1.5 Reach Engine Workflow Guide
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>4</td>
</tr>
<tr>
<td>Anatomy of a Workflow</td>
<td>4</td>
</tr>
<tr>
<td>Workflows Concepts</td>
<td>5</td>
</tr>
<tr>
<td>Best Practices</td>
<td>6</td>
</tr>
<tr>
<td>The Workflow Element</td>
<td>6</td>
</tr>
<tr>
<td>Practice 1: Create a Workflow</td>
<td>9</td>
</tr>
<tr>
<td>Steps</td>
<td>10</td>
</tr>
<tr>
<td>Initial Step Name</td>
<td>10</td>
</tr>
<tr>
<td>Step Elements</td>
<td>10</td>
</tr>
<tr>
<td>Transitions</td>
<td>13</td>
</tr>
<tr>
<td>A simple Workflow Step with a transition</td>
<td>13</td>
</tr>
<tr>
<td>Practice 3: Add Transitions</td>
<td>13</td>
</tr>
<tr>
<td>Branching Workflows with Multiple Transitions</td>
<td>15</td>
</tr>
<tr>
<td>Context Data Defs</td>
<td>16</td>
</tr>
<tr>
<td>A simple contextDataDef</td>
<td>17</td>
</tr>
<tr>
<td>A contextDataDef with more attributes defined</td>
<td>17</td>
</tr>
<tr>
<td>A more complex contextDataDef</td>
<td>18</td>
</tr>
<tr>
<td>Practice 4: Branching and Context Data Defs</td>
<td>19</td>
</tr>
<tr>
<td>Importing Workflows</td>
<td>21</td>
</tr>
<tr>
<td>Executing Workflows</td>
<td>23</td>
</tr>
<tr>
<td>Watchfolder Executions</td>
<td>23</td>
</tr>
<tr>
<td>Scheduled Executions</td>
<td>24</td>
</tr>
<tr>
<td>User-Initiated Executions</td>
<td>25</td>
</tr>
<tr>
<td>Workflow Configuration</td>
<td>25</td>
</tr>
<tr>
<td>Workflow Subjects</td>
<td>25</td>
</tr>
<tr>
<td>Controlling the Execution Method</td>
<td>26</td>
</tr>
<tr>
<td>Workflow Step Queues</td>
<td>30</td>
</tr>
<tr>
<td>Configuration</td>
<td>30</td>
</tr>
<tr>
<td>Expressions</td>
<td>31</td>
</tr>
<tr>
<td>Expression Delimiters</td>
<td>31</td>
</tr>
<tr>
<td>Expression Roots</td>
<td>32</td>
</tr>
<tr>
<td>Expression Paths</td>
<td>34</td>
</tr>
<tr>
<td>Accounting for Null Values</td>
<td>34</td>
</tr>
<tr>
<td>Functions</td>
<td>35</td>
</tr>
<tr>
<td>Advanced Expressions</td>
<td>35</td>
</tr>
<tr>
<td>Practice 6: Determine what these expressions do</td>
<td>35</td>
</tr>
<tr>
<td>Data Objects</td>
<td>35</td>
</tr>
<tr>
<td>Assets</td>
<td>36</td>
</tr>
<tr>
<td>Properties</td>
<td>36</td>
</tr>
<tr>
<td>Associations</td>
<td>36</td>
</tr>
<tr>
<td>Practice 7: Use a “Clip” as the subjectDOClassName</td>
<td>36</td>
</tr>
</tbody>
</table>
Introduction

When developing large, data-driven, and file-based applications, the need to perform certain tasks repeatedly and generically usually leads to the creation of thousands of lines of code to execute these “jobs” or “tasks”. This “job-code”, built to perform actions such as converting a video from one format to another, updating a database, or delivering a file over FTP, typically grows bloated and brittle as the video is ported from project to project.

The Reach Engine Workflow plugin solves this problem by abstracting basic tasks into individual modules that can be linked together with XML. Because the Workflow plugin knows nothing domain- or project-specific, the same code can be used to import a video into a Timeline in one project and into a Package in another, requiring only a few lines of project-specific XML to be written in each case, and creating individual modules with a high-level of reuse.

Even more importantly, existing workflows can be modified to add functionality to an existing process without deploying new code. For example, after you run the Reach Engine Ingest Asset workflow, you could run a distribution workflow.

PREREQUISITES

This document assumes that you have access to a Web Server running Reach Engine Studio User Interface (UI), its local properties file, its logs, and the Workflow Import folder.

See the following tutorials for more information about Reach Engine Studio UI and the Workflow Import folder.

ANATOMY OF A WORKFLOW

A Workflow element provides general information about the workflow, including the workflow name, description, subject, and result information. Each workflow element must contain the following information:

- XSD and namespace definitions
- Workflow id

Steps define the actions the workflow should take. Each workflow must have an <initialStepName> element as the first child element in the workflow. The initialStepName must refer to a valid step in the workflow. Each step must contain:

- A step type that is referenced via the xml namespace/schemaLocation in the workflow element.
- A name to identify the step within the workflow.
Transitions are within steps and define which step should be run next. Transitions must contain:

- A condition: An expression that is evaluated when a step completes. If true, the workflow goes to the step defined in targetStepName.
  
  **Note:** If a workflow step fails to complete, none of its transitions are evaluated, and workflow execution stops.

- A targetStepName: Defines the workflow step to perform next when a condition is true.

Context Data Defs (contextDataDef) are like variables for workflows, and are used to accept external input to your workflow, maintain and communicate data between steps and sub-flows, and store the result of a workflow. Context Data Def values are determined at runtime and must contain:

- A datatype that defines the type of data.
- A name that is used to reference the contextDataDef.

Workflow Concepts

Within workflows, the following concepts are important:

- **Branching** allows workflows to execute in a non-linear order. Transition conditions are evaluated sequentially in the order listed in the workflow. After a transition condition evaluates to true, the workflow moves to the target step and does not continue evaluating the remaining transitions within that step.

- **Workflow expressions** let you customize the generic Reach Engine workflow steps to fit your needs. The Reach Engine Expression syntax is built off Spring Expressions (SpEL).

- **Workflow step queues** allow you to limit the number of simultaneous executions of a step.

- **Importing a workflow** allows the workflow to be used by Reach Engine. Workflows are imported into Reach Engine Studio UI by placing the XML file into a configured watchfolder.

- **Executing a workflow** allows the workflow to run. Workflows are executed in three ways:
  - Watchfolder executions occur when a file is dropped into a watchfolder associated with a specific workflow.
  - Scheduled executions occur at a specific time.
  - User-initiated executions occur through user interaction and are available through the UI.
BEST PRACTICES

- When you use an XML editor to create your workflow files, the XML editor can use the Workflow XSD to validate your files and assist with code completion. Check out Oxygen, Eclipse, XMLSpy, or Komodo.

- It is fine to use the =expression syntax for simple things like =true; however, the best practice is to use the ${} syntax at all times (e.g., ${true})

- If you have a lot of noOpSteps that are simply evaluating conditions, you can hide them in the UI to reduce clutter by setting the step’s devStep attribute to true.

- Within workflows, always have one last transition that uses the condition ${true}. This ensures the workflow knows how to behave even if every other statement is false, similar to the last else block in an if, else, if else control structure in a programming language. You may not want your workflow to ever be in this state, but this practice helps your workflow “fail gracefully” and send an email or print information to the logs about why it is unable to proceed.

THE WORKFLOW ELEMENT

The workflow element declares general information about the workflow itself, including a unique key, name, descriptive information, subject information, and workflow result information. For example:

```xml
<workflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://levelsbeyond.com/schema/workflow"
  xmlns:production="http://levelsbeyond.com/schema/workflow/production"
  xsi:schemaLocation="
  http://levelsbeyond.com/schema/workflow
  http://www.levelsbeyond.com/schema/workflow-2.0.xsd
  http://levelsbeyond.com/schema/workflow/production
  http://www.levelsbeyond.com/schema/production/nimbus-projects-workflow-1.2.xsd
  
  id="_multiXsdExample"
  ...
  >
  <production:importFinalCutProject name="import project"
    projectExpression="${project}"
    projectFileExpression="${fcpProjectFile}"
    projectXmlExpression="${fcpProjectXML}"
    ingestExternalMediaExpression="${ingestExternalMedia}"
Figure 1: A Basic Workflow Element

In the basic workflow element:

1. Workflow XML is based on XML Schema Documents (XSDs) provided by Levels Beyond. XSDs provide a validation structure for your XML. If you have a good XML editor, the editor can read the supplied XSDs and provide instant validation and auto-completion.
   a. `xmlns`: Attributes that provide an alias for a particular XML namespace. Namespaces are URLs that have an underlying XSD. In our example, two namespaces are referenced: xsi, which is a W3C standard namespace, and empty, or the default namespace, which points at the default Levels Beyond workflow namespace.
   b. `xsi:schemaLocation`: Attribute that tells the document which XSD fulfills the supplied namespace. XSD locations are declared in pairs. First, by location (i.e., `http://levels beyond.com/schema/workflow`). Then by the full location and name of the XSD itself (e.g., `http://www.levels beyond.com/schema/workflow-2.0.xsd`). In the example, the default namespace is mapped to the default Reach Engine Workflow XSD, v2.0.

   **Note:** The workflow XSDs are updated periodically as new versions of Reach Engine Workflow are released. When a new version is available, version numbers can be updated, or the new version can be added in addition to the schemas already defined.

After you set up the header attributes, you can add Reach Engine Workflow elements to your Workflow XML document. In the Step Elements section, you learn how to add additional XSDs to your Workflow and reference them.

The following attributes are used to identify your workflow throughout different parts of Reach Engine.

2. `id`: A required attribute that is a unique identifier for this workflow within your Reach Engine Studio UI installation. When you import a Workflow XML, Reach Engine Studio UI looks at the `id` attribute value and does one of two things: If a workflow already exists in the system with the same ID, it creates a new version of that workflow with your XML. Otherwise, it creates a new workflow in the system. Using IDs allows you to modify workflows over time to correct a problem or introduce new functionality.

   **Tip**
When you tell Reach Engine Studio UI to run a workflow with a given ID, it always uses the latest version for the workflow execution.

Currently, you cannot go back in the version history.

All of the built-in Reach Engine Workflows have IDs that contain an underscore. This underscore is a special designation for system default workflows

3. **name**: The name of the workflow as it will appear throughout the Reach Engine Studio User Interface (UI).

4. **description**: A more verbose description of what the workflow does. This information is used primarily as a reference when looking at the workflow through an XML editor.

5. **executionLabelExpression**: An optional attribute that can be used to describe an individual execution of your workflow. A *Workflow* is really a *definition* of a workflow, while a *Workflow Execution* is the result of running a workflow. The *executionLabelExpression* is a label for a single run of the workflow within Reach Engine and is evaluated each time that workflow is run.

   The text entered here displays in logs as well as in the Workflow tab of the Reach Engine UI. Expressions may be added here to make the values that display unique each time the workflow runs, based on the subject of the workflow.

6. **showInUserInterface**: An optional attribute that declares whether this workflow should display in the Reach Engine Studio UI. By default, the workflow displays as an action that users can take. However, you may want to hide your workflow from a user, for example, if your workflow includes other workflows that should only be triggered by a schedule or watchfolder, or you have workflows that only exist as sub-flows.

7. **subjectDOClassName**: An optional attribute that tells the Reach Engine what kind of data objects this workflow should be available to execute against. All data in Reach Engine, whether an asset, a clip, a project, or a user, is represented by a Data Object. Specify a *subjectDOClassName* to ensure:
   a. The Reach Engine validates that an appropriate Data Object is provided as the “subject” when your workflow starts.
   b. The Reach Engine Studio UI only displays your workflow against the appropriate Data Objects. For example, if you declare your *subjectDOClassName* as “Clip”, then your workflow only displays after one or more Clips are selected in the UI. If no value is specified, then the workflow applies globally.

8. **resultDataDef**: An optional attribute that specifies the name of a *contextDataDef* that represents the result of the workflow. Workflow results can be interpreted by the UI for display to the end user, and can be aggregated by the Execute Subflow Step to represent the result of the step.
Workflows are interesting because they can interact with files and send emails, but they are useful because they can also interact with Reach Engine DataObjects. The most common way for a workflow to interact with a data object is through its subject. A workflow’s subject is defined, in generic terms, in the subjectDOClassName attribute of the <workflow /> tag.

**Practice 1: Create a Workflow**

To write your first workflow, open your favorite text editor, create a new file, and name it HelloWorkflow.xml. Add the text below to your document. Every workflow you create should include this information. Specifically, it declares that you are working in the “workflow” namespace and tells your editor where to find the “workflow” XSD.

**Practice 1A: HelloWorkflow.xml**

```xml
<workflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://levelsbeyond.com/schema/workflow"
  xsi:schemaLocation="http://levelsbeyond.com/schema/workflow
  http://www.levelsbeyond.com/schema/workflow-2.0.xsd"

Next, add the following text to your HelloWorkflow.xml document. This text defines attributes that describe the workflow and will help you find it later. The workflow’s ID should be unique, and its name and description should contain helpful information about what the workflow does and why a user would want to run it.

**Practice 1B: HelloWorkflow.xml**

```xml
    id="HelloWorkflow"
    name="Hello Workflow"
    description="A workflow that says 'Hello'"
    subjectDOClassName=""
    deadlineExpression=""/>
```

Now that your workflow’s namespace and ID are defined, you are ready to add the steps that will define your workflow’s actions.

Refer to the Workflow XSD, as it defines how well-formed workflows XML should look and contains documentation and comments about individual workflow steps.
It’s a good idea to use an editor that can use the Workflow XSD to validate your Workflow files and help you with code-completion. Check out Oxygen, Eclipse, XMLSpy, or Komodo.

STEPS

Step elements represent the actual actions that the workflow will take. Each workflow is required to have an `<initialStepName>` element as the first child element in the workflow. Other step elements are used to further define the workflow.

Initial Step Name

All workflows must contain an `<initialStepName>` element as the first child element in the workflow. The `<initialStepName>` must refer to a valid step name in the workflow. That step will always be the first step run.

If the first thing you need to perform is conditional, and based on expression data, make your first step a `noOpStep` so that you can evaluate your transitions and move on to the correct “real” first action.

Step Elements

Step elements represent the actual actions that the workflow performs. You can use any step type that you have referenced via the XML namespace/schemaLocation in the workflow element. Different steps have different parameters, depending on the functionality of the step and the variables that are required to perform that step.

The following example uses a step from an add-on XSD, the nimbus-projects workflow:

```xml
<workflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://levelsbe<om.com/schema/workflow"
  xmlns:production="http://levelsbe<om.com/schema/workflow/production"
  id="_multiXsdExample"
  ...
> production:importFinalCutProject name="import project"
  projectExpression="${project}"
```
In this example, the nimbus-projects workflow XSD is imported into the workflow XML with the namespace alias “production”. This allows the nimbus-project step *importFinalCutProject* to be referenced by prefixing it with “production:”.

Each step type has different attribute requirements, but some attributes are common:

1. **name**: The step’s name is its identifier within the context of this workflow. *name* must be unique within the workflow and is used whenever you need to refer to a step in your workflow XML, such as the initialStepName element and transition targetStepName elements.

2. **displayName**: An optional attribute that gives the Reach Engine Studio UI a separate name to display in the workflow status interface. If you name your step more programmatically (e.g., “doSomething”), you might want to include a more human-readable *displayName* (e.g., “do something”).

3. **executionLabelExpression**: An optional attribute that evaluates an expression at runtime so that you can create a label with per-execution contextual data. This attribute works just like the workflow attribute of the same name.

4. **devStep**: An optional Boolean that, if true, causes the Reach Engine Studio UI to hide the step in the workflow status viewer. If you have a lot of noOpSteps that are simply evaluating conditions, you can hide them in the UI to reduce clutter by setting *devStep* to “true”.

5. **continueOnException**: By default, if an exception is thrown during the processing of a step, that exception is logged and the workflow execution moves into a STALLED state. If a particular step is not considered a requirement for the workflow to complete successfully, mark *continueOnException* as “true”. This logs the exception, but the workflow execution continues as if the step was successfully completed.

6. **priorityExpression**: An optional expression that evaluates to an integer value. Priorities are used with Workflow Queues to enable higher priority queued steps to be pulled from the queue before lower priority steps. See Workflow Queue documentation for details.

7. **pctComplete**: An optional integer that tells the workflow engine the percentage complete that the workflow is considered after this step completes. The workflow engine can provide incremental progress updates, given the previous step’s
pctComplete, this step’s pctComplete, and the percentage of the step execution itself that has been completed. Static pctCompletes are necessary because the workflow engine does not know how a workflow will execute given transition conditions.

See the step reference documentation for details.

**Practice 2: Add Steps to Your Workflow**

Add the bold text to your existing HelloWorkflow.xml file (content from Practice 1 is shown in gray).

**Practice 2: HelloWorkflow.xml**

```xml
<workflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://levelsbeyond.com/schema/workflow"
  xsi:schemaLocation="http://levelsbeyond.com/schema/workflow
  http://www.levelsbeyond.com/schema/workflow-2.0.xsd"
  id="HelloWorkflow"
  name="Hello Workflow"
  description="A workflow that says 'Hello'"
  subjectDOClassName=""
  deadlineExpression="">
  <initialStepName>Hello_Workflow</initialStepName>
  <testStep name="Hello_Workflow" outputExpression="Why, Hello Workflow."/>
</workflow>
```

These simple steps demonstrate how to say “Hello” within the Reach Engine.

The additional lines add the required `<initialStepName />` element to the workflow and add a `<testStep />` element as the second step.

The `<initialStepName />` element tells the workflow plugin which step it should execute first.

The `<testStep />` exists only to print messages to the logs, so it currently only has two attributes:

1. **name**: The name of your workflow step. Notice that the name is used to identify the step in the `<initialStepName />` element.
2. outputExpression: The expression that you’ll be printing to the logs. Expressions are extraordinarily powerful, and they’re typically used to inject dynamic information into workflows. In this example, the outputExpression is a literal string.

**TRANSITIONS**

In Workflow Step XML, transitions define how a workflow should proceed when an individual step completes. Transitions are declared within the step element and should precede other, step-specific elements.

**A simple Workflow Step with a transition**

```xml
<testStep name="say_hello" outputExpression="Hello. I am a simple workflow step. I just say hello. So, hello.">
    <transition condition="${true}">
        <targetStepName>
            say_something_else
        </targetStepName>
    </transition>
</testStep>
```

*Tip*

*If a workflow step fails to complete, none of its transitions are evaluated and the workflow execution stops. The workflow then enters a “STALLED” state, from which it can be resumed later.*

All transitions must contain two parts: a condition and a targetStepName. After a step completes successfully, the Reach Engine evaluates the expression in the condition attribute. If the expression resolves to true, the workflow continues to the step identified by the targetStepName element. By setting the condition attribute to true, this workflow’s author has guaranteed that this transition will always occur; that is, the step say_something_else will always execute after the step say_hello.

**Practice 3: Add Transitions**

Modify the “Hello_Workflow” step to include a transition that refers to a new step named “emailStep”.

In this section, you’ll add a second step to your workflow that sends a “Hello” message to a specified email address. This requires two new elements in the workflow: a new step and a transition.

Change your “Hello_Workflow” step by adding the following transition and new step (existing content from Practice 1 and 2 is shown in gray).
**Practice 3: HelloWorkflow.xml**

```xml
workflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://levelsbeyond.com/schema/workflow"
  xsi:schemaLocation="http://www.levelsbeyond.com/schema/workflow
  http://www.levelsbeyond.com/schema/workflow-2.0.xsd"
  id="HelloWorkflow"
  name="Hello Workflow"
  description="A workflow that says 'Hello'"
  subjectDOClassName=""
  deadlineExpression="">
  <initialStepName>Hello_Workflow</initialStepName>

  <testStep name="Hello_Workflow" outputExpression="Why, Hello Workflow.">
    <transition condition="${true}"
      <targetStepName>Email_Hello</targetStepName>
    </transition>
  </testStep>

  <emailStep name="Email_Hello"
    emailAddressesExpression="workflowGuru@levelsbeyond.com"
    subjectExpression="I'm just writing to say..">
    <body><![CDATA[
      HELLO!
    ]]]>
  </emailStep>
</workflow>
```

In the new code:

1. The transition condition is set to true, which means that this workflow always proceeds to the “Email_Hello” step.
2. The “Email_Hello” step sends a “Hello” message to a specified email address. Within the “Email_Hello” step, the following attributes are used:
   - **name**: As in the Hello_Workflow step, the step's name (Email_Hello) helps identify it in transitions and `<initialStepName />` elements.
- **emailAddressesExpression**: Defines the email addresses where the message will be sent. Like the expression in testStep, you’re using a literal string for now, but eventually you can inject dynamic information into the step using expressions.

- **subjectExpression**: Identifies the email’s subject.

- **body**: A CDATA block that describes the body of your email.

You can change the email address to your own, or spam Levels Beyond. Follow the steps in Practice 5 (Importing the Workflow) and Practice 6 (Running the Workflow) to execute the updated workflow.

**Tip**

*When you make changes to a workflow, but not its ID, you are creating a new version of the workflow. So when you tell Reach Engine Studio UI to run a workflow with a given ID, it always uses the latest version for the workflow execution!*  

If you check the logs, you can see the workflow executes your emailStep before it completes. If you updated the email address expression to use your email address, there will be an email in your inbox from Reach Engine Studio UI.

**Branching Workflows with Multiple Transitions**

Rarely does a workflow’s author want every step in a workflow to execute in the same linear order each time the workflow runs. Instead, authors may want to implement logic that allows for workflows to branch dynamically at run-time. Workflow transitions are the mechanism for this behavior. Consider the following workflow step:

```
<saveDataObjectStep devStep="true" name="set_order_status_pending_metadata" displayName="Update Order Status: Pending Metadata" dataObjectExpression="=subject">
  <transition condition="=subject.includeMetadata">
    <targetStepName>enter_pending_metadata_state</targetStepName>
  </transition>
  <transition condition="${true}">
    <targetStepName>set_order_status_active</targetStepName>
  </transition>
  <property name="orderStatus">PENDING_METADATA</property>
</saveDataObjectStep>
```

**Figure 3**: A more complex workflow step with multiple transitions before a step-specific element
The `saveDataObjectStep` defines two transitions. One directs the workflow to the `enter_pending_metadata_state` step, and the other to the `set_order_status_active` step.

Each transition condition is evaluated sequentially in the Workflow XML, until one of the statements evaluates to true, and the workflow moves on to the target step.

**Tip**

*Each transition condition within a step is evaluated in the order listed until one is true, then no other transitions are evaluated or executed. This behavior is similar to the “if, else if, else if logic” that can be found in programming languages.*

In Figure 3, the condition that is evaluated first is `=subject.includeMetadata`, which checks the subject of the executing workflow for the property “includeMetadata” (a Boolean in the example). If that property is true, the workflow executes the `enter_pending_metadata_state` step, and does nothing with `set_order_status_active` step. If “includeMetadata” does not resolve to true, the workflow then evaluates the next condition, which always evaluates to true, and advances to the `set_order_status_active` step.

**Best Practice**

*Always have one last transition that uses the condition “${true}”. This ensures the workflow knows how to behave even if every other statement is false, similar to the last “else” block in an “if, else, if else” control structure in a programming language. You may not want your workflow to ever be in this state, but this practice helps your workflow “fail gracefully” and send an email or print information to the logs about why it can’t proceed.*

**CONTEXT DATA DEFS**

Context DataDefs provide you with workflow variables that can be used to accept external input to your workflow, maintain and communicate data between steps and sub-flows, store the result of a workflow, and have a default value based on the fields of DataObject. For more detailed information about contextDataDefs, see the Workflow XSD or the workflow reference document.

Context DataDefs are created by specifying a name and a data type. Other attributes provide you with control over how the contextDataDef will be used and an optional default value expression. To conform to the Workflow XSD, they must be the last elements in your workflow.

Context DataDefs are defined just above the closing `</workflow>` tag. For example:
A simple `contextDataDef`:

```
<contextDataDef name="checkinDescription"
    dataType="String"
    required="true"/>
```

A `contextDataDef` contains the following attributes:

- **name**: The identifying name for this variable. The name must be unique within the workflow definition. The name is used throughout the workflow to call this value.

- **dataType**: Defines the kind of variable (i.e., data type) for the `contextDataDef`. See the Appendix A: Data Type Reference or the Workflow XSD for information about different data types.

- **defaultDataExpression**: An optional attribute that determines the default initialized value of the `contextDataDef`. If the `contextDataDef` is given a value from a user or config parameter during the workflow initialization, that value overrides the result of this expression.

  **Note**: A `defaultDataExpression` element is also available that allows for easier, multi-line string editing. If the `contextDataDef` contains content in both the `defaultDataExpression` element and attribute, preference is given to the attribute.

A `contextDataDef` with more attributes defined:

```
<contextDataDef dataType="String"
    name="testResults"
    defaultDataExpression=""
    multiple="true"
    required="true"
    userInput="true"
/>
```

Here is an example of this technique.

- **required**: Determines if the field is required. If this workflow is being run via the UI, it reinforces that users MUST enter a value before the workflow will run.
  
  - If `required= true` and no `defaultDataExpression` is provided, the UI prompts the user to enter data.
  - If `required = false`, `userInput` defines whether the UI prompts the user for input. You can always rely on the `defaultDataExpression` to ensure there’s not a null data state.
• Use the `userInput` boolean to determine whether the value displays in the UI or not and prompts the user for a value when the workflow is executed.

• **picklist**: A list of values that displays as a drop-down list in the UI for any field that is presented to the user for `userInput`.

• **multiple**: Boolean value that specifies if this variable can only hold one value (false) or multiple values (true). It provides the underlying data storage for the `contextDataDef` and typically contains an array or collection of values, hold strings, or files.

If `contextDataDef` is set as single (i.e., `multiple="false"`) and multiple values are presented to it, it only retains the first value. If a `contextDataDef` is set as multiple and a single value is presented to it, it works as a collection of values with only one element.

**A more complex contextDataDef**

```xml
<contextDataDef name="transcodedFile" dataType="File"/>
<contextDataDef name="transcodedFileName" dataType="String" userInput="true"
defaultDataExpression="${transcodedFile?.name}"
><picklist>
  <picklistItem value="${transcodedFile?.name}" label="File Name"/>
  <picklistItem value="${transcodedFile?.name}_${#getDay(date)}" label="Filename with Date"/>
</picklist>
</contextDataDef>
```

In the more complex contextDataDef example above, two dataDefs are defined.

• **transcodedFile**: In the Practice workflows, assume that the `transcodedFile` contextDataDef initially defaults to a null value, as it holds the result of a `convertVideoStep` that happens midway through the workflow.

• **transcodedFileName**: The other contextDataDef, `transcodedFileName`, stores the name of the transcoded file after the transcode has occurred. In other words, it has a `defaultDataExpression` that is only valid after the workflow has executed the transcode.

Note the “?” right after the `transcodedFile` reference in the `defaultDataExpression`. The ? is very important when writing expressions. If the expression was `${transcodedFile.name}` and `transcodedFile` is null, then the expression will fail when it attempts to reference the `.name` property. Adding ? tells the expression engine to ignore the rest of the path if `transcodedFile` is null, and simply return null.

By returning null if `transcodedFile` is null, the `transcodedFileName` contextDataDef continues to be evaluated until `transcodedFile` has a value, at which point the name is stored as the value of `transcodedFileName`.
A common beginner’s mistake when working with resultDataDef and contextDataDefs is to confuse them with expressions.

Expressions are bracketed with ${...} and often refer to dataObjects or dataDefs.

Reach Engine Workflow evaluates expressions and conditions using the Spring Expression Language (SPeL) at runtime. For example, the following expression passes the subject’s videoType field into a method, returning a string that the workflow will use:

```
sourceFileExpression="${subject.getSourceFileForConversion(subject.videoType)}"
```

The resultDataDef attribute, however, is not expecting an expression, but a direct reference to a contextDataDef in the workflow.

```
resultDataDef="episodeResult"
```

Note that there is no "${}" in this attribute, and the episodeResult contextDataDef is explicitly defined in the workflow.

```
<contextDataDef name="episodeResult" dataType="String"></contextDataDef>
```

**Practice 4: Branching and Context DataDefs**

Update your workflow to decide whether or not to execute the “Hello_Email” step based on user input.

To implement branching, you’ll need an additional transition in the “Hello_Workflow” step, as well as a new element, `<contextDataDef />`.

Update the “Hello Workflow” step to match the code below. (Existing content from previous practices is shown in gray).
### Area | XML
--- | ---
A | workflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://levels beyond.com/schema/workflow"


id="HelloWorkflow"
name="Hello Workflow"
description=""
subjectDOClassName=""
deadlineExpression="">

<initialStepName>Hello_Workflow</initialStepName>

B | <testStep name="Hello_Workflow" outputExpression="Why, Hello Workflow."

<transition condition="${sendEmail}"
<targetStepName>Email_Hello</targetStepName>
</transition>
<transition condition="${true}"
<targetStepName>Say_Goodbye</targetStepName>
</transition>
</testStep>

C | <emailStep name="Email_Hello"

eemailAddressesExpression="workflowGuru@levels beyond.com"

subjectExpression="I'm just writing to say..">

<body><![CDATA[
HELLO!
]]></body>
</emailStep>

D | <testStep name="Say_Goodbye"

outputExpression="I Didn't send an email because you told me not to. Goodbye."
</testStep>

E | <!--DATA DEFS -->
<contextDataDef dataType="Boolean"

name="sendEmail"
required="true">
</contextDataDef>

</workflow>
In the new code:

A. The workflow element information has not changed.

B. The original transition, which targets the “Email_Hello” step, no longer has true hard-coded into its condition attribute. Instead, the condition references the `sendEmail contextDataDef`, which the user will define.

This means when the “Hello_Workflow” step’s execution has completed, the workflow plugin begins evaluating transitions, one after the other. First, it evaluates the transition targeting “Email_Hello”. If it evaluates to true, “Email_Hello” executes and no other transitions are evaluated. If it does not evaluate to true, the next transition, targeting a new step named “Say_Goodbye”, is evaluated.

C. The “Email_Hello” step used in Practice 3 has not changed.

D. This transition’s condition is hard-coded to true, so if the first condition is false, the next step to execute is “Say_Goodbye”.

E. The `contextDataDef` element is used to determine whether or not execute the email step.

Save your workflow, then import it to the UI to test it.

**IMPORTING WORKFLOWS**

Workflows are imported into the Reach Engine using the Reach Engine Studio UI.

1. From the Reach Engine Studio UI, click the **admin** tab.
2. From the menu on the left, click the **Workflows** tab. The list of workflows displays.
3. Above the list, click the **Import** button.

4. Click the **Browse** button and select if the file is Remote or Local.
5. Navigate to the file, select it, then click **Open**.

6. Click **Upload**.

7. On a successful upload, a configuration dialog displays where you can modify some properties of the workflow.

<table>
<thead>
<tr>
<th>Cancel</th>
<th>Save</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Hello Workflow</td>
</tr>
<tr>
<td>ID</td>
<td>helloWorkflow</td>
</tr>
<tr>
<td>Enabled</td>
<td>✗</td>
</tr>
<tr>
<td>Visible</td>
<td>✗</td>
</tr>
<tr>
<td>Right-Click</td>
<td></td>
</tr>
<tr>
<td>Admin Only</td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td></td>
</tr>
<tr>
<td>Roles</td>
<td></td>
</tr>
</tbody>
</table>

- The workflow’s name attribute displays in the Name field. You can edit the name of the workflow after import, which changes the name of the workflow in the database.
- The workflow’s id attribute displays. This attribute cannot be modified.
- The Enabled option must be selected to use the workflow.
- The Visible option maps to the workflow’s `showInUserInterface` attribute. If selected, this workflow displays in the Reach Engine Studio UI.
- The Right-click option allows this workflow to be listed in the popup menu that displays when a user right-clicks on an item within the content list, as long as that item matches the data object type specified in the workflow’s `subjectDOClassName` attribute. Use this option to display key workflows and ensure the Flash limit for right-click options is not encountered.
- The Admin only option allows only users with the Admin role to access this workflow.
- Optionally, select the Groups that have access to this workflow. Groups are created in the Reach Engine Studio UI.
• Optionally, select the Roles that have access to this workflow. If the Admin option is selected, the Roles option is not available. Roles are also created in the Reach Engine Studio UI.

8. When you have specified the workflow options, click **Save**. The new workflow displays in the list of workflows.

To modify the workflow's options after it has been imported, double-click the workflow name within the list to display the options panel.

**EXECUTING WORKFLOWS**

Workflows are executed in three ways:

- **Watchfolder executions** occur when a file is dropped into a watchfolder associated with a specific workflow.
- **Scheduled executions** occur at a specific time.
- **User-initiated executions** occur through user interaction and are available through the UI.

See the [REST API Reference](#) for information about using the API to trigger workflow functions.

**WATCHFOLDER EXECUTIONS**

Workflows that interact with files are typically launched via a watchfolder. When a file is dropped into a folder that is associated with a specific workflow, the workflow executes its steps on the file.

To associate a workflow with a watchfolder:

1. **Create a specific watchfolder for the workflow.**
   You can create your watchfolder at any location that your Reach Engine install can see.

2. **Create a watchfolder configuration in local.reach-engine.properties.**
   Watchfolder configuration properties are grouped by a watchfolder config name. The basic syntax is:

   `Workflow.watchfolder.name_of_watchfolder_config.propertyName=value.`

Required properties for any watchfolder configuration include:

```
workflow.watchfolder.watchfolderTest.path=path/to/watchfolder
workflow.watchfolder.watchfolderTest.workflow=myWorkflow
```
workflow.watchfolder.watchfolderTest.fileDataDef=fileDataDefName

The path property declares where the watchfolder is located. The workflow property declares the ID of a workflow to run when a file is dropped in the path folder. The fileDataDef declares what dataDef should be populated with the incoming file. Other properties may also be set, but are optional (default values in parentheses):

workflow.watchfolder.watchfolderTest.pollingInterval=time in seconds (30)
workflow.watchfolder.watchfolderTest.extensions=valid, file, extensions (all)
workflow.watchfolder.watchfolderTest.deleteOnSuccess=true/false (false)
workflow.watchfolder.watchfolderTest.maxConcurrent=max concurrent files (1)

Any contextDataDef in your workflow can be initialized in the watchfolder configuration. Each property has the following format:

workflow.watchfolder.watchfolderTest.myDataDef=Bob

After files are moved to the watchfolder, they are then moved into various subfolders depending on their processing status.

- When the file is stable, it is moved into the received folder.
- When Reach Engine is able to work on the file (the file might be queued for processing), the file moves into the working folder.
- When the workflow succeeds, the file moves into the completed folder, or if the deleteOnSuccess folder is true the file is deleted.
- If the workflow fails, the file moves into the failed folder, along with a corresponding log file that provides information about why the workflow failed.

SCHEDULED EXECUTIONS

Reach Engine workflows can be executed using schedules. Scheduled workflow executions are currently only configurable via the local.reach-engine.properties file. Future releases will allow for schedule configuration via the Reach Engine Studio UI.

Configuring a scheduled workflow execution is very similar to a watchfolder configuration. Here is a sample basic configuration:

workflow.cron.cronTest.workflow=myWorkflow
workflow.cron.cronTest.schedule=*/2 * * * *

These two configuration properties are required for any scheduled workflow. Set the first property, workflow, to the ID of the workflow you want to execute. Set the second property,
.schedule, to a valid cron expression. Details on how to format a workflow cron expression can be found [here](#).

In addition to these two properties, workflow dataDef values may be set on the executed workflow, just like watchfolder configurations. For example, if `myWorkflow` has a dataDef called “maxAge”, you can initialize the value in the configuration:

```
workflow.cron.cronTest.maxAge=30
```

**USER-INITIATED EXECUTIONS**

You may want workflows that can be executed by users in the Reach Engine Studio UI. By default, workflows are exposed in the Reach Engine Studio UI for users to act upon. They can be associated to a specific Data Object class, so that the workflow only displays in the UI when a user selects a matching search result.

**Workflow Configuration**

Workflows are exposed in the UI by default. You can hide them from UI execution by not selecting Visible when the workflow is imported into the Reach Engine Studio UI. Or, you can set the `showInUserInterface` boolean on the workflow element to false:

```xml
<workflow id="hideInUI"
  showInUserInterface="false">
  ...
</workflow>
```

When `showInUserInterface` is set to true, the `contextDataDef` parameters may also need to be exposed to the user. Reach Engine currently supports dataDefs of types `String`, `Boolean`, `Integer`, `Double`, `Date`, `Date/Time`, `File`, and `Directory`. Other data types will be supported in a future release.

**Workflow Subjects**

If you set a `subjectDOClassName` value on the workflow to a valid Data Object class name, that workflow only appears in the UI when the user selects a matching search result. For example, if your workflow’s subject class is a Clip, when the user right-clicks on a Clip search result, your workflow appears. Your workflow then gets the selected Data Object as the subject of your workflow. See Practice 7, which specifies a Data Object class, and shows how it affects the UI.
Controlling the Execution Method

Normally, workflows run in the background. When a user starts a workflow execution, the server responds immediately and the user can move on and do other things, as a lot of video processing workflows can take minutes or even hours to complete. This is referred to as running a workflow asynchronously.

In some cases, however, the workflow is designed to quickly perform an action and respond. In these cases, you can set your workflow to run synchronously so that the user receives an immediate result. For example, you create a workflow that creates metadata XML for a clip and returns it to the user. This workflow will execute quickly because it doesn’t involve any long-running processes like transcodes or file delivery. In this case, you can tell the workflow to default its execution mode to be synchronous by setting the defaultExecutionMode to sync:

```
<workflow id="runSync"
  defaultExecutionMode="sync">
  ...
</workflow>
```

**Note**: This setting only applies to user-initiated workflow executions; watchfolder- and cron-triggered executions ignore this setting.

**Tip**

*Be careful when setting the defaultExecutionMode to sync. The UI times out after a couple of minutes waiting for a result, so make sure that your workflow runs quickly and does not have to wait for resources in order to complete."

**Practice 5: Running the Workflow**

Now that you’ve imported HelloWorkflow, you can run it.

**Reach Engine Studio UI**

By default, the Reach Engine Studio UI exposes your workflow in a variety of ways.

1. Log in to Reach Engine Studio UI.
2. From the content tab, click on the cog wheel in the upper-right corner. Select Hello Workflow.
3. The popup contains the sendEmail checkbox.
• If you check the sendEmail box and click the Submit button, an email is sent to the address specified in your workflow and the Email_Hello step displays in the workflow summary as complete.

• If you do not click the sendEmail checkbox and then click the Submit button, no email is sent but the workflow summary lists the Say_Goodbye step (rather than the Email_Hello step) as being complete.

In the Workflow Tab, you can see evidence that your workflow executed successfully:

Tip

Workflows execute asynchronously by default, so you will receive a response from the “start” request before the workflow has finished executing. For your practice workflows, these events all happen virtually simultaneously. For more advanced workflows that encode video and copy files, you will receive a response while the workflow continues to run for several minutes, or even hours!

Reach Engine UI

Now, run the workflow from the cogwheel in the upper-right corner of the Reach Engine UI. This time, the popup contains your practice code:
If you check the sendEmail box, you will see Reach Engine Studio UI send the “Hello” email when executing the workflow. Leave it unchecked, and it skips to the “Goodbye_Step”.

What would happen if the Reach Engine Studio UI didn’t add the sendEmail parameter to your request? The Spring Expression Language and Workflow Plugin evaluate null to false. However, you should always give contextDataDefs a default value if you can:

```xml
<contextDataDef dataType="Boolean" name="sendEmail" userInput="true" defaultDataExpression="${false}"/>
```

**An Exercise**

Try to alter HelloWorkflow so the user supplies an email address when executing the workflow. **Hint:** You’ll need a new contextDataDef to hold the email address information.
Workflow Step Queues

Workflow step queues allow you to place limitations on the number of simultaneous executions of a particular step. A queue is configured with selection criteria. If a step execution matches the selection criteria, it qualifies for the queue. Queues are configured with a maximum number of active steps allowed. If a qualifying step matches the queue’s criteria and the maximum number of steps are currently being executed, the step is queued and waits until a slot opens up.

CONFIGURATION

Workflow queues are configurable either through a startup config file or via the Reach Engine UI (coming soon).

To configure a queue via the startup config file, create or open the file localContext.xml located in the TOMCAT_HOME/lib folder. Add the following XML:

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean class="com.routeto1.services.system.RuntimeConfigPropertyPlaceholderConfigurer">
        <property name="ignoreResourceNotFound" value="true" />
        <property name="locations">
            <list>
                <value>classpath:local.reach-engine.properties</value>
            </list>
        </property>
    </bean>

    <bean id="transcodeWorkflowQueue" parent="baseDataObjectInjector">
        <property name="keyPropertyName" value="name"></property>
        <property name="injectionData">
            <bean class="com.routeto1.data.spring.DataObjectInjectionData">
                <property name="dataObjectClassName" value="WorkflowQueue"></property>
                <property name="runtimeOverrideAllowed" value="true"></property>
            </bean>
        </property>
    </bean>
</beans>
```
EXPRESSIONS

Reach Engine Expressions provide the real power in workflows. Expressions allow you to substantially customize the generic Reach Engine workflow steps to fit your domain model and needs. Besides data access, there are built in functions, conditions, and collection filtering.

The Reach Engine Expression syntax is built off of Spring Expressions (SpEL). There are basic syntax reference guides available here and here. All of the information included in the guides is valid when writing Reach Engine Expressions. Levels Beyond has extended the syntax to make referencing things like DataObject path data easier.

EXPRESSION DELIMITERS

Expressions are recognized as such by the use of delimiters. There are two basic forms that an expression can take:

=expression

or

${expression}$
Levels Beyond recommends that you standardize your expressions on the ${} syntax, because this syntax allows you to create “mixed” expressions, which are strings with both literal and expression content. For example:

testExpression="Hello there, ${fullName}!”

The example is an expression that is almost entirely literal. Only ${fullName} is evaluated as an expression. When you use the =expression syntax, you must use the concatenation operator (+) to achieve the same value:

testExpression=”='Hello there, ‘ + fullName + ‘!’”

This form is much harder to read and more awkward to write.

**Best Practice**

It is fine to use the =expression syntax for simple things like =true, but the best practice is to use the ${} syntax at all times. E.g., ${true}

**EXPRESSION ROOTS**

All valid expressions start at a root. The root is the first thing mentioned in the expression. The following are all valid roots in Reach Engine:

- Context Data Defs
- The workflow subject (if available)
- A function

```xml
<workflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://levelsBeyond.com/schema/workflow"
         xsi:schemaLocation="http://levelsBeyond.com/schema/workflow
         http://www.levelsBeyond.com/schema/workflow-2.0.xsd"
         id="expressionExample"
         name="Expression Example"
         subjectDOClassName="Clip"
>
  <initialStepName>log stuff</initialStepName>

  <testStep name="log stuff"
          outputExpression="Sample expressions:  ${dataDefVar} ${subjectVar}"

```
Figure 4: A workflow showing examples of the three Reach Engine expression roots

In this example, three different contextDataDefs are defined with different defaultDataExpressions, and each uses a different expression root type.

- The **dataDefVar’s** expression is `${fullName}`. This tells the workflow engine to look for a contextDataDef called “fullName” and set the value of `dataDefVar` to the value of `fullName`.

- The **subjectVar’s** expression is `${subject.name}`. `subject` is a special expression root that is only available if the workflow has a `subjectDOClassName` defined. From `subject`, you can reference properties and associations of the DataObject class defined in `subjectDOClassName`. This example references the name property of a Clip DataObject.

- The **functionVar’s** expression is `${#newDate()}`. Functions can be used almost anywhere in an expression, including the root, and are always prefixed with a `#` symbol. Refer to the Reach Engine Expression Function Reference for details on the
various functions available. The `newDate()` function returns a new Date or Date/Time value that represents the current date and time.

**EXPRESSION PATHS**

After you reference something in an expression using one of the roots, you gain access to the various properties and methods of the result of that expression. The various data types available in Reach Engine Workflow offer different properties and methods. Both properties and methods are referenced by putting a dot (.) after the expression variable. For example:

- `stringVar.trim()` Assuming `stringVar` is a String, this expression invokes the trim method, which returns a new String with all leading and trailing white space removed.
- `fileVar.absolutePath` Assuming `fileVar` is a File, this expression returns a String with the file’s full path.

**ACCOUNTING FOR NULL VALUES**

A common mistake when writing expressions is to assume that your expression path components will all be null. This creates problems when an expression path component, that is not the last component, is null. For example:

```java
subject.asset.originalContent
```

So long as `subject` and `.asset` are not null, this expression path is fine, but if one of those path components is null, an expression exception occurs and the workflow stalls; this may be what you want. If your workflow expects these values to be there and they are not, that may represent an error condition and a stalled workflow is the desired end result.

There may also be situations where an expression component may be null for a while, meaning that it will be filled later in the workflow. In these cases, you can make your expression null-safe by adding a question mark (?) to the end of your path component name. In the previous example, the expression is rewritten to:

```java
subject?.asset?.originalContent
```

Then, if either `subject` or `.asset` resolves to null, the expression stops and returns null.

See the [SpEL documentation](#) under 6.5.15: Safe Navigation Operator for more information.
FUNCTIONS

Expression functions provide you with a pre-built set of operations that you can use on your expression results. Reach Engine Studio UI ships with a number of utility functions that can help you obtain and manipulate asset, file, date, and system data. See the Reach Engine Expression Reference for details on how to use these functions.

ADVANCED EXPRESSIONS

After you have a good sense of basic expression syntax, start using Advanced Expressions. Study the Elvis/ternary operators, collection selections, and collection projections. These operations provide amazing power and flexibility for your expressions so that you can work with exactly the data you need to make your workflows go.

Practice 6: Determine what these expressions do

For extra credit, can you figure out what these expressions are trying to do?

1. `<contextDataDef name="inputDir" dataType="Directory" multiple="false" required="true"/>

2. `<contextDataDef name="files" dataType="File" multiple="true" defaultDataExpression="${#ls(inputFile, true, false, 'mov')}/">

3. `<contextDataDef name="mezzContents" dataType="Data Object" multiple="true" defaultDataExpression="${assets.![mezzanineContent]}"/>

Answers

1. This expression required user input: required="true" and there is no default dataDef.

2. This expression finds all the mov files (dataType="File") in the directory and puts them in the ls directory: defaultDataExpression="${#ls(inputFile, true, false, 'mov')}".

Data Objects

The Data Object type ties a workflow into Reach Engine’s metadata model. While you can store any Data Object in a dataDef of this type, you must make sure that you know what you are getting, or your path expressions won’t work correctly.
The following common properties are available from Data Objects, along with their return type. For associations, the return type is one or multiple Data Objects, meaning that you can “dot down” through several related objects to obtain the data you need. For a complete listing of available properties and associations, refer to the Reach Engine Metadata Model Guide (coming soon).

**ASSETS**

The Asset class represents any repository-managed asset in the system. Asset has two types: AssetMaster and AssetVersion.

**Properties**

- name (String)
- filePath (String)

**Associations**

- metadata (Metadata, single)
- thumbnail (ImageAssetContent, single)

**Practice 7: Use a “Clip” as the subjectDOClassName**

Update HelloWorkflow to use “Clip” as its subjectDOClassName.

The subjectDOClassName attribute informs the Workflow Plugin that every instance of HelloWorkflow will interact with a “Clip” dataObject. However, each execution of the workflow might focus on a different instance of a clip.

```xml
<wflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://levelsbeyond.com/schema/workflow"
      xsi:schemaLocation="
          http://levelsbyond.com/schema/workflow
          http://www.levelsbeyond.com/schema/workflow-2.0.xsd"
      id="HelloWorkflow"
      name="Hello Workflow">
```
After you update HelloWorkflow to use “Clip” as its subject, import the workflow, log in to Reach Engine Studio UI and run the workflow.

When you select the cog wheel in the upper-right corner of the Reach Engine Studio UI, “HelloWorkflow” is gone. What happened?

Reach Engine Studio UI automatically filters workflows by subject. Initially, when you did not set a subject for HelloWorkflow, it was by default a *global workflow*, which made it available from the cog wheel in the UI. Now that “HelloWorkflow” expects a *Clip* as its subject, you must find a clip in the UI and right-click on the clip to expose “HelloWorkflow”.

In the image below, the user right-clicked on a clip and HelloWorkflow displays in the popup menu.
When you run the workflow from the right-click prompt, its subject is defined and can be referenced throughout the workflow XML. For example, consider the following change to the "Email_Hello" step.

```xml
<emailStep name="Email_Hello"
  emailAddressesExpression="ecobb@levelsbeyond.com"
  subjectExpression="I'm just writing to say..">
  <body><![CDATA[
    HELLO! You should really take a look at the clip ${subject.name}.
  ]]]</body>
</emailStep>
```

The workflow's `subjectDOClassName`, an instance of a "Clip", can be referenced directly inside of the workflow XML with `${subject.name}`.

Here is the entire workflow:

```xml
<workflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://levelsbeyond.com/schema/workflow"
  id="HelloWorkflow"
  name="Hello Workflow"
  description=""
  subjectDOClassName="Clip"
  deadlineExpression="">
  <initialStepName>Hello_Workflow</initialStepName>
  <testStep name="Hello_Workflow" outputExpression="Why, Hello Workflow.">
    <transition condition="${sendEmail}"
      <targetStepName>Email_Hello</targetStepName>
    </transition>
    <transition condition="${true}"
      <targetStepName>Say_Goodbye</targetStepName>
    </transition>
  </testStep>
  <emailStep name="Email_Hello"
```
After you update your HelloWorkflow.xml document, import it to Reach Engine Studio UI, then right-click a clip object in the Reach Engine Studio UI to run your workflow. If you check the “sendEmail” checkbox, the resulting email now includes the clip’s name in your email’s body.
Appendix A: Workflows

Reach Engine Studio UI comes with its own set of built-in workflows. One of the most commonly used is the Ingest Video Asset workflow. The Ingest Video Asset workflow may trigger a number of other smaller workflows (subflows) to complete the ingestion, depending on the flow of the ingested video. In the image below, green boxes indicate subflows. The subflows are:

- Create Mezzanine Video
- Create Proxy Video
- Create Thumbnail Video

Note: All images look much better in the Reach Engine Wiki.
INGEST VIDEO WORKFLOW
CREATE MEZZANINE SUBFLOW

The Create Mezzanine subflow creates a mezzanine video as part of the video ingestion if a mezzanine doesn’t already exist.
CREATE PROXY VIDEO SUBFLOW

The Create Proxy Video subflow creates a proxy video as part of the video ingestion, if a proxy video doesn’t already exist.
CREATE THUMBNAIL VIDEO SUBFLOW

The Create Thumbnail subflow creates a thumbnail video as part of the video ingestion if a thumbnail doesn’t already exist.
Appendix B: Data Type Reference

String

Strings are a set of characters. They do not have any special awareness of their contents, so even if a String has all numbers in it (“123”), you cannot perform math functions on it. In fact, if you have two Strings in an expression and try to add them with the + operator, the result is a concatenation, not an addition!

At runtime, a String is represented by a Java String and thus has access to all String properties and methods. See the String Javadoc page for information on what is available. Commonly used String methods include trim(), startsWith(), endsWith(), toUpperCase(), and toLowerCase().

Boolean

Booleans are true/false values. They are extremely useful for workflow conditions but don’t really have any additional properties or methods.

Integer

Integers are integral or whole numbers; any value past the decimal point is stripped and ignored. Therefore, if you set an Integer to a decimal value like 29.97, it is stored as 29.

You can perform math functions with Integers. The Integer type is represented as a Java Integer so there are a few methods available. See the Javadoc page for details.

Double

Doubles represent floating-point numbers (or fractional numbers). Otherwise they act much like Integers. The Double type is represented as a Java Double.

Date

Dates store only the date portion of a date/time; i.e., the time portion is ignored. For comparison purposes, a Date is considered to be midnight of the specified date. Dates are represented as a Java Date.

There are many date functions available, check out the Reach Engine function reference for details.

Date/Time

Date/Times act just like Dates except that the Time portion is preserved.
File

The File is one of the more powerful data types in Reach Engine. A File represents a file on a file system that Reach Engine can read; that file system can be local, SAN, NFS, or even S3. You can declare File dataDefs for input or output. If you declare a workflow result to be a File dataDef, the result file can be downloaded from the client application. There are several properties and methods available from a File:

Properties

- Name
- Checksum
- parentFile

Methods

- absolutePath()
- exists()
- canWrite()
- isHidden()
- lastModified()
- length()

Additionally, there are a number of file functions available.

Directory

Directories work just like Files, except that they represent file directories, not files themselves. Directories have most of the same properties and methods as File plus a couple of extras:

- listFiles()
- mkdirs()
Appendix C: Elements and Attributes

The following lists of elements and attributes are used within this document. For a complete list of elements and attributes, refer to the Reach Engine Workflow XSD.

<table>
<thead>
<tr>
<th>Elements</th>
<th>Description</th>
</tr>
</thead>
</table>
| <contextDataDef>      | User-definable variables that are the primary way of communicating values to workflows and between steps and sub-flows. They can be used to accept external input to your workflow, maintain and communicate data between steps and sub-flows, store a result of a workflow, and have a default value based on the fields of DataObject.  
**Note:** Context data defs must be the last elements in your workflow. |
<p>| &lt;defaultDataExpression&gt; | An element within context data defs that provides an alternative to using the defaultDataExpression attribute. This element allows for easier, multi-line string editing. If the context data def contains content in both the defaultDataExpression element and attribute, preference is given to the attribute. |
| &lt;initialStepName&gt;     | <strong>Required</strong> element that must be the first child element in the workflow.                       |
| &lt;steps&gt;               | Represents the actions that the workflow takes.                                                   |
| &lt;transition&gt;          | Defines how a workflow should proceed when an individual step completes. Transitions are declared within the step element, and should precede other, step-specific elements. |
| &lt;workflow&gt;            | <strong>Required</strong> element that is the container for the workflow’s actions.                           |</p>
<table>
<thead>
<tr>
<th>Attributes</th>
<th>Element Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>Transition</td>
<td><strong>Required</strong> expression that is evaluated after a step completes successfully. If expression resolves to true, the workflow continues to the step identified in the targetStepName attribute.</td>
</tr>
<tr>
<td>continueOnException</td>
<td>Step</td>
<td>By default, when an exception is thrown during a step, the exception is logged and the workflow moves to a Stalled state. If a step is not considered a requirement for workflow completion, set this attribute to true to log the exception but continue workflow execution.</td>
</tr>
<tr>
<td>dataType</td>
<td>Context data defs</td>
<td>Optional attribute that defines the variable type for the data def.</td>
</tr>
<tr>
<td>defaultDataExpression</td>
<td>Context data defs</td>
<td>Optional attribute that determines the default initialized value of this contextDataDef. If the contextDataDef is given a value from a user or config parameter during workflow initialization, that value overrides the result of this expression. <strong>Note:</strong> A defaultDataExpression element is also available for multi-line string editing. If both the attribute and element contain content, the attribute is given preference.</td>
</tr>
<tr>
<td>defaultExecutionMode</td>
<td>Workflow</td>
<td>Optional attribute that defines whether the workflow should run synchronously or asynchronously. This attribute only applied to user-initiated workflows. When a workflow is triggered by a watchfolder or by a timed event, this attribute is ignored. This attribute accepts</td>
</tr>
<tr>
<td>Attributes</td>
<td>Element Location</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>Workflow, defaultDataExpression</td>
<td>Optional attribute that details what the element does. Primarily used for reference when looking at workflows through an XML editor.</td>
</tr>
<tr>
<td>devStep</td>
<td>Step</td>
<td>Optional Boolean attribute that, if true, causes the Reach Engine Studio UI to hide the step in the workflow status viewer.</td>
</tr>
<tr>
<td>displayName</td>
<td>Step</td>
<td>Optional attribute that is used to display an alternate name within the workflow status interface.</td>
</tr>
<tr>
<td>executeSubflowStep</td>
<td>Step</td>
<td>Prior to this version 1.5, the executeSubflowStep would wait for a subflow’s completion before continuing. Additionally, the status of a subflow would determine the status of the executeSubflowStep. Even if a single stall would result in the parent step stalling as well. With version 1.5, the executeSubflowStep has a &quot;waitForCompletionExpression&quot; attribute. This attribute determines whether the step waits for its subflows to complete before proceeding, or if it immediately completes and transitions to the next step. Also, if the waitForCompletionExpression attribute evaluates to false, subflows that fail or stall will not cause the parent step to stall as well. The value of this attribute defaults to true, which causes the executeSubflowStep to</td>
</tr>
<tr>
<td>Attributes</td>
<td>Element Location</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>behave exactly as it has in previous versions of Reach Engine Studio.</td>
</tr>
<tr>
<td>executionLabelExpression</td>
<td>Workflow</td>
<td>Optional attribute that is used to describe an individual execution of your workflow. This text displays in the logs and workflow tab of the Reach Engine Studio UI. Expressions are allowed to make the values unique each time the workflow runs. The execution label expression is evaluated multiple times over the lifetime of the workflow execution.</td>
</tr>
<tr>
<td>id</td>
<td>Workflow</td>
<td>Required identifier that must be unique within your Reach Engine installation.</td>
</tr>
<tr>
<td>multiple</td>
<td>Context data def</td>
<td>Optional Boolean attribute that defines whether this data def contains multiple values or not. Defaults to false. If a dataDef is set as single (multiple=false) and multiple values are presented to it, it will only retain the first value. If a dataDef is set as multiple and a single value is presented to it, it will work as a collection of values with only one element.</td>
</tr>
<tr>
<td>name</td>
<td>Workflow, Step, Context data def</td>
<td>Required attribute that identifies the element. For workflows, the name displays throughout the Reach Engine Studio UI. For steps and contextDataDefs, the name must be unique within the workflow. The name is used to identify the element within the context of the workflow.</td>
</tr>
<tr>
<td>noOpSteps</td>
<td>Step</td>
<td>Optional attribute to evaluate transitions and move on to the correct “real” first action.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Element Location</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>outputExpression</td>
<td>Step</td>
<td>Optional expression that is printed to the logs.</td>
</tr>
<tr>
<td>pctComplete</td>
<td>Step</td>
<td>Optional integer that identifies the workflow’s percentage complete once this step finishes.</td>
</tr>
<tr>
<td>priorityExpression</td>
<td>Step</td>
<td>Optional expression that should resolve to an integer to determine priority order if this step qualifies for a workflow queue. If queues are not being used, this expression will have no effect.</td>
</tr>
<tr>
<td>required</td>
<td>Context data def</td>
<td>Optional Boolean attribute that determines if the field is required. If this workflow is being run via the UI, it will enforce that users MUST enter a value before the workflow will run.</td>
</tr>
<tr>
<td>resultDataDef</td>
<td>Workflow</td>
<td>Optional attribute that specifies the name of a context data def that represents the result of the workflow.</td>
</tr>
<tr>
<td>showInUserInterface</td>
<td>Workflow</td>
<td>Optional Boolean attribute that defines whether or not the workflow displays in the Reach Engine Studio UI.</td>
</tr>
<tr>
<td>subjectDOClassName</td>
<td>Workflow</td>
<td>Optional attribute that defines the type of data object against which the workflow should be available to be executed.</td>
</tr>
<tr>
<td>targetStepName</td>
<td>Transition</td>
<td><strong>Required</strong> attribute that identifies the workflow’s next step.</td>
</tr>
<tr>
<td>userInput</td>
<td>Context data def</td>
<td>Optional Boolean attribute that determines whether to prompt the user for a value for the context data def variable in the UI when the workflow is executed.</td>
</tr>
</tbody>
</table>
Appendix D: For More Information

- Oxygen, Eclipse, XMLSpy, and Komodo are XML editors that can assist with validating and coding your workflow files.

- Spring Expression Language (SpEL) is the syntax off which Reach Engine Expressions are built. Basic syntax reference guides are available here and here.

- The default Reach Engine Workflow XSD location is http://www.levelsbehind.com/schema/workflow-2.0.xsd.

- Java SE 6 reference page