FlowWright Developer's Guide

April 2nd, 2018

Table of Contents

Introduction

Welcome to the FlowWright developer's guide, this guide will help you navigate through the FlowWright API using code and examples.

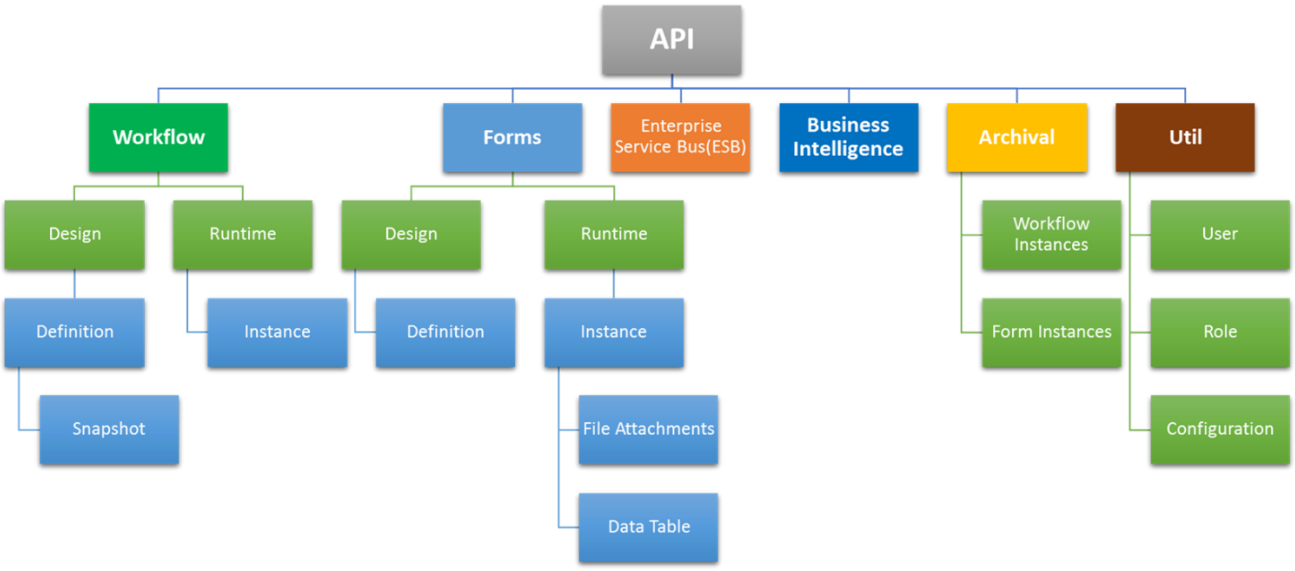
cDevWorkfow Concepts

- A Workflow Definition (WD) is made up of steps, connections and comments
- A Workflow Instance (WI) is instantiated from a Workflow Definition
- A Form Definition (FD) is made up of fields, text labels and controls
- A Form Instance (FI) is instantiated from a Form Definition

FlowWright API Components

API consists of many components, and they are:

- deDesign – supports all design time operations, such as create/getting/managing workflow definitions
- deDefinition – manages a single workflow definition object, provides access to its properties and methods
- deSnapshot – manages all operations for a given snapshot
- deRuntime – manages all runtime operations, such as managing workflow instaces
- deInstace – manages a single workflow instance object, provides access to its properties and methods
- deStepExecutionInfo – contains detail information for step execution data
- deBusinessIntelligence – manages all business intelligence data operations
- deExpression – manages all expression evaluation operations
- deUser – manages all operations for the virtual workflow users
- deEventServiceBus – manages all event sub bus operations, such as managing events



Using the deDesign API

Creating the deDesign object

```
deDesign oDesign = new deDesign("connection string to the FlowWright
database", "exernal user name");
```

Creating a new Workflow Definition

```
deDefinition oDef = oDesign.createDefinition("name of the definition");
```

createDefinition method creates a deDefinition object using the provided name of the definition.

Remove a Workflow Definition

```
bool bFlag = oDesign.removeDefinition("id of the definition");
```

removeDefinition method removes a given definition using the definitons id.

Get a Workflow Definition

```
deDefinition oDef = oDesign.getDefinition(id of the definition);
```

Gets the deDefinition object using the id of the definition.

```
deDefintion oDef = oDesign.getDefUsingName(name of the definition);
```

Gets the deDefinition object using the name of the definition.

Get Workflow Definitions

```
Dictionary<string, deDefinitions> oDefs = oDesign.getDefinitions();
```

Gets all the definitions within the application.

```
Dictionary<string, deDefinitions> oDefs = oDesign.getDefinitions("created by
user id");
```

Gets all definitions created by a given user.

```
Dictionary<string, deDefinitions> oDefs = oDesign.getDefinitions("", "tenant
ID");
```

Gets all definitions for a given tenant.

Create a Workflow User

```
bool bFlag = oDesign.createUser("external user identification", "external
user full name", "external user email", "admin user true or false", "ref
userID");
```

creates an User object given the user information.

Get a Workflow User

```
deUser oUser = oDesign.getUserUsingExternalUserName("external user full
name");
```

Get the deUser object using the user's external user name

Get a list of Workflow Users

```
Dictionary<string, deUser> oUsers = oDesign.getUsers();
```

Gets a list of Workflow users into a dictionary object.

Get the current user's Workflow user id

```
String sUserID = oDesign.getWorkflowUserID();
```

Gets the current user's workflow user ID, this user id is maintained internally to identify the user.

Using the deDefinition API

deDefiniton API lets you manage all operations on a Workflow definition. A Workflow Definition object is retrieved using the deDesign object.

Get Workflow Definition properties

```
Hashtable oProps = oDef.getProperties();
```

Returns all properties, all fields of the definition object.

```
Hashtable oProps = oDef.getProperties("defID", "defName");
```

Returns only the specified properties for the definition object.

Copy a Workflow Definition

```
deDefinition oDef2 = oDef.copyDefinition("name of the new definition");
```

```
deDefinition oDef2 = oDef.copyDefinition("name of the new definition", "id of the new definition");
```

Create a copy of the selected definition.

Create a new Workflow Instance from a Workflow Definition

```
deInstance oInst = oDef.createInstance("instance name");
```

```
deInstance oInst = oDef.createInstance("instance name", "instance ID");
```

Create a new Workflow Instance object based on a selected Workflow Definition.

Generate a graphical representation of the definition

```
bool bFlag = oDef.generateModelGraphic("image output file path", "images file path");
```

Generates a PNG file of the Workflow Definition.

Create Snapshot

```
deSnapShot oSnapShot = oDef.createSnapshot("snapshot name");
```

Creates a snapshot object from the selected Workflow Definition

Get a Snapshot

```
deSnapShot oSnapShot = oDef.getSnapShot("snapshot id");
```

Gets a specific snapshot using the snapshot ID

Get a list of snapshots for the definition

```
Dictionary<string, deSnapshot> oList = oDef.getSnapshots();
```

Gets a list of snapshots for the current definition using a Dictionary object.

Get a list of start variables for the definition

```
Hashtable oVars = oDef.getStartVariables();
```

Retuns a hastable of all variables for the Workflow Definition.

Get the UI model object for the Workflow Definition

```
clsUIModel oModel = oDef.getUIModel();
```

Returns the UI model object for navigating the design of the Workflow Definition.

Remove all instances for a given Workflow Definition

```
Bool bFlag = oDef.removeAllInstances();
```

Removes all instances for the selected definition

Remove all snapshots for the given Workflow Definition

```
Bool bFlag = oDef.removeAllSnapshots();
```

Removes all snapshots for the selected definition

Update the XML for the Workflow Definition

```
Bool bFlag = oDef.updateDefinition("xml definition");
```

Updates the Definition object using the given xml string

Update definition properties

```
Hashtable oInParms = new Hashtable();

oInParms[“defName”] = “new definition name”;

bool bFlag = oDef.updateProperties(oInParms);
```

Updates the properties of the Definition object using the hashtable of fields and values

Using the deSnapshot API

Get a Snapshot

```
deSnapShot oSnapShot = oDef.getSnapShot("snapshot id");
```

Gets a specific snapshot using the snapshot ID

Get a list of snapshots for the definition

```
Dictionary<string, deSnapshot> oList = oDef.getSnapshots();
```

Gets a list of snapshots for the current definition using a Dictionary object.

Generate a graphical model

```
Bool bFlag
= oSnapshot.generateModelGraphic(ref Bitmap oImageBitmap, string imageFilesPa
th, ref Point oDeltaPoint)
```

Generates a graphical model for the snapshot as an bitmap image.

Get snapshot properties

```
Hashtable oProps = oSnapshot.getProperties(params string[] propertyNames)
```

Gets a list of properties for the snapshot, if property names are provided, then only those properties will be retrieved.

Using the deRuntime API

The deRuntime API manages all runtime operations such as managing instances.

Creating the deRuntime Object

```
deRuntime oRuntime = new deRuntime("connection string", "external user name");
```

Creates the deRuntime object

Get a Workflow Instance

```
deInstance oInst = oRuntime.getInstance("id of the instance");

deInstance oInst = oRuntime.getInstanceUsingName("name of the instance");
```

Gets the Workflow Instance object using the id or the name

Abort a Workflow Instance

```
Bool bFlag = oRuntime.abortInstance("id of the instance");
```

Aborts the selected Workflow Instance

Complete a selected task

```
Bool bFlag = oRuntime.completeTask("id of the task", "task return value", "task comment");
```

Completes a given task programmatically

Extend a selected task

```
Bool bFlag = oRuntime.extendTask("id of the task", "number of days to extend by");
```

Extends the expiration of a task by a given number of days

Get a list of Workflow Instances

```
Dictionary<string, deInstances> oList = oRuntime.getInstances("created by user id")
```

Gets all Workflow Instances created by a certain user

```
Dictionary<string, deInstances> oList = oRuntime.getInstances("", "tenant ID")
```

```
Gets all Workflow Instances for a given tenant

Hashtable oList = oRuntime.getInstances("execution status");

Gets all instances based on a selected execution status
```

Get the status of the license file

```
licenseStatus oStatus = oRuntime.getLicenseStatus(ref "error message string",
ref "license properties");

Gets the status of the license and the licensing information
```

Get all open tasks for Users

```
Datatable oTasks = oRuntime.getOpenTasks();

Gets all open tasks for the current user

Datatable oTasks = oRuntime.getOpenTasks("string user id");

Gets all open tasks for a selected user
```

Get a selected task

```
Datatable oTask = oRuntime.getTask("task id");

Gets a selected task using a task ID
```

Get task choices

```
List<string> oList = oRuntime.getTaskChoices("task ID");

Gets a list of task choices for a selected task
```

Reassign all tasks

```
Bool bFlag = oRuntime.reAssignAllTasks("old user id", "new user id");

Reassigns all tasks from one user to another
```

Reassign tasks

```
Bool bFlag = oRuntime.reAssignTask("id of the task", "new user ID to
reassign");
```

Reassigns a selected task to another user

Remove a Workflow Instance

```
Bool bFlag = oRuntime.removeInstance("id of the instance");
```

Removes a selected Workflow Instance

Run the engine manually

```
oRuntime.runEngine();
```

Tells the engine to process the next item on the workflow queue

Using the deInstance API

Get a specific Workflow Instance

```
deInstance oInst = oRuntime.getInstance("id of the instance");
```

Gets the specific Workflow instance object

Execute a Workflow Instance

```
Bool bFlag = oInst.execute();
```

Executes the Workflow instance

```
Hashtable oInParms = new Hashtable();
oInParms["varA"] = "some value";

bFlag = oInst.execute(oInParms);
```

Sets the default values of the Workflow instance variables using the InParms and executes the Workflow instance

Get the definition for the Workflow instance

```
deDefinition oDef = oInst.getDefinition();
```

Returns the Workflow definition object for the instance

Get the parent instance

```
deInstance oParentInst = oInst.getParentInstance();
```

Returns the parent instance for the selected instance

Get Sub-Workflow instances

```
List<deInstance> oList = oInst.getSubWokflowInstances();
```

Returns a list of sub-workflow instances for the current instance

Check if the Workflow instance has sub-workflow

```
Bool bFlag = oInst.instanceHasSubWorklfows();
```

Returns True/False indicating if the Workflow instance has child workflows

Get a list of tasks for the Workflow instance

```
DataTable oTaskListData = oInst.getTasks();
```

Returns a data list table of tasks for the current instance

Close all open tasks for the Workflow instance

```
Bool bFlag = oInst.closeAllOpenTasks();
```

Closes all open tasks for the current instance

Get properties for the Workflow instance

```
Hashtable oProps = oInst.getProperties();
```

Gets all properties for the Workflow instance

```
Hashtable oProps = oInst.getProperties("instanceName", "defID");
```

Returns property information for only selected properties

Update properties of the Workflow instance

```
Hashtable oProps = new Hashtable();

oProps["instanceName"] = "FMLA Leave 0045";
bool bFlag = oInst.updateProperties(oProps);
```

Updates the provided properties for the Workflow instance

Update the Workflow instance XML

```
Bool bFlag = oInst.updateInstanceXML("xml string");
```

Updates the xml for the Workflow instance

Get all start variables for the Workflow instance

```
Hashtable oVars = oInst.getStartupVariables();
```

Gets a list of startup variables for the Workflow instance

Get a list of required variables for the Workflow instance

```
List<string> oList = oInst.getRequiredVariables();
```

Gets a list of all required variables

Get UI model for the Workflow instance

```
clsModel oModel = oInst.getUIModel();
```

Returns the UI modeling object for the current instance

Add message to execution iteration

```
bool bFlag= oInst.addMessageToExecutionIteration("id of the step", "execution
iteration #", "message key", "message");
```

Adds messages to the execution iteration information

Generate graphical view of the Workflow instance

```
bool bFlag = oInst.generateModelGraphic("show execution path", "image output
file path", "image file path");
```

Generates a graphical view of the instance using the PNG file format

Generate execution view of the Workflow instance

```
bool bFlag=oInst.generateModelExecutionViewGraphic("image output file path",
"image file path", "string to hold the image map");
```

Generates an execution view of the instance using the PNG file format

Get execution log messages

```
DataTable oDT = oInst.getLogMessages();
```

Gets a list of all execution log messages for the instance

Gets a specific step from the Workflow instance

```
clsStep oStep = oInst.getStep("id of the step");
```

Returns a clsStep object for a selected step

Get a collection of all steps for the Workflow instance

```
SortedList<string, clsStep> oList = oInst.getSteps();
```

```
Get a list of all steps, collection has step id and clsStep as objects
```

Reset the Workflow instance

```
Bool bFlag = oInst.reset();

Resets the Workflow instance to the start, also updates the instance using
the latest definition
```

Reset the Workflow instance back to a figure

```
Bool bFlag = oInst.resetBackToFigure("id of the execution iteration");

Resets the Workflow instance back to selected figure

Bool bFlag = oInst.resetBackToFigure("id of the figure", "iteration #");

Resets the Workflow instance back to the selected figure and iteration
```

Using the deBusinessIntelligence API

Creating the deBusinessIntelligence object

```
deBusinessIntelligence oBI = new deBusinessIntelligence("connection string to
the FlowWright database", "exernal user name");
```

Get # of Workflow Definitions

```
long lCount = oBI.getNumberOfDefinitions();
Gets the # of Workflow definitions
```

Get # of Workflow Instances

```
long lCount = oBI.getNumberOfInstances();
Gets the # of Workflow instances
```

Get # of Form Definitions

```
long lCount = oBI.getNumberOfFormDefinitions();

Gets the # of Form definitions
```

Get # of Form Instances

```
long lCount = oBI.getNumberOfFormInstances();

Gets the # of Form instances
```

Get # of Users

```
long lCount = oBI.getNumberOfUsers ();
```

Gets the # of users

Get # of Steps configured

```
long lCount = oBI.getStepCount();
```

Gets the # of steps configured

Using the deExpression class

The deExpression class provides methods for replacing expression values with variable values. The expression class also lets you evaluate expressions using the expression evaluation engine.

Replace an expression with Variable values

```
String sExpression = deExpression.replaceVariables("enter your
expression", "list of variable collection", ref string sError);
```

All variables used within the expression will be replaced with the variables values and the expression returned back with the replacements.

Evaluation and expression

```
Object sVal = deExpression.eval("enter your
expression", clsVariableCollection oVars, clsBusinessObjectCollection oBusine
ssObjects, ref string sError);
```

Evaluates the expression using the variables and business objects and returns the result.

Using the deEventServiceBus class

The deEventServiceBus class provides all API level methods required for managing functionality within the Event Service Bus (ESB)

Creating the deEventServiceBus object

```
deEventServiceBus  oESB =
deEventServiceBus(string databaseConnectString, string externalUserID, IError
Provider oErrProvider = null);
```

Creates an instance of the deEventServiceBus API object

Create an Event Definition

```
Bool bFlag = oESB.createEventDefinition("event name", "event category");
```

Create an event definition

Publish an event to the ESB

```
Bool bFlag = oESB.
publishEventUsingName(string eventDefName, string eventSource, Hashtable oEve
ntParms, eventPriority iEventPriority);
```

Publishes the event to the ESB and the engine will process the event based on the handlers.

Run the event engine manually

```
runEventEngine()
```

Execute the event engine manually for processing published events