");
}
```

For details of NameValueArrays and ProcedureResponse, and other listed methods and data types, please refer to the specific sections that discuss them throughout the rest of the Integration Services documentation.

Hosted javadoc

For specific information about the public API methods, refer to the iPlanAPI class in the javadoc. The javadoc is available on IBM Unica Customer Central: http://customers.unica.com/.

2 About Integration Webservice

About Marketing Operations Integration Webservice

The webservice provides a client view of the Marketing Operations Integration Services.

The webservice provides a client view of the Marketing Operations Integration Services, which is part of the deployment of the IBM Unica Marketing Operations server. The service is designed to be used concurrently with Marketing Operations web users.

The webservice supports one API call, executeProcedure.

A client makes this webservice call directly.

Authentication

Authentication is not required; all clients are associated with a known IBM Unica Marketing Operations user named PlanAPIUser. We assume that the security capabilities of this special user are configured by a Marketing Operations system administrator to the needs all webservice clients.

A future version of the service may provide a more general mechanism for secure client authentication.

Locale

The only locale supported is the locale currently configured for the IBM Unica Marketing Operations system instance. All locale-dependent data accessible via the service (messages, currency, and so on) are assumed to be in the system locale.

A future version of the service may provide a mechanism for the client to tell Marketing Operations which locale to use.

State management

The webservice is *stateless*; no per-client information is saved by the service implementation across API calls. This provides for an efficient service implementation and simplifies cluster support.

Database transactions

The webservice does not expose database transactions nor edit locks to the client. It does, however, guarantee that the effects of any procedure execution will be *atomic*. This means the procedure either succeeds or fails; a failure leaves the database in the same state as if the API was never called at all.

Related Topics

- executeProcedure

About Marketing Operations Integration Webservice Data Types

This section defines the data types used by the webservice, independent of a particular service binding or programming implementation.

The following notation is used.

- <*type*>: <*type definition*> defines a simple data type. For example:

  Handle: string
- <*type*>: [<*type definition*>] defines a complex data type or a data structure.
- <*type*>: { <*type definition*> } defines a complex data type or a data structure.

Complex type elements and API parameters can make use of these types to declare arrays. For example:

```
Handle [] handles
```

The type, handles, is an array of Handle types.

Primitive types

Primitive types are restricted to those defined in the table below to simplify support for SOAP 1.1 bindings. All types can be declared as arrays, for example, **String [ ]**. Note that inherently binary data types such as **long**, may be represented as strings by a protocol binding (for example SOAP). This, however, has no effect on the semantics of the type, permissible values, etc., as seen by the client.

API Type	Description	SOAP Type	Java Type
boolean	Boolean value: **true**or **false**	xsd:boolean	boolean

API Type	Description	SOAP Type	Java Type
dateTime	A date time value	xsd:datetime	Date
decimal	An arbitrary-precision, decimal value	xsd:decimal	java.math.BigDecimal
double	A double-precision, signed, decimal value	xsd:double	double
int	A signed, 32-bit, integer value	xsd:int	int
integer	An arbitrary-precision, signed, integer value	xsd:integer	java.math.BigInteger
long	A signed, 64-bit, integer value	xsd:long	long
string	A string of Unicode characters	xsd:string	java.lang.String

MessageTypeEnum

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

MessageTypeEnum is an enumerated type that defines all possible message types.

- INFORMATION: an informational message
- WARNING: a warning message
- ERROR: an error message

Message

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

Message is a data structure that defines the result of a webservice API call. It provides optional fields for a non-localized code, localized text and log detail. Currently, all localized text uses the locale set for the IBM Unica Marketing Operations server instance.

Parameter	Description
type	A MessageTypeEnum, setting the type of the message.
code	An optional code, in string format, for the message.
localizedText	An optional text string to associate with the message.

Parameter	Description
logDetail	An optional stack trace message.

NameValue

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

NameValue is a base complex type that defines a name-value pair. It also defines an optional sequence that the service uses to construct value arrays as needed (the sequences are zero-based).

All NameValues with the same name, but different sequence numbers, will be converted into an array of values and associated with the common name.

The array size is determined by the maximum sequence number; unspecified array elements will have null values. Array sequence numbers must be unique. The value—and its type—are provided by the extended type.

Parameter	Description
name	A string defining the name of a NameValue type.
sequence	A zero-based integer setting the sequence number for the NameValue implied value.

Extended NaveValue types are defined for each primitive type, as follows:

Extended type	Description
BigDecimalNameValue: NameValue [decimal value]	A NameValue type whose value is an arbitrary-precision, decimal number.
BigIntegerNameValue: NameValue [integer value]	A NameValue type whose value is arbitrary sized integer.
BooleanNameValue: NameValue [boolean value]	A NameValue type whose value is a boolean.
CurrencyNameValue: NameValue [string locale, decimal value]	A NameValue type suitable for representing currency in some locale. Locale is an ISO Language Code, that is, the lower-case, two-letter codes as defined by ISO-639. Note that currently, the locale must agree with the locale set in the IBM Unica Marketing Operations server instance.
DateNameValue: NameValue [datetime value]	A NameValue type whose value is a date.
DecimalNameValue: NameValue [double value]	A NameValue type whose value is a double-precision, decimal number.

Extended type	Description
IntegerNameValue: NameValue [long value]	A NameValue type whose value is a 64-bit integer.
String NameValue: NameValue [string value]	A NameValue type whose value is a string.

And finally, an array of the extended NameValue types is defined for use when you need to define a set of NameValues of with different types.

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

Related Topics

- About Marketing Operations Integration Webservice
- executeProcedure

executeProcedure

Syntax

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

Returns

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

Description

This method invokes the specified procedure with an optional array of parameters. The call executes synchronously; that is, it blocks the client and returns the result upon completion.

Parameters

Name	Description
key	The unique key of the procedure to execute. A *RemoteException* error is returned if no procedure is bound to **key**.
jobid	Optional string identifying the job associated with this procedure execution. This is a pass-through item, but may be used to tie client jobs to the execution of a particular procedure.
paramArray	An array of parameters to pass to the procedure. An error status and message is returned if one or more of the parameters is invalid (the wrong type, an illegal value, and so on). It is up to the client to determine the parameters, their types, and the number of values required by the procedure.

Return Parameters

Name	Description
status	An integer code: • 0 indicates the procedure executed successfully • an integer indicates an error. Procedures may use the status to indicate different levels of errors.
messages	An array of zero or more message data structures. If **status** is 0, this array will never contain ERROR messages, but could contain INFORMATION and WARN-ING messages. If **status** is non-zero, **messages** may contain any mix of ERROR, INFORMATION and WARNING messages.

Marketing Operations Integration Service WSDL

This topic defines the Web Services Definition Lanaguage (WSDL) for the Marketing Operations Integration Service. The WSDL was defined by hand and is the final word on the webservice definition.

Axis

This version of the webservice uses Axis2 1.4.1 to generate the server-side classes that make up the Web service implementation from the WSDL file.Users are free to use any version of Axis or any other non-axis technique to create a client side implementation for integrating with the API from the supplied WSDL.

Versioning

The version of the protocol is explicitly bound to the WSDL as follows:

- As part of the WSDL name, for example, PlanIntegrationService1_0.wsdl.
- As part of the WSDL targetNamespace, for example xmlns:tns=”http://webservices.unica.com /plan/services/PlanIntegrationServices1.0?wsdl.”

WSDL

One WSDL file is provided with the IBM Unica Marketing Operations Integration Module: **PlanIntegrationServices1.0.wsdl**. The WSDL is delivered in the **integration/examples/soap/plan** directory. The example build script uses this file to generate the appropriate client-side stubs to connect to the webservice.

3 IBM Unica Marketing Operations procedures

IBM Unica Marketing Operations procedures

A *procedure* is a custom or standard Java class hosted by IBM Unica Marketing Operations that does some unit of work. Procedures provide a way for customers, partners and Unica Professional Services to extend Marketing Operations business logic in arbitrary ways.

Procedures follow a simple programming model, using a well-defined API to affect components managed by Marketing Operations. Procedures are "discovered" through a simple lookup mechanism and XML-based definition file. Marketing Operations executes the procedures according to needs of their "clients." For example, in response to an integration request (incoming) or a trigger firing (internal or outgoing).

Procedures run synchronously with respect to their client; results are made available directly to the client, and through a persisted auditing mechanism. The execution of a procedure could also cause other events and triggers to fire in Marketing Operations.

Procedures must be written in Java.

Assumptions

Note the following assumptions concerning procedures.

- The procedure implementation classes are packaged into a separate classes tree or jar file and made available to IBM Unica Marketing Operations through a URL path. The procedure execution manager uses an independent class loader to load these classes as needed. By default, Marketing Operations looks in the following directory:

```
<Marketing Operations_home>/devkits/integration/examples/classes
```

  To change this default, set the **integrationProcedureClasspathURL** parameter under **Settings > Configuration > Marketing Operations > umoConfiguration > integrationServices**.

- The procedure implementation class name follows the accepted Java naming conventions, to avoid package collisions with Unica and classes from other vendors. In particular, customers must not place procedures under the **com.unica** or **com.unicacorp** package tree.
- The procedure implementation is coded to the Java runtime version used by IBM Unica Marketing Operations on the application server (at least JRE 1.5.10).
- IBM Unica Marketing Operations provides some number of open source and third-party libraries; application servers also use different versions of these libraries. Generally, this list changes from release to release.

  To avoid possible compatibility problems, procedures should not use any open source, third-party or application server-specific libraries.

  However, if such packages are used by a procedure—or the secondary classes imported by the procedure—their use must agree exactly with the packages provided by Marketing Operations and/or the application server. Note that in this case, you may have to rework your procedure code, if a later version of Marketing Operations upgrades or abandons a library.

- The procedure implementation class is loaded by the class loading policy normally used by IBM Unica Marketing Operations (typically **parent-last**). The application server may provide development tools and options to reload classes that would apply to Marketing Operations procedures, but that is not required.
- The procedure must be thread-safe with respect to its own state; that is, its execute method cannot depend on internal state changes from call to call.
- A procedure cannot create threads on its own.

Design

The design should focus on producing a single unit of work that must be performed atomically. Ideally, the procedure performs some series of tasks that can be scheduled asynchronously to execute at some later time; this "fire and forget" integration model results in the least load on both systems.

The procedure implementation class uses the IBM Unica Marketing Operations API to read and update Marketing Operations components, invoke services, and so on. Other Java packages can be used to perform other tasks.

Only the documented classes and methods will be supported in future releases of Marketing Operations. All other classes and methods in Marketing Operations should be considered off-limits.

Once coded and compiled, the procedure implementation classes must be made available to Marketing Operations. The build scripts supplied with the Marketing Operations Integration Services place the compiled procedures in the default location. The final development step is to update the custom procedure plug-in definition file used by Marketing Operations to discover the custom procedures.

The procedure must implement the **com.unica.publicapi.plan.plugin.procedure.IProcedure** interface and have a parameter-less constructor (usual JavaBean model). Coding and compilation of each procedure is done in a Java IDE of the customer’s choice, such as Eclipse, Borland JBuilder, Idea, and so on. Sample code is provided with IBM Unica Marketing Operations as developer toolkits, in the following location.

```
<Marketing Operations-home>/devkits/integration/examples/src/procedure
```

Configuration

Use the parameters under **Settings > Configuration > Marketing Operations > umoConfiguration > integrationServices** to configure the Marketing Operations Integration Module.

For details, see the *Marketing Operations Installation Guide*.

Procedure lifecycle

The runtime lifecycle of a procedure is:

1. Discovery and initialization
2. Selection (optional)
3. Execution

4. Destruction

Discovery and initialization

IBM Unica Marketing Operations must be made aware of all standard and custom procedures available for a particular installation instance. This process is called discovery.

Standard procedures (procedures defined by the Marketing Operations engineering team) are known implicitly and so do not need any action to be discovered.

Custom procedures are defined in the procedure plug-in definition file. The Marketing Operations plug-in manager reads this file during initialization. For each procedure found, the plug-in manager does the following:

1. Instantiate the procedure; transition its state to INSTANTIATED.
2. Create a new procedure audit record.
3. If the procedure could be instantiated, its **initialize()** method is called with any initialization parameters found in its plug-in description file. If this method throws an exception, the status is logged and the procedure is abandoned. Otherwise, the procedure transitions to the INITIALIZED state. It is now ready to execute.
4. Create a new procedure audit record.
5. If the procedure could be initialized, its **getKey()** method is called to determine the key used by clients to reference the procedure. This key is associated with the instance and saved for later lookup.

Selection

From time to time, IBM Unica Marketing Operations may need to present a list of available procedures to users, for example to enable administrators to setup a trigger. This is done only after the procedure has been initialized. The procedure's **getDisplayName()** and **getDescription()** methods are used for this purpose.

Execution

At some point after the procedure has been initialized, IBM Unica Marketing Operations receives a request to execute the procedure. This may happen concurrently with other procedures (or the same procedure) executing on other threads.

At execution time, the procedure execution manager does the following.

1. Start a database transaction.
2. Set the procedure state to EXECUTING.
3. Create a new procedure audit record.

4. Call the procedure’s **execute()** method with an execution context and any execute parameters provided by the client. The method implementation uses the Marketing Operations API as needed, acquiring edit locks and passing along the execution context. If the execute method throws an exception, the execution manager marks the transaction for rollback.
5. Commit or rollback the transaction according to the execution results; set procedure state to EXECUTED.
6. Release any outstanding edit locks.
7. Create a new procedure audit record.

The **execute()** method should not alter the procedure instance data.

Destruction

When IBM Unica Marketing Operations shuts down, the procedure plug-in manager walks through all loaded procedures. For each procedure found, it does the following:

1. Calls the procedure's destroy() method to allow the procedure to clean up before the instance is destroyed.
2. Changes the state of the procedure to FINALIZED (it cannot be executed).
3. Creates a new procedure audit record.
4. Destroys the instance of the procedure.

Data locking

IBM Unica Marketing Operations uses a pessimistic edit locking scheme; that is, only one user is granted update access to component instance(s) at a time. For the GUI user, this locking is done at the visual tab level—in some cases this represents a subset of an instance (for example, a project summary tab), while in others it represents many instances (the workflow tab). Once a user has acquired a lock, all other users are restricted to read-only access to the related data.

Edit locks are not the same as database transactions.

In order to ensure that the changes made by a procedure to a component instance or group of instances are not inadvertently overwritten by another user, a procedure must acquire the appropriate lock(s) before updating component data. The execution context object passed to the procedure's **execute()** method is used to accomplish this.

Before the procedure updates any data, it must call the context's **acquireLock()** method for each lock it needs. For example, if a procedure is going to update a project and the associated workflow, the procedure needs to acquire locks for both.

If another user already has a lock, the **acquireLock()** method throws a **LockInUseException** immediately. In order to minimize collisions, the procedure should release the lock as soon as it updates the object.

The execution manager automatically releases any outstanding locks when the execute method returns. In any case, locks are only held for the life of the database transaction. That is, locks expire if the database-specific transaction timeout is exceeded.

Procedure transactions

The procedure execution manager automatically wraps execution of the procedure with a database transaction, committing or rolling it back as appropriate based on the outcome of the procedure execution. This guarantees that updates to the IBM Unica Marketing Operations database are not visible to other users until committed and that the updates are atomic.

The procedure writer still must acquire the necessary edit locks to ensure that other users cannot write changes to the database before the procedure execution completes.

Communicating results

The **execute()** method of a procedure returns an integer status code and zero or more messages which are logged and persisted to the IBM Unica Marketing Operations procedure audit table. The client may also communicate the status information in some other fashion.

Procedure logging

IBM Unica Marketing Operations has a separate log file for procedures.

```
<Marketing Operations-home>\logs\procedure.log
```

The procedure execution manager logs the lifecycle of each procedure and creates audit records.

- **logInfo()**: write an informational message to the procedure log
- **logWarning()**: write a warning message to the procedure log
- **logError()**: write an error message to the procedure log
- **logException()**: dump the stack trace for the exception to the procedure log

Key Java classes

The supplied integration devkit contains a set of Javadoc for the public IBM Unica Marketing Operations API and supporting classes. The most important are listed here.

- IProcedure (com.unica.publicapi.plan.plugin.procedure.IProcedure): interface that all procedures must implement. Procedures go through a well-defined lifecycle and access the Marketing Operations API to perform work.
- ITriggerProcedure (com.unica.publicapi.plan.plugin.procedure.ITriggerProcedure): interface that all trigger procedures must implement (marker interface).
- IExecutionContext (com.unica.publicapi.plan.plugin.procedure.IExecutionContext): interface of opaque context object handed to the procedure by the execution manager. This object has public methods for logging and edit lock management. The procedure also passes this object to all PlanAPI calls.
- IPlanAPI (com.unica.publicapi.plan.api.IPlanAPI): interface to the Marketing Operations API. The execution context provides a **getPlanAPI()** method to retrieve the proper implementation.

Procedure example

This is a standard procedure to change the state of a project from an integration webservice or a trigger.

Do not modify the sample procedures and their XML definitions, as the samples are over-written when you upgrade IBM Unica Marketing Operations and you will lose your changes. You should create and modify all custom procedures in a different directory.

```
// ProjectStateChangeProcedure
// (c) Copyright 2006 by Unica Corporation. All rights reserved.

package com.unica.uap.plugin.procedure.standard;

import java.util.Iterator;
import java.util.List;
import java.util.Locale;
import java.util.Map;

import com.unica.publicapi.plan.api.Handle;
import com.unica.publicapi.plan.api.IExecutionContext;
import com.unica.publicapi.plan.api.IPlanAPI;
import com.unica.publicapi.plan.api.LockInUseException;
import com.unica.publicapi.plan.api.ProjectHandle;
import com.unica.publicapi.plan.api.ProjectStateEnum;
import com.unica.publicapi.plan.plugin.PluginVersion;
import com.unica.publicapi.plan.plugin.procedure.IProcedure;
import
com.unica.publicapi.plan.plugin.procedure.ProcedureExecutionException;
import
com.unica.publicapi.plan.plugin.procedure.ProcedureInitializationException;
import com.unica.publicapi.plan.plugin.procedure.ProcedureMessage;
import com.unica.publicapi.plan.plugin.procedure.ProcedureMessageTypeEnum;
import com.unica.publicapi.plan.plugin.procedure.ProcedureResult;

/**
```

```
 * <b>ProjectStateChangeProcedure</b> is a standard Marketing Operations
procedure that attempts to
 * transition the state of a project.
 * <p>
 * Expects the following initialization parameters:
 * <ul>
 *   <li>debug: Boolean object, <tt>true</tt> or <tt>false</tt>, indicating

 *   if debug tracing is enabled or not</li>
 * </ul>
 *
 * <p>
 * Expects the following execute parameters:
 * <ul>
 *   <li>hProject: string array form of project handle, e.g.,
 *
"http://mymachine:7001/plan/affiniumplan.jsp?cat=projecttabs&projectid=12"</
li>
 *   <li>uapState: string array form of new project state, e.g.,
"COMPLETED". Note,
 *   case matters!</li>
 * </ul>
 *
 */
public final class ProjectStateChangeProcedure implements IProcedure {
// initialization parameters
private final static String DEBUG_INITPARAMETER_NAME = "debug";
    // execute parameters
    private final static String HPROJECT_PARAMETER_NAME = "hProject";
    private final static String STATE_PARAMETER_NAME
        = IPlanAPI.PROJECT_ATTRIBUTE_STATEENUM; // same as attribute name

    // our status codes
    private final static int STATUS_SUCCESS = 0;

    // debug property. set the procedure's "debug" init parameter to true
to enable debug trace
    private boolean _debug = false;
    private boolean isDebug() { return _debug; }

    // simple name is unqualified class name
    public String getName() {
        return "uapProjectStateChangeProcedure";
    }

    // display name is always key
    public String getDisplayName(Locale locale) {
        // only do EN for now
        return getName();
    }

    // description always in english
    public String getDescription(Locale locale) {
        // only do EN for now
        return "A procedure to transition the state of a project.";
    }

    // version we're coded to; must be 1.0.0 for now
    public PluginVersion getVersion() {
        return new PluginVersion(1,0,0);
    }

    // initialize instance from init parameters
    public void initialize(Map initParameters) throws
ProcedureInitializationException {
        // the only init parameter we have is: debug, Boolean
        if (initParameters.containsKey(DEBUG_INITPARAMETER_NAME)) {
            try {
                _debug =
((Boolean)initParameters.get(DEBUG_INITPARAMETER_NAME)).booleanValue();
            } catch (Exception exception) {
                throw new ProcedureInitializationException("Problem using "

                                                            +
DEBUG_INITPARAMETER_NAME
                                                            + " init
parameter: "
                                                            +
exception.getMessage());
            }
        }
    }

    // execute: expect hProject and state enum
    public ProcedureResult execute(IExecutionContext context, Map
parameters)
        throws ProcedureExecutionException {
        // get execute parameters: we expect two:
        // - hProject: string[] form of project handle
        // - uapState: string[] form of ProjectStateEnum
        ProjectHandle hProject = null;
        if (parameters.containsKey(HPROJECT_PARAMETER_NAME)) {
            try {
                hProject = (ProjectHandle)
Handle.makeHandle(((String[])parameters.get(HPROJECT_PARAMETER_NAME))[0]);
            } catch (Exception exception) {
                throw new ProcedureExecutionException("Problem using "
                                                        +
HPROJECT_PARAMETER_NAME
                                                        + " parameter: "
                                                        +
exception.getMessage());
            }
        } else throw new
ProcedureExecutionException(HPROJECT_PARAMETER_NAME
                                                        + " parameter must
be provided.");

        ProjectStateEnum stateEnum = null;
        if (parameters.containsKey(STATE_PARAMETER_NAME)) {
            try {
                stateEnum =

ProjectStateEnum.valueOf(((String[])parameters.get(STATE_PARAMETER_NAME))[0]
);
            } catch (Exception exception) {
                throw new ProcedureExecutionException("Problem using "
                                                        +
STATE_PARAMETER_NAME
                                                        + " parameter: "
                                                        +
exception.getMessage());
            }
        } else throw new ProcedureExecutionException(STATE_PARAMETER_NAME
                                                        + " parameter must
be provided.");

        int status = -1;
        ProcedureMessage[] messages = null;
        try {
             // try to acquire an edit lock for the project
            context.acquireLock(hProject,
IExecutionContext.LOCK_ALL_FIELDS);

            // use PlanAPIImpl to update state
            IPlanAPI planAPI = context.getPlanAPI();
            planAPI.updateAttribute(context, hProject,
STATE_PARAMETER_NAME,
                                    new ProjectStateEnum[]{stateEnum});

            // success
            status = STATUS_SUCCESS;

        } catch (Exception exception) {
            // write stack trace if debug
            if (isDebug()) {
                context.logError(getName(), exception);
            }
            throw new ProcedureExecutionException(exception);
        } finally {
            // release our lock
            try {
                context.releaseAllLocks();
            } catch (Exception exception) { /* ignored */ }
        }

        return new ProcedureResult(status, messages);
    }

    public void destroy( ){
        // we don't need to do anything
    }
}
```

Procedure plug-in definition file

This topic describes the procedure plug-in definition file. This file defines implementation class, metadata, and other information about the custom procedures to be hosted in IBM Unica Marketing Operations. By default, the procedure plug-in definition is assumed to be in the following path:

```
<Marketing Operations-
home>/devkits/integration/examples/src/procedures/procedure-plugins.xml
```

This file is an XML document that contains the information described below.

Procedures: a list of zero or more **Procedure** elements.

Procedure: an element that defines a procedure. Each procedure contains the following elements.

- **key** (optional): string defining the lookup key for the procedure. This key is must be unique with respect to all standard (IBM-supplied) and custom procedures hosted by a particular Marketing Operations instance. If not defined, defaults to the fully-qualified version of the **className** element. Names starting with the string "uap" are reserved for use by IBM Unica Marketing Operations.
- **className** (required): fully-qualified package name of the procedure class. This class must implement the IProcedure class (com.unica.public.plan.plugin.procedure.IProcedure).
- **initParameters** (optional): a list of zero or more initParameter elements.
    - **initParameter:** (optional): parameter to be passed to the procedure's initialize() method. This element includes the nested parameter name, type and value elements.
        - name: string defining the parameter name
        - type: optional class name of the Java wrapper class that defines the type of the parameter value. Must be one of the following types:
            - java.lang.String (the default)
            - java.lang.Integer
            - java.lang.Double
            - java.lang.Calendar
            - java.lang.Boolean
        - value: string form of the attribute value according to its type

4 Unica Marketing Operations API

About the IBM Unica Marketing Operations API

The IBM Unica Marketing Operations API is a façade that provides a client view of a running Marketing Operations instance. Only a subset of the Marketing Operations capabilities are exposed. The API is designed to be used concurrently by Marketing Operations web users, Marketing Operations Integration WebService SOAP requests and triggers. The API supports the following types of operations.

- Component creation and deletion
- Discovery (by component type, attribute value, and so on)
- Component inspection (via its attributes, specialized links, and so on)
- Component modification

Versioning and backwards-compatibility

Future versions of this API will be backwardly-compatible with all minor and maintenance releases that share the same major version number. However, IBM reserves the right to break backward-compatibility for dot zero (x.0) major releases if the business or technical case warrants doing so.

The major version number of this API will be incremented if any of the following changes are made.

- Data interpretation changed
- Business logic changed (for example service method functionality changed)
- Method parameters and/or return types changed

The minor version number of the API will be incremented if any of the following changes are made (note, that these changes are by definition backward-compatible).

- New method added
- New data type added and its usage restricted to a new method
- New element added to an enumerated type
- A new version of an interface is defined with a version suffix

User security

Authentication is assumed to be done by the procedure's execution manager and the authenticated user information bound to the execution context used by all APIs. The API does not expose the authenticated user, but will pass it on to IBM Unica Marketing Operations to use as needed.

The authenticated user, however, may not be authorized to perform all the operations exposed by the API; in this case, the API method will throw an**AuthorizationException**.

Locale

The only locale supported by this version is the locale currently configured for the IBM Unica Marketing Operations server instance. All locale-dependent data accessible via the API (messages, currency, etc.) are assumed to be in this system locale.

State management

This API is stateless, meaning that no per-client information is saved by the API across calls.

Note, however, that specific API calls may change the state of underlying component instances managed by IBM Unica Marketing Operations, and these state changes may be persisted to a database.

Database transactions

This API does not expose database transactions to the client, but will use such information if included in the execution context. If a transaction is started, then the effect of all API calls within a particular procedure will be atomic. Other users of IBM Unica Marketing Operations will not see the changes until the procedure successfully commits the transaction.

API calls that update the database must first acquire an edit lock to prevent other Marketing Operations users from modifying the underlying data during the course of the API call(s). Other users will not be able to update locked components until the API completes; likewise, another Marketing Operations user—or API client—may have acquired the lock on the desired data which will prevent the API call from completing.

Event processing

Operations performed on IBM Unica Marketing Operations components via this API generate the same events as if the operation were performed by a Marketing Operations Web user. In particular, triggers waiting for certain events eventually will fire in both cases. Users that subscribed to certain notifications (for example, when a project state changes) will be notified of state changes that result from API calls as well as Web user actions.

About the API data types

The IBM Unica Marketing Operations API contains the following public data types.

- **ApprovalMethodEnum**: Enumerated type for approval methods.
- **ApprovalStateEnum**: Enumerated type for approval states.
- **AssetLibraryStateEnum**: Enumerated type for asset library states.
- **AssetStateEnum**: Enumerated type for assert states.
- **AttachmentTypeEnum**: Enumerated type for attachment.
- **AttributeMap**: a java Map containing attributes and their values.
- **BudgetPeriodEnum**: Enumerated type for budget period types.
- **BudgetTypeEnum**: Enumerated type for budget types
- **Handle**: identifies a component instance in a particular Marketing Operations instance.
- **InvoiceStateEnum**: Enumerated type for invoice states.
- **MonthEnum**: Enumerated type for months in a calendar year
- **ProjectCopyTypeEnum**: Enumerated type for copy project options.
- **ProjectStateEnum**: Enumerated type for project states.
- **TaskStateEnum**: Enumerated type for task states.
- **QuarterEnum**: Enumerated type for quarters in a calendar year
- **WeekEnum**: Enumerated type for weeks in a calendar year

Possible values for each data type are mentioned below.

Enumerated types

ApprovalMethodEnum

ApprovalMethodEnum is an enumerated type that defines all possible methods of a approval. Possible values are:

- SIMULTANEOUS

- SEQUENTIAL

ApprovalStateEnum

ApprovalStateEnum is an enumerated type that defines all possible states of a approval. Possible values are:

- NOT_STATED
- ON_HOLD
- IN_PROGRESS
- COMPLETED
- CANCELLED

AssetLibraryStateEnum

AssetLibraryStateEnum is an enumerated type that defines all possible states of a asset library. Possible values are:

- ENABLED
- DISABLED

AssetStateEnum

AssetStateEnum is an enumerated type that defines all possible states of a asset. Possible values are:

- DRAFT
- LOCK
- FINALIZE
- ARCHIVE

AttachmentTypeEnum

AttachmentTypeEnum is an enumerated type that defines all possible types of attachment. Possible values are:

- URL
- FILE
- ASSET

BudgetPeriodEnum

BudgetPeriodEnum is an enumerated type that defines all possible budget periods. Possible values are:

- QUARTERLY
- MONTHLY
- WEEKLY
- YEARLY

- ALL

BudgetTypeEnum

BudgetTypeEnum is an enumerated type that defines all possible budget types. Possible values are:

- TOTAL
- FORECAST
- COMMITTED
- ACTUAL
- ALLOCATED

InvoiceStateEnum

InvoiceStateEnum is an enumerated type that defines all possible states of a invoice. Possible values are:

- DRAFT
- PAYABLE
- PAID
- CANCELLED

MonthEnum

MonthEnum is an enumerated type that defines all possible values for month.

ProjectCopyTypeEnum

ProjectCopyTypeEnum is an enumerated type that defines all possible types of copy project. Possible values are:

- COPY_USING_PROJECT_METRICS
- COPY_USING_TEMMPLATE_METRICS

For details on these project and task states, see the *Marketing Operations User Guide*.

ProjectStateEnum

ProjectStateEnum is an enumerated type that defines all possible states of a project or request. Possible values are:

- NOT_STARTED
- IN_PROGRESS
- ON_HOLD
- IN_RECONCILIATION
- LATE: indicates the project did not start by its scheduled begin date.
- OVERDUE: indicates the project was not completed before its scheduled end date.
- DRAFT

- SUBMITTED
- RETURNED
- ACCEPTED
- COMPLETED
- CANCELLED
- DELETED

TaskStateEnum

TaskStateEnum is an enumerated type that defines all possible states of a workflow task. Possible values are:

- PENDING
- ACTIVE
- FINISHED
- SKIPPED
- DISABLED

QuarterEnum

QuarterEnum is an enumerated type that defines all possible values for quarter.

WeekEnum

WeekEnum is an enumerated type that defines all possible values for week.

Handle

A Handle is special kind of URL object that refers to a component instance in a particular Marketing Operations instance. Handles include the component type, internal data identifier, an instance base URL, and so on. Handles used or generated by the API may be externalized to a full URL that can be used to navigate to a view of the component in the Marketing Operations GUI, be cut-and-pasted, sent in emails, used in another procedure as a parameter, and so on.

Handles are valid only for a particular Marketing Operations service instance—or clustered instance—but are guaranteed valid for the life time of the deployed service. This means that handles can be saved in a file for later reference, but they cannot be used to access components on another Marketing Operations instance (even if on the same machine). Marketing Operations does provide, however, a mechanism for mapping different base URLs to the current instance to accommodate relocating an instance to another machine (for example, in case the original machine malfunctions).

Handles are client-independent. For example, a trigger may pass a handle to a procedure, which uses it as a parameter to a SOAP call to a 3rd-party system, which turns around and issues a SOAP request back to Marketing Operations to invoke a procedure that updates an attribute.

The Handle class has factory methods for creating handles from various types of URLs.

Approval Object:

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=approvaldetail&approvalid=10
1
```

AssetLibrary Object:

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

Asset Folder Object:

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

Asset Object:

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

Attachment Object:

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

Financial Account Object:

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

Invoice line item object:

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

Invoice Object:

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

Marketing object:

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

Marketing object grid:

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

Marketing object grid row:

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

Project:

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

Project grid:

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

Project grid row:

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

Project line item object:

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

Team Object:

```

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

User Object:

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

Workflow task:

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

Attribute Map

An AttributeMap is a Map that contains only attributes. The attribute *name* is the map entry key, and the attribute *values* array (note plural) is the map entry value.

AttributeMap includes the following fields:

- *Name*: the programmatic name of the attribute. This name serves as a unique key for accessing the attribute within the component instance in which it occurs.

  Name is not necessarily the display name presented to a user in the GUI. For components created with templates (such as projects or workflow tasks), the attribute name is specified by the template element definition and must be unique. For other components, the attribute name is usually derived programmatically from the server-side component instance (for example through Java introspection)

  By convention, custom attributes include the name of the form in which the editable version is defined: <form_name>.<attribute_name>.

- *values*: a Java object array, containing zero or more attribute values. The type of each value must be the same and agree with the type of the attribute as it is defined in Marketing Operations. Only the following Java wrapper and Marketing Operations types are supported:
  - AssetLibraryStateEnum: a AssetLibraryStateEnum enumerated type value.
  - AssetStateEnum: a AssetStateEnum enumerated type value.
  - AttachmentTypeEnum: a AttachmentTypeEnum enumerated type value.
  - AttributeMap: a map which hold attributes.
  - BudgetPeriodEnum: a BudgetPeriodEnum enumerated type value.
  - BudgetTypeEnum: a BudgetTypeEnum enumerated type value.
  - Handle: a reference to a component instance, grid row, attribute, and so on.
  - InvoiceStateEnum: an InvoiceStateEnum enumerated type value.
  - java.io.File: representation of a file.
  - java.lang.Boolean: a boolean value, either True or False
  - java.lang.Double: a double-precision decimal number value
  - java.lang.Float: a single-precision decimal number value
  - java.lang.Integer: a 32-bit integer value
  - java.lang.Long: a 64-bit integer value
  - java.lang.Object: Generic java object
  - java.lang.String: a string of zero or more Unicode characters

- java.math.BigDecimal: arbitrary-precision signed decimal number value. Suitable for currency; the interpretation of the value depends on the client’s currency locale.
- java.math.BigInteger: arbitrary-precision integer value
- java.net.URL: a Universal Resource Locator (URL) object.
- java.util.ArrayList: List of objects.
- java.util.Calendar: a date-time value for a particular locale.
- java.util.Date: a date-time value. This type is deprecated. Instead of this use java.util.Calendar or java.util.GregorialCalendar.
- java.util.GregorianCalendar: GregorianCalendar is a concrete subclass of java.util.Calendar and provides the standard calendar system used by most of the world.
- MonthEnum: a MonthEnum enumerated type value.
- ProjectStateEnum: a ProjectStateEnum enumerated type value.
- QuarterEnum: a QuarterEnum enumerated type value.
- TaskStateEnum: a TaskStateEnum enumerated type value.
- WeekEnum: a WeekEnum enumerated type value.

An attribute's meta data (such as localized display name, description, and so on) is defined by the template associated with the attribute and its parent object instance. Attributes provide a simple yet extensible mechanism for exposing both required and optional object instance attributes, for example a project's name, code, start date and so on.

As per the implementation of date, user can select either java.util.Calendar or java.util.GregorianCalendar.

Common exceptions

This topic describes some common exceptions thrown by the API.

- AuthorizationException: The user associated with the client’s execution context is not authorized to perform the requested operation. This can be thrown by any API method, so is undeclared.
- DataException: An exception occurred in the underlying database layer in IBM Unica Marketing Operations. Check the SQL log for details.
- InvalidExecutionContextException: There is a problem with an execution context passed to an API method (for example, the method was not initialized correctly). This can be thrown by any API, so is undeclared.
- NotLockedException: Attempt to update component data without first acquiring the required lock. See the acquireLock() method of IExecutionContext.

API methods

See the iPlanAPI class in the javadoc, posted on IBM Unica Customer Central (http://customers.unica.com/), for specific information about the public API methods.