Open RPA Documentation

Release 0.2.8

Balaraman Kunduvara, Madhav Sivadas, Sivakumar Madhavan

Mar 17, 2018

Contents:

CHAPTER 1

Automation Framework

Here is a quick introduction and few steps to get you started! Are you ready?

1.1 Introduction

OPAL is an Automation Framework for process and application automation. Guess what? It is an open source framework and free to use. Yippe! Now there is nothing stopping you from implementing all those creative process improvements projects you had in your mind.

OPAL is based on the Node-RED project by IBM. Node-RED was primarily for Internet of Things. OPAL has extended the capability of automation to almost any other application in your daily use.

CHAPTER 2

Getting Started

Let us get the tool installed. You can refer to the steps provided in the rst_setup

2.1 Terms

Node- A action that is performed. Nodes are available in the palette. Read the *opalnodes* documentation to view all nodes.

Properties - The properties of a node are the list of attributes or values that is required for executing a node. They are configured through the editor and passed to the run-time as part of the generated json configuration.

Module - A collection of related nodes that are published together to the repository. They are installed and managed as a collection and they mostly have their own palette category.

Flow - A sequence of nodes that represent tasks performed on various resources. The resources maybe UI based like TextField, Button etc or Non-UI resources like File, Webservice, Email.

Context - Refers to the state that is available to a node. This could be any valid JS types. There are 3 types of context namely; Global, Flow and Node.

Project - A collection of flows that contains a deployed unit. Typically a process to be automated

Editor - The user interface that is used to compose flows. Editor has sections to define flows and resources. It shows debug messages, help associated with various nodes and managing projects.Flows are composed by configuring nodes and connecting them together as the process dictates.

Runtime - The engine that executes the flow once it is deployed from the Editor

Object Finder - The component integrated with studio that allows finding UI objects on various types of application.

2.2 Opal Nodes

opal-node-control-flow ———————– **This module is part of the OPAL framework**

2.2.1 opal-node-control-flow

Nodes: * opal loops

Loops * Provides `for-loop`, `foreach-loop`, `while-loop` functionality. THe loop node provides 2 output ports. The next item `output` and the `done` output port. Connect the nodes that are to be in the loop body to the `next` output port. At the end of the loop connect the last action to the input port of the loop node. Connect the `done` output port to the nodes that are to be executed after the loops ends.

Inputs:

- **Name**: A Name for this instance
- **Type**: choose from `for-loop`, `foreach-loop`and `while-loop`
- **for-loop**: Takes start, end and step as parameters
  - **start**: A number that represents the starting value for the loop
  - **end**: A number that represents the ending value of the loop. This must be greater than the start value
  - **step**: A number y which the loop counter is incremented for each iteration
- **foreach-loop**: Takes a collection to be iterated as input.
  - **collection**: The collection to be iterated. Expects a JSON array or variable containing a json array.
- **while-loop**: Takes start, end and step as parameters
  - **property**: The value to be used in the comparison for while condition
  - **rules**: A collection of rules that can be applied to the property. The rules when applied to the property forms the condition for the loop

Outputs: `msg.error` When an error happens contains the error message from the read operation `msg.payload` contains the data read from spreadsheet

2.2.2 opal-node-database

This module is part of the OPAL framework

Nodes: * querydb-read

Querydb Read Connects to mysql database based on the connections details provided and executes the specified query. Query results are returned as the output of the node.

Inputs:

- **DB Type**: Specify the type of database to connect (e.g. mysql, oracle, db2, etc.)
- **Server**: Specify the database server details (IP address or hostname)
- **Database**: Specify the database instance name
- **Port**: Specify the port number to be used for connection
- **Connection String**: Specify the connection string for the database.
- **Username**: Specify the username for connecting to the database.
- **Password**: Specify the password for connecting to the database
- **Query**: Specify the query to be executed on the database

- **Timeout**: specify the timeout during lookup for the title operation

Outputs: `msg.error` When an error happens contains the error message from underlying database connection `msg.payload` contains the data read from database query

2.2.3 opal-node-email

This module is part of the OPAL framework

Nodes: * email send * email listen

Email Send Sends the `msg.payload` as an email, with a subject of `msg.topic`.

The default message recipient can be configured in the node, if it is left blank it should be set using the msg.to property of the incoming message. If left blank you can also specify `msg.cc` and/or `msg.bcc` properties.

You may optionally set `msg.from` in the payload which will override the userid default value.

The payload can be html format.

If the payload is a binary buffer then it will be converted to an attachment. The filename should be set using `msg.filename`. Optionally `msg.description` can be added for the body text.

Alternatively you may provide `msg.attachments` which should contain an array of one or more attachments in nodemailer format.

If required by your recipient you may also pass in a `msg.envelope` object, typically containing extra from and to properties.

Note: uses SMTP with SSL to port 465.

Email Listen Repeatedly gets a single email from an IMAP server and forwards on as a msg if not already seen.

The subject is loaded into `msg.topic` and `msg.payload` is the plain text body. If there is text/html then that is returned in `msg.html`. `msg.from` and `msg.date` are also set if you need them.

Additionally `msg.header` contains the complete header object including to, cc and other potentially useful properties.

Uses the imap module.

Note: this node only gets the most recent single email from the inbox, so set the repeat (polling) time appropriately.

Note: uses IMAP with SSL to port 993.

Any attachments supplied in the incoming email can be found in the `msg.attachments` property. This will be an array of objects where each object represents a specific attachments. The format of the object is:

```
{
  contentType:        // The MIME content description
  fileName:           // A suggested file name associated with this attachment
  transferEncoding:   // How was the original email attachment encoded?
  contentDisposition: // Unknown
  generatedFileName:  // A suggested file name associated with this attachment
  contentId:          // A unique generated ID for this attachment
  checksum:           // A checksum against the data
  length:             // Size of data in bytes
  content:            // The actual content of the data contained in a Node.js Buffer␣
↪object
```

```
                    // We can turn this into a base64 data string with content.
↪toString('base64')
}
```

2.2.4 opal-node-ftp

This module is part of the OPAL framework

Nodes: * ftp service

FTP Service Connects to a ftp server and helps to read or write the specified file.

Inputs: * **Name**: A Name for this instance * **Host**: Specify the port number of FTP server * **Port**: Specify the port number of FTP server * **Filename**: Specify the filename to be transferred * **Mode**: Specify the type of operation. READ / WRITE a file * **Location**: Specify the location of the file to be uploaded * **Remote Location**: Specify the remote location of the file on the FTP server * **Username**: Specify the username for connecting to FTP server * **Password**: Specify the password for connecting to FTP server

Outputs: `msg.error` When an error happens contains the error message from the read operation `msg.payload` contains the data read from spreadsheet

2.2.5 opal-node-msexcel

This module is part of the OPAL framework

Nodes: * read excel * write excel

Read Excel Opens an ***xlsx*** or ***xls*** file spreadsheet from a specified location and reads the contents. The contents can then be passed to downstream node or saved to a store variable

Inputs:

- **Name**: A Name for this read-excel instance
- **File Path**: choose the location from where the spreadsheet can be read
- **Sheet**: specify the name of the sheet to be read (E.g. Sheet 1)
- **Read Mode**: specify the mode used for reading. This can be Full Content,Rows,Columns,Region
  - **Full Contents** - Fetches the entire contents of the specified sheet
  - **Rows** - A comma separated list of Numbers of the Rows that are to be fetched.
  - **Columns** - A comma separated list of Names(e.g. A, B, AA etc) of the columns that are to be fetched
  - **Region** - A region notation as defined by this MSDN article (Range Notation). Only a single range expression is supported.

Output Formats: The output of the read operation can be a Simple JSON (Array of Arrays or Column Header indexed objects) with only the data or a more descriptive format. Details can be found from the xlsx project
* **As Json** - Uses the simple JSON format * **Use Column Labels** - Use column headers such A,B etc to index the data rather using Array of Array. Only for JSON mode * **Remove Empty Rows** - Removes Empty Rows fromm a JSON output. Only for JSON mode * **Timeout**: specify the timeout during lookup for the title operation.

Outputs: `msg.error` When an error happens contains the error message from the read operation `msg.payload` contains the data read from spreadsheet

Write Excel Creates or Opens an xlsx/xls file spreasheet from the specified location and write the contents that are provided. The contents can be from upstream actions, variables (flow/global) or provided as literal text

Inputs: * **Name**: A Name for this read-excel instance * **File Path**: choose the location where the spreadsheet is to be written * **Sheet**: specify the name of the sheet to be updated (E.g. Sheet 1) * **Write Mode**: specify the mode used for writing. This can be Full Content,Rows,Columnns,Region * **Full Contents** - Updates the entire contents of the specified sheet - TBD * **Rows** - A comma separated list of Numbers of the Rows that are to be updated. * **Columns** -A comma separated list of Names(e.g. A, B, AA etc) of the columns that are to be updated * **Region** - A region notation as deinfed by this MSDN article (Range Notation). Only a single range expression is supported. * **Timeout**: specify the timeout during lookup for the title operation.

Input Formats: The input data to be written to the spreadsheet is always provided as an array of arrays. This is expected to be a valid json. Use the below guidelines. A JSON formatter will be available in a later release.

- **Number** - Uses the simple JSON format
- **String** - Use column headers such A,B etc to index the data rather using Array of Array. Only for JSON mode
- **Boolean** - Removes Empty Rows fromm a JSON output. Only for JSON mode
- **Date** - Specified like a string with surrounding quotes. The list of date formats shown below are support and will be automatically parsed. Others maybe set as text. The list of supported formats will be configurable in a later release

Outputs: `msg.error` When an error happens contains the error message from update operation

2.2.6 opal-node-pdfreader

This module is part of the OPAL framework

Nodes: * read pdf

Read PDF Opens an pdf from a specified location and reads the contents. The contents can then be passed to downstream node or saved to a store variable.

Inputs: * **Name**: A Name for this instance * **File Path**: choose the location from where the pdf can be read * **Preprocess**: specify if pdf document needs to be processed, optionally based on rules specified.

Output Formats: The output of the read operation can be a JSON

Outputs: `msg.error` When an error happens contains the error message from the read operation `msg.payload` contains the data read from pdf

2.2.7 opal-node-soap

This module is part of the OPAL framework

Nodes: * soap service

Soap Service Connects to a soap service and provides the response.

Inputs:

- **Name**: A Name for this read-excel instance

- **Url**: Specify the url for soap request
- **Method**: Specify the methods to invoke based on the wsdl
- **Parameters**: Specify the parameters in json format
- **Headers**: Specify the header for the request

Outputs:

`msg.error` When an error happens contains the error message from the read operation `msg.payload` contains the data read from spreadsheet

2.2.8 opal-node-ui-web-browser

This module is part of the OPAL framework

Nodes:

- open web
- close web
- find object
- send keys
- click on
- set value
- to file
- identify page
- get value
- get attribute
- get text
- run script
- screenshot
- nav to
- nav back
- nav forward
- nav refresh

Open Web Create an instance of selenium webdriver and connect to the Selenium Server when an event is triggered.

Inputs: * **Name**: A Name for this read-excel instance * **Browser**: specify the selenium node which support the web browser you wan tot test such as [Chrome, Firefox etc.,] * **Page**: the name of the page to be launched. pages can be defined by editing this configuration item * **Timeout**: specify the timeout during lookup for the title operation.

Outputs: `msg.error` When an error happens contains the error message from the read operation.

Close Web Close the web browser which is openned by the open-web node

Inputs: * **Name**: a Name for this instance * **Wait For**: time in milliseconds to waiit before execution

Outputs: `msg.error` When an error happens contains the error message from the read operation

Find Object Finds a target specified in the specified page

Inputs: * **Name**: a Name for this instance * **Page**: the page where this browser action executes. Pages can be defined by editing this configuration item * **Target**: specify the target to lookup. The `msg.target` will override the Target field if set. * **Timeout**: specify the timeout during lookup operation. The `msg.timeout` will override the Timeout field if set. * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set.

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information
`msg.element` If the element is found, it will be passed though the msg object to the next node.

Send Keys Sends the specified keys to the target

Inputs: * **Name**: a Name for this instance * **Page**: the page where this browser action executes. Pages can be defined by editing this configuration item * **Target**: specify the target to lookup. The `msg.target` will override the Target field if set. * **Timeout**: specify the timeout during lookup operation. The `msg.timeout` will override the Timeout field if set. * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set.

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information
`msg.element` If the element is found, it will be passed though the msg object to the next node.

Click On Clicks on the target

Inputs: * **Name**: a Name for this instance * **Page**: the page where this browser action executes. Pages can be defined by editing this configuration item * **Target**: specify the target to lookup. The `msg.target` will override the Target field if set. * **Timeout**: specify the timeout during lookup operation. The `msg.timeout` will override the Timeout field if set. * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set.

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information
`msg.element` If the element is found, it will be passed though the msg object to the next node.

Set Value Sets the value of the target

Inputs: * **Name**: a Name for this instance * **Value**: the value to set * **Page**: the page where this browser action executes. Pages can be defined by editing this configuration item * **Target**: specify the target to lookup. The `msg.target` will override the Target field if set. * **Timeout**: specify the timeout during lookup operation. The `msg.timeout` will override the Timeout field if set. * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set.

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information
`msg.element` If the element is found, it will be passed though the msg object to the next node.

To File Save a value to the File location with `msg.payload` content as a string.

Inputs: * **Name**: a Name for this instance * **File**: specify the file path to write to. The msg.filename will override the File field if set * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set.

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information

Identify Page Identifies the current page. If the page specified is available in the browser the `identified` output port is activated. If the page is not found the `not identified` output port is activated.

Inputs: * **Name**: a Name for this instance * **Page**: the page where this browser action executes. Pages can be defined by editing this configuration item * **Timeout**: specify the timeout during lookup operation. The `msg.timeout` will override the Timeout field if set. * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set.

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information

Get Value Gets the value of the target

Inputs: * **Name**: a Name for this instance * **Page**: the page where this browser action executes. Pages can be defined by editing this configuration item * **Target**: specify the target to lookup. The `msg.target` will override the Target field if set. * **Timeout**: specify the timeout during lookup operation. The `msg.timeout` will override the Timeout field if set. * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set. * **Store**: the location where the value is stored

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information
`msg.element` If the element is found, it will be passed though the msg object to the next node.

Get Attribute Gets the specified attribute value of the target

Inputs: * **Name**: a Name for this instance * **Page**: the page where this browser action executes. Pages can be defined by editing this configuration item * **Target**: specify the target to lookup. The `msg.target` will override the Target field if set. * **Timeout**: specify the timeout during lookup operation. The `msg.timeout` will override the Timeout field if set. * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set. * **Store**: the location where the value is stored

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information
`msg.element` If the element is found, it will be passed though the msg object to the next node.

Get Text Gets the text property of the target

Inputs: * **Name**: a Name for this instance * **Page**: the page where this browser action executes. Pages can be defined by editing this configuration item * **Target**: specify the target to lookup. The `msg.target` will override the Target field if set. * **Timeout**: specify the timeout during lookup operation. The `msg.timeout` will override the Timeout field if set. * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set. * **Store**: the location where the value is stored

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information
`msg.element` If the element is found, it will be passed though the msg object to the next node.

Run Script Executes the script in the context of the page

Inputs: * **Name**: a Name for this instance * **Page**: the page where this browser action executes. Pages can be defined by editing this configuration item * **Target**: specify the target to lookup. The `msg.target` will override the Target field if set. * **Timeout**: specify the timeout during lookup operation. The `msg.timeout` will override the Timeout field if set. * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set. * **Function**: the javascript code that is executed in the context of the page

Logging and Error Handling To log any information, or report an error, the following functions are available:

```
node.log("Log")
node.warn("Warning")
node.error("Error")
```

The Catch node can also be used to handle errors. To invoke a Catch node, pass msg as a second argument to node.error:

```
node.error("Error",msg)
```

Sending messages The function can either return the messages it wants to pass on to the next nodes in the flow, or can call node.send(messages).

It can return/send:

a single message object - passed to nodes connected to the first output an array of message objects - passed to nodes connected to the corresponding outputs If any element of the array is itself an array of messages, multiple messages are sent to the corresponding output.

If null is returned, either by itself or as an element of the array, no message is passed on.

See the online documentation for more help. You can manage your palette of nodes with ctrl-p

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information
`msg.element` If the element is found, it will be passed though the msg object to the next node.

Screenshot Take a screenshot fo the current page and save to a file

Inputs: * **Name**: a Name for this instance * **File**: full filename to store the screenshot * **Page**: the page where this browser action executes. Pages can be defined by editing this configuration item * **Target**: specify the target to lookup. The `msg.target` will override the Target field if set. * **Timeout**: specify the timeout during lookup operation. The `msg.timeout` will override the Timeout field if set. * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set.

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information

Nav To Navigate to the specified URL

Inputs: * **Name**: a Name for this instance * **URL**: the page where this browser action executes. Pages can be defined by editing this configuration item * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set.

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information

Nav Back Triggers the back action in browser

Inputs: * **Name**: a Name for this instance * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set.

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information

Nav Forward Triggers the forward action in browser

Inputs: * **Name**: a Name for this instance * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set.

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information

Nav Refresh Refreshes the page in the browser

Inputs: * **Name**: a Name for this instance * **Wait For**: specify the time to wait before looking up. The `msg.waitfor` will override the Wait For field if set.

Outputs: `msg.error` When the object is unexpected or cannot find element, this node generates the error with detail information