



Sheraton New Orleans

New Orleans, LA

February 13–15, 2017



Use the Nintex Platform as a microServices Platform

Vadim Tabakman
Matt Briggs

Function to microService

Use the Nintex Platform as a microServices Platform

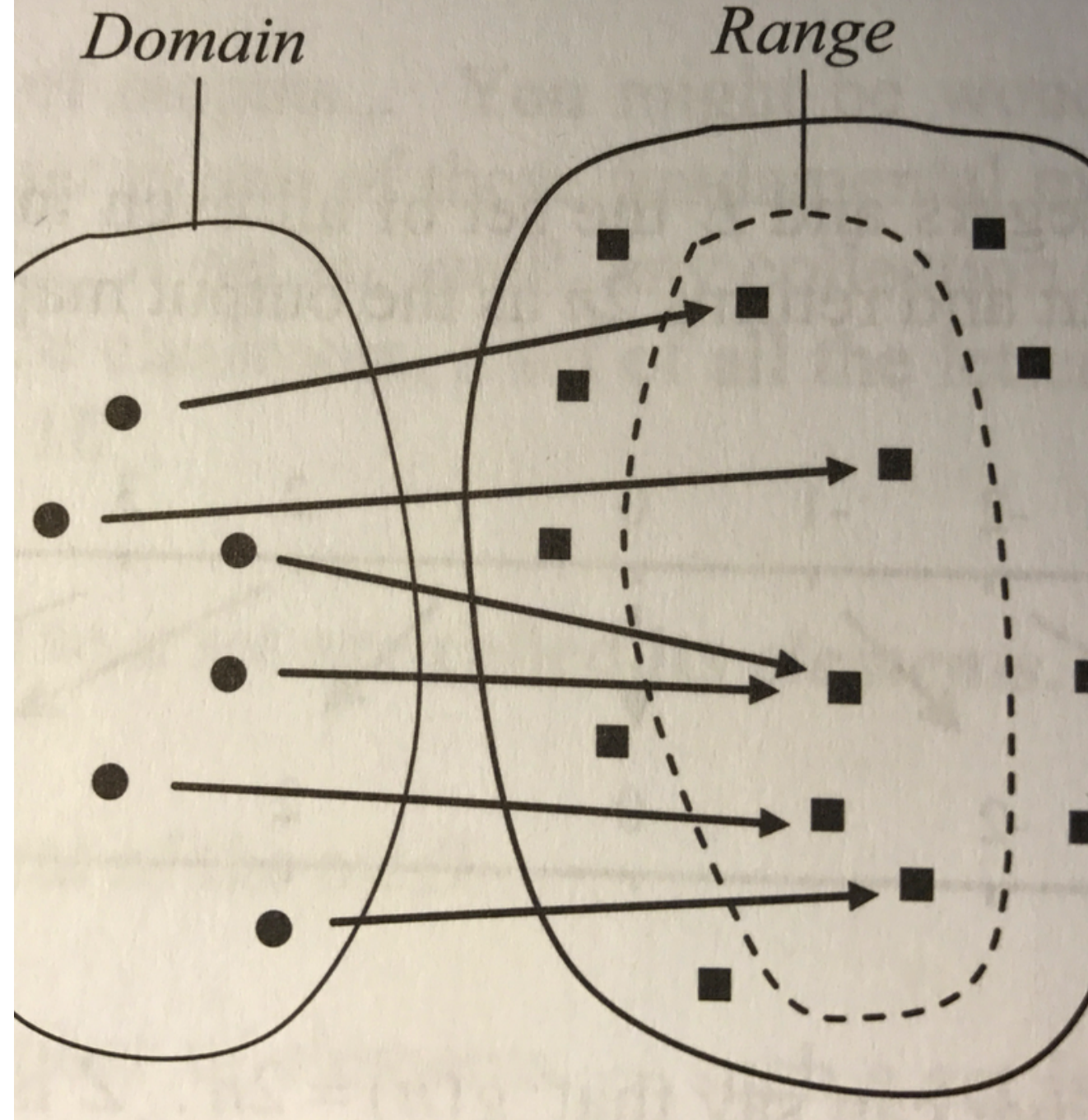
The Rise of the Lowly Console Application

If you take away three things from this talk:

- A method or function is a self-contained *thing* in the Nintex Platform.
- As a self-contained *thing* it is portable. It can be run inside of the Nintex Platform within a workflow, or run outside of the Nintex Platform and accessed by a workflow.
- Workflows supply supporting logic, context, and connectivity to your functions. Just as a function is a *thing*, a workflow can also take input and produce output via a RESTful endpoint.

Function

What is a Function?



A Function

- In math:

A function establishes a relation between a set of inputs (numbers, points, and objects) and a set of outputs. A function has a domain and a range.

- In code:

A function has input (parameters) consumed by an algorithm that produces a return.

Examples of Functions (methods) in Code

C#

```
//Converts a JSON string to XML.
public string XmlToJson(string inputXMLasString)
{
    try
    {
        var outputJsonAsString = "";
        var xmldoc = new XmlDocument();
        xmldoc.LoadXml(inputXMLasString);
        outputJsonAsString =
        JsonConvert.SerializeXmlNode(xmldoc);

        return outputJsonAsString;
    }
    catch (Exception ex)
    {
        var expectmessage = ex.ToString();
        var
        returnmessage = "JSON not valid. " +
        expectmessage;
        return returnmessage;
    }
}
```

JavaScript

```
/**
 * This is a function that "Return Cosine Value - cosine(x)
 * @param {int} xValue The cosine input value.
 * @return {int} cosine The cosine
 */
c$.cosine = function (xValue) {
    var c = Math.cos(xValue);
    return c;
};
```

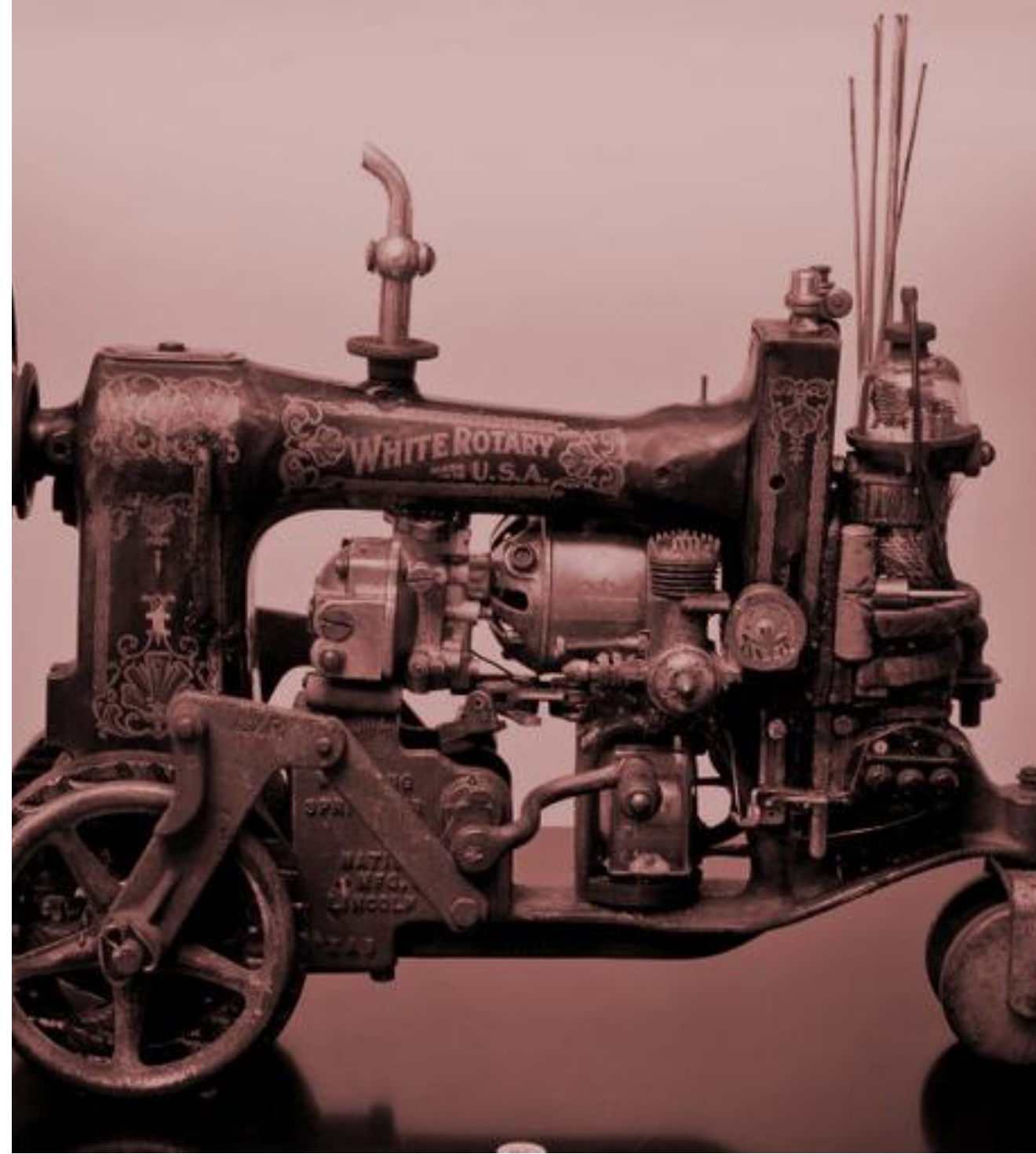

A Function is Portable

Portability includes several aspects:

- You will need a runtime that will run your function
- You will need an interface where external applications can pass input into your function and get the return from your function

A function also:

- Focuses, or simplifies, your development problem
- Is stateless
- Allows higher-level logic to be handled via Nintex Workflow



Portable Functions

Runtime

- Within the context of the Nintex Platform, you can create managed code solutions and add your function as a dynamic link library (DLL) to the SharePoint environment.
- External to Nintex you can host your function as a *serverless* function within an API on a cloud platform such as Amazon Web Services (AWS), Google Cloud, and Windows Azure.

Interface

- REST: The one interface to rule them all. Where you place your function, it can be reached via a RESTful interface.
- The Nintex Platform offers a variety of REST endpoints as events and workflow actions.
- Cloud services are built to offer accessible endpoints via REST.

Why Place your Function Inside of the Nintex Platform?

- **Function in the SharePoint Context.**

- Access to SharePoint runtime
- Access to SharePoint entities
- Part of the SharePoint environment
- Integrates with your existing SharePoint processes

- **Function in the Nintex Workflow Context.**

- Available to designer-level users
 - Inline functions: accessible via Insert Reference
 - Custom Actions: accessible via Workflow Designer Toolbox
 - User-Defined Actions: encapsulates your function in a workflow with supporting logic.
- Function supports separation of concerns
 - Function handles specific processing
 - Workflow handles higher-level logic

Why Place Your Function In the Cloud?

- **Function as a Web API or serverless function**

- Abstracts your runtime (independent of a SharePoint context)
- Can be accessed by any Nintex Product via REST using a Web Services action
- Supports Do Not Repeat Yourself (DRY)

- **Function in Nintex Workflow Cloud**

- Function focuses on lower-level process
- NWC focuses on higher-level process logic
 - Context handling
 - Validation
 - Error handling
 - Monitoring (Nintex Hawkeye, for example)
- Supports separation of concerns
- Supports encapsulation

Adding a REST
endpoint to
your function
makes your
function a
microService.



So, I have developed a method in a class (a function in an object) and I am thinking, "Where do I put this?"

We will look at where to put your function in the Nintex Platform:

inline function

custom
workflow action

user-defined
action

a workflow
with external
start

in the cloud

In Nintex
Workflow
Cloud

The Nintex Platform enables apps to talk to your function:

External
Start in Nintex
Workflow

Rest Action in
Nintex
Workflow

Integrating
serverless
functions

Nintex
Workflow
Cloud

when you have multiple functions

You have an

an ecosystem of
functions

or a microServices
platform

Inline Function

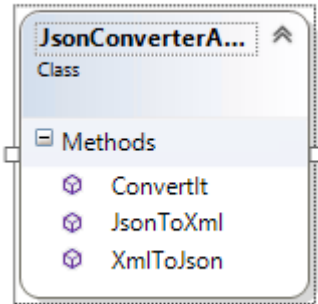
An Example Function

JsonConverterAction: ConvertIt

```
2 references
internal class JsonConverterAction
{
    1 reference
    public string ConvertIt(string inputText)[...]

    //Converts a JSON string to XML.
    1 reference
    public string XmlToJson(string inputXMLasString)[...]

    //Converts an XML string to JSON.
    3 references
    public string JsonToXml(string inputJSONasString)[...]
}
```



JsonConverterAction

Takes a string and checks it is JSON or XML and then converts JSON to XML or XML to JSON.

- Input: String (JSON/XML)
- Output: String (XML/JSON)

```
1 reference
16 public string ConvertIt(string inputText)
17 {
18     var convertedtext = inputText.Trim();
19     var firstChar = inputText[0].ToString();
20     switch (firstChar)
21     {
22         case "1":
23             convertedtext = XmlToJson(convertedtext);
24             break;
25         case "2":
26             convertedtext = JsonToXml(convertedtext);
27             break;
28         case "{":
29             convertedtext = JsonToXml(convertedtext);
30             break;
31         case "[":
32             convertedtext = JsonToXml(convertedtext);
33             break;
34         default:
35             convertedtext = JsonToXml(convertedtext);
36             break;
37     }
38     return convertedtext;
39 }
```


JsonConverterAction: XMLtoJSON

```
2 references
internal class JsonConverterAction
{
    1 reference
    public string ConvertIt(string inputText)[...]

    //Converts a JSON string to XML.
    1 reference
    public string XmlToJson(string inputXMLasString)[...]

    //Converts an XML string to JSON.
    3 references
    public string JsonToXml(string inputJSONasString)[...]
}
```

```
//Converts a JSON string to XML.
1 reference
public string XmlToJson(string inputXMLasString)
{
    try
    {
        string outputJsonAsString = "";
        XmlDocument xmldoc = new XmlDocument();
        xmldoc.LoadXml(inputXMLasString);
        outputJsonAsString = JsonConvert.SerializeXmlNode(xmldoc);

        return outputJsonAsString;
    }
    catch (Exception ex)
    {
        string expectmessage = ex.ToString();
        string returnmessage = "XML not valid. " + expectmessage;
        return returnmessage;
    }
}
```

JsonConverterAction: JsontoXML

```
2 references
internal class JsonConverterAction
{
    1 reference
    public string ConvertIt(string inputText)[...]

    //Converts a JSON string to XML.
    1 reference
    public string XmlToJson(string inputXMLasString)[...]

    //Converts an XML string to JSON.
    3 references
    public string JsonToXml(string inputJSONasString)[...]
}
```

```
//Converts an XML string to JSON.
3 references
public string JsonToXml(string inputJSONasString)
{
    try
    {
        string outputXmlAsString = "";
        XmlNode node = JsonConvert.DeserializeXmlNode(inputJSONasString, "Root");
        StringWriter stringWriter = new StringWriter();
        XmlTextWriter xmlTextWriter = new XmlTextWriter(stringWriter);
        node.WriteTo(xmlTextWriter);
        outputXmlAsString = stringWriter.ToString();
        return outputXmlAsString;
    }
    catch (Exception ex)
    {
        string expectmessage = ex.ToString();
        string returnmessage = "JSON not valid. " + expectmessage;
        return returnmessage;
    }
}
```

Inline Function

In Nintex Workflow 2013, an inline function, also known as a string function, is a .NET Framework static method that can be used like any other workflow reference in a workflow action.

To create the inline function:

- Create an empty SharePoint 2013 project.

- Write your function as in a method in a class that contains your library inline functions.

- Deploy the WSP to SharePoint.

- Register the Inline Function with Nintex Workflow using the NWAdmin tool.

Register your Function

```
NWAdmin.exe -o AddInlineFunction -functionalias  
fn-Jsonconvert -assembly  
"Custom.InlineFunctions.Custom,  
Version=1.0.0.0, Culture=neutral,  
PublicKeyToken=1b923b394396c09b" -namespace  
Custom.InlineFunctions.Custom -typename Custom  
-method Jsonconvert -usage "fn-  
Jsonconvert(string)" -description "Returns JSON  
as XML or XML as JSON. Note, cannot convert a  
list of JSON objects as XML."
```

Insert Reference

Common	Item Properties	Inline Functions
IsCurrentUser		
IsDate		
IsMemberOfGroup		
IsNullOrEmpty		
IsNumeric		
Jsonconvert		
Length		
LessThan		
LessThanOrEqual		
Max		
Min		
NewGuid		
Not		

Returns JSON as XML or XML as JSON. Note, cannot convert a list of JSON objects as XML.
fn-Jsonconvert(string)

Dynamic text to insert

OK



Create an Inline Function

Matt Briggs

Custom Workflow Action

Custom Workflow Action

You can create custom workflow activities that support functions not found in the packaged set of actions.

To create a custom workflow action:

- Build the Custom Action project in Visual Studio

- Parts of the project

- Build the empty project

- Code the action

- Code the activity class

- Code the adapter class

- Code the configuration page

- Add the graphics to the layouts folder

- Add the event listener feature

- Add and update the NWA file

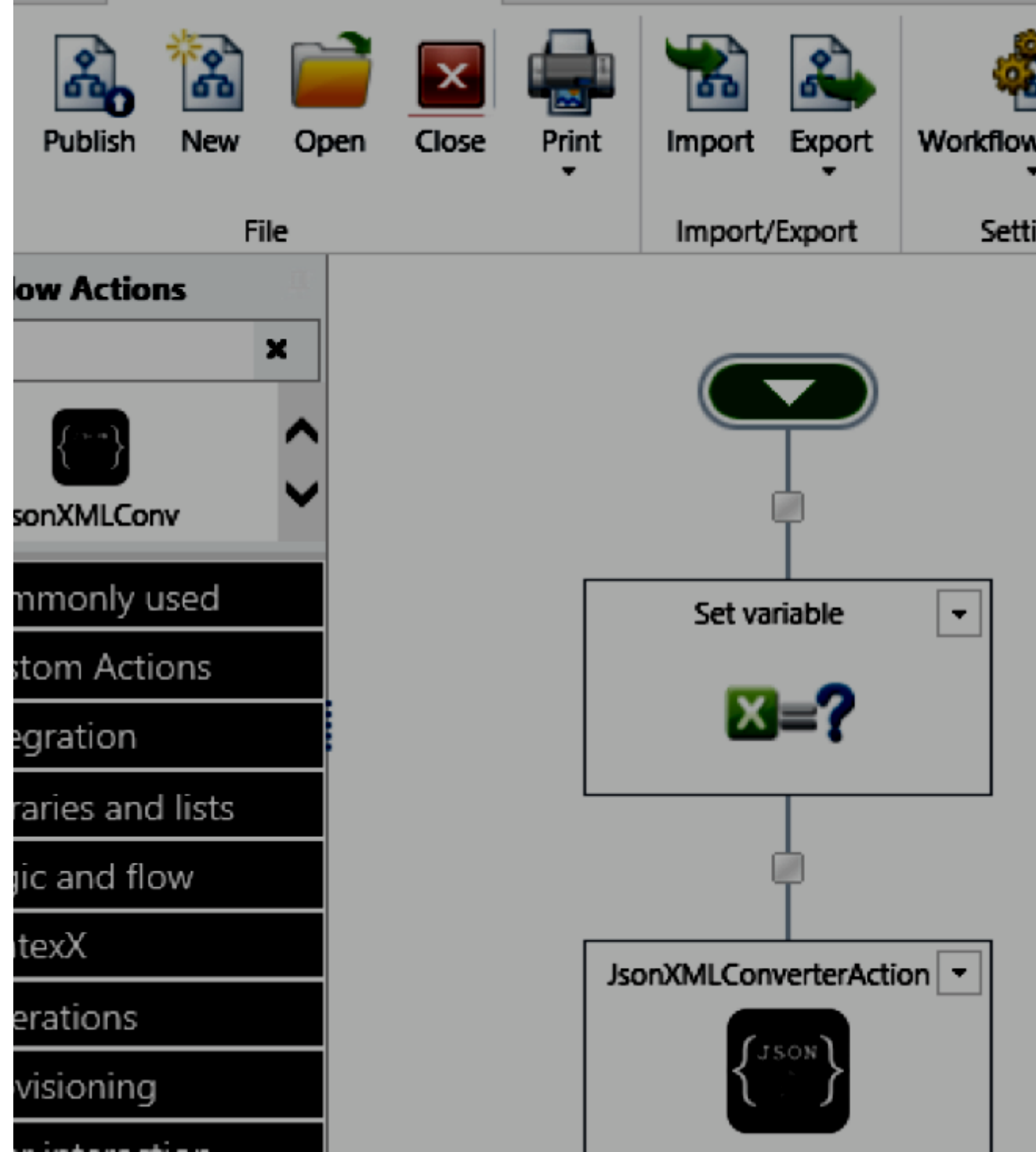
- Build, deploy, and import to Nintex Workflow Management

- Verify the action works

Parts of a Custom Action:

Portability includes several aspects.

- Event Receiver
- Package
- Configuration Page (aspx)
- NWA file
- ActionAdapater.cs
- ActionActivity.cs





Create a Custom Action

Matt Briggs

User Defined Action

User Defined Action

User Defined Actions (UDA) provides the ability encapsulate a workflow as a reusable element. You can define Input and Output parameters.

To create a UDA:

Navigate to the Manage User Defined Actions page.

Click on Create. Design the UDA just as you would a Nintex Workflow.

Configure the User Defined Action Settings.

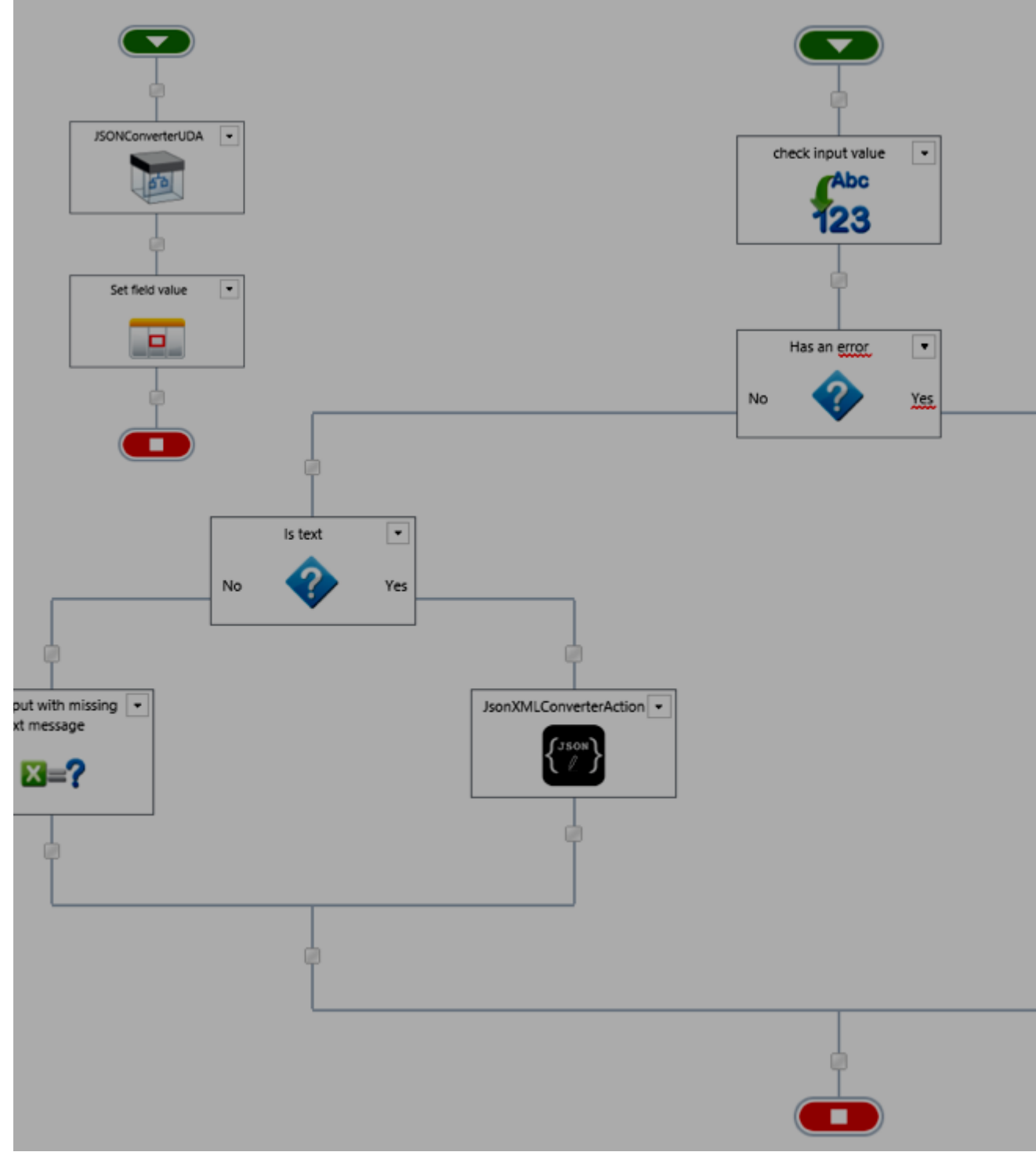
Save and Publish the UDA.

Note: You can also import and export the UDA.

Wraps your function

Roll your function in higher-level logic:

- Add actions that provide support for:
 - Context handling
 - Validation
 - Error handling
 - Monitoring (Nintex Hawkeye, for example)
- Supports separation of concerns
- Supports encapsulation





Create a UDA

Vadim Tabakman

External Start
(as a RESTful service)

External Start

External Start allows you to initiate site workflows by contacting a RESTful endpoint. Each endpoint URL stores the workflow information along with its start variables of supported data types (if any). Endpoint URLs reference `run.nintex.io`.

Example endpoint URL: `https://run.nintex.io/x-start/weuyj1TkGD`

To create the inline function:

- Configure Nintex Workflow to support External Start
- Generate an External Start endpoint URL
- Retrieve information about workflow variables
- Pass a security key and retrieve the HMAC code
- Start the workflow and receive the correlation ID

Available via REST

Your workflow can be called from any REST client. A client can pass a parameter and receive the return in a call back.

- In your workflow you can provide support for:
 - Context handling
 - Validation
 - Error handling
 - Monitoring (such as Nintex Hawkeye)
- Supports separation of concerns
- Supports encapsulation

Workflow

in the URLs when used outside the Nintex platform

and security key

Calls

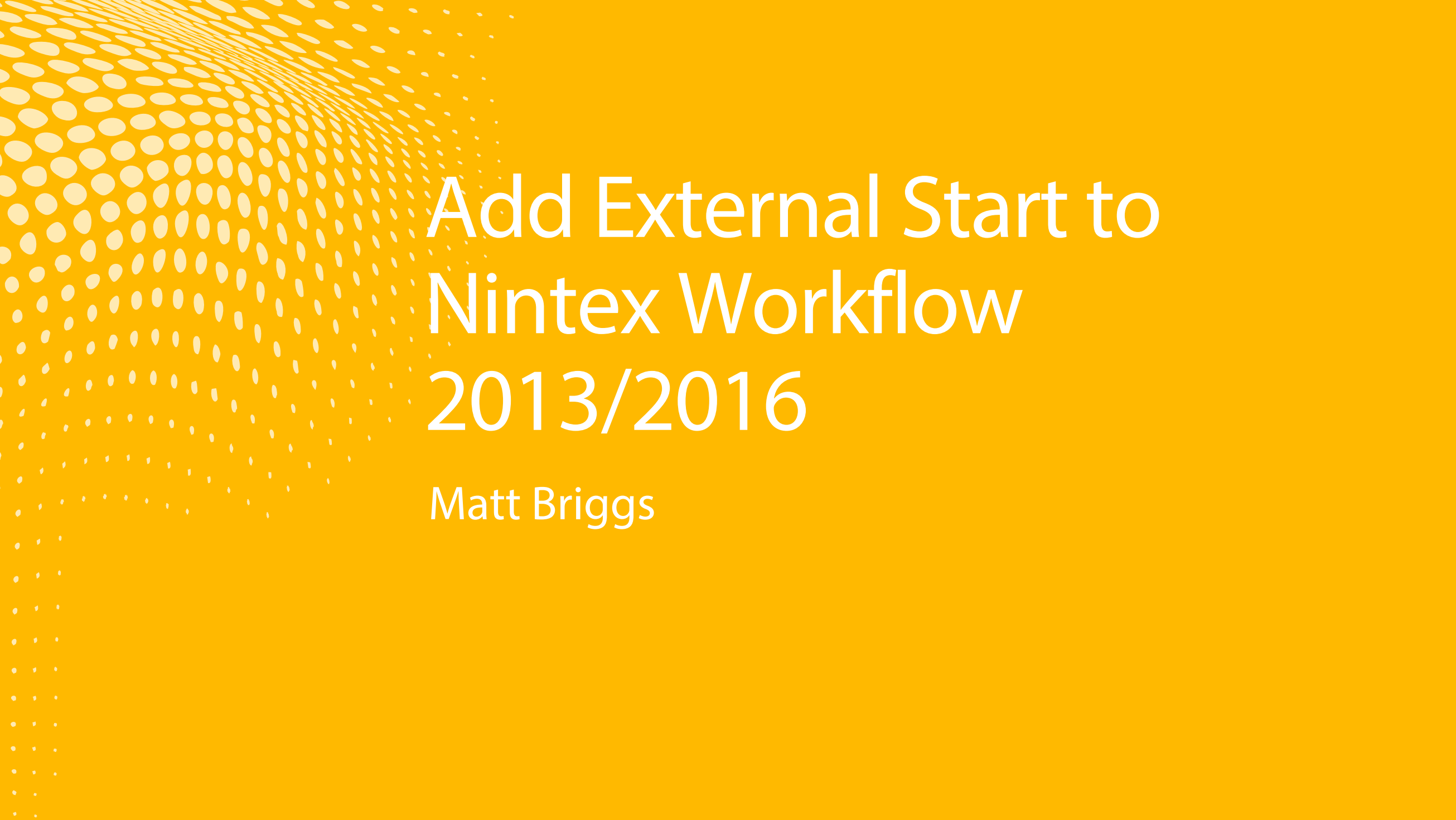
[/view details](#)

2

La

1

A



Add External Start to Nintex Workflow 2013/2016

Matt Briggs

Serverless Function

Serverless Function (Windows Azure)

You can rapidly deploy your function to Azure which will provide the runtime and interface (REST). Your function will be available via a HTTP call. In addition, you can use nearly any code to accomplish this task.

To create a serverless function:

- Create the project in Windows Azure.

- Upload your function.

- Wire the HttpResponseMessage in Run.CSX to your function.

- Retrieve the HTTP trigger (a Restful call).

Available via REST

Your function can be called from any REST action in Nintex Workflow:

- Your function can extend workflows in Nintex 2013/2015, Nintex 0365, and NWC
- Update your code in one location.
- Conceal any complexity behind the REST interface.
- Note: Your function must be stateless.

Function app
NintexUATextFunction1A

Search my functions

+ New Function

JsonXML

</> Develop

Integrate

Manage

Monitor

Code (JsonConverterAction.cs)

Save Run

Function Url: <https://nintexuatextfunction1a.azurewebsites.net/api/JsonXML?code=rQq9EkeKautjDL>

```
1 using System;
2 using System.IO;
3 using System.Xml;
4 using System.Xml.Linq;
5 using Newtonsoft.Json;
6
7 /// <summary>
8 /// Converts text form JSON to XML or from XML to JSON. The converter detect
9 /// <parameter>Input string (in either JSON or XML format.</parameter>
10 /// </summary>
11
12 namespace JsonConvertAct
13 {
14     internal class JsonConverterAction
15     {
16         public string ConvertIt(string inputText)
17         {
18             var convertedtext = inputText.Trim();
19             var firstChar = inputText[0].ToString();
20             switch (firstChar)
21             {
22                 case "1":
23                 case "<":
24                     convertedtext = XmlToJson(convertedtext);
25                     break;
26                 case "2":
27                 case "{":
28                     convertedtext = JsonToXml(convertedtext);
29                     break;
30                 case "3":
31                 case "[":
32                     convertedtext = JsonToXml(convertedtext);
33                     break;
34                 default:
35                     convertedtext = JsonToXml(convertedtext);
36                     break;
37             }
38         }
39     }
40 }
```

Logs

```
2016-12-11T02:51:58.455 Exception while executing function: Functions.JsonXML. System.Net.Http.
2016-12-11T02:52:05.324 Function started (Id=12446bf8-324c-44d6-a3c6-d06cc5b8e0b0)
2016-12-11T02:52:05.324 C# HTTP trigger function processed a request.
2016-12-11T02:52:05.324 Function completed (Failure, Id=12446bf8-324c-44d6-a3c6-d06cc5b8e0b0)
2016-12-11T02:52:05.356 Exception while executing function: Functions.JsonXML. System.Net.Http.
2016-12-11T02:52:25.021 Function started (Id=f9da9bff-bb19-40ae-bdd3-f17352867adb)
2016-12-11T02:52:25.021 C# HTTP trigger function processed a request.
2016-12-11T02:52:25.021 Function completed (Success, Id=f9da9bff-bb19-40ae-bdd3-f17352867adb)
2016-12-11T02:52:37.326 Function started (Id=5efdc534-a145-457b-9a81-20a54da0e42d)
2016-12-11T02:52:37.326 C# HTTP trigger function processed a request.
2016-12-11T02:52:37.326 Function completed (Success, Id=5efdc534-a145-457b-9a81-20a54da0e42d)
```




Add a Function to Azure Cloud

Matt Briggs

Nintex Workflow Cloud (as a RESTful service)

External Start Event

You can configure and send an HTTP Post request to start a workflow that is configured for External Start; you can then return your information via a RESTful call.

To create an external start event:

- Define your workflow in Nintex Workflow Cloud.

- Click the Start Event, and Select Connector and select Nintex. Select External Start.

- Configure your start event and add any input variables to be handled by the workflow.

- Save your workflow.

- Copy the External Start Event values for your REST client.

Call your workflow

```
POST /api/v1/workflow/published/601107ff-10eb-4340-bb44-86a93f984993/instances?token=eyJhbGciOiJIUzI1bnQoInRlc2UiOiJlZmVudDkIjoInjAxMTA3ZmYtMTBlYi00MzQwLWJiNDQtODZhOTNmOTg0OTkzIiwidGVuYW50SWQiOiJlOTc5NjgzYiIsImJAwLTRjODAtODlmOS03NjEwOWRlNzJkYzgiLCJpYXQiOiJE0ODE0MDE5OTV9.KyjfTMmYGM545pLG1DtePSlT-0NMjv_ZhS8L0UGxOVs
```

HTTP/1.1

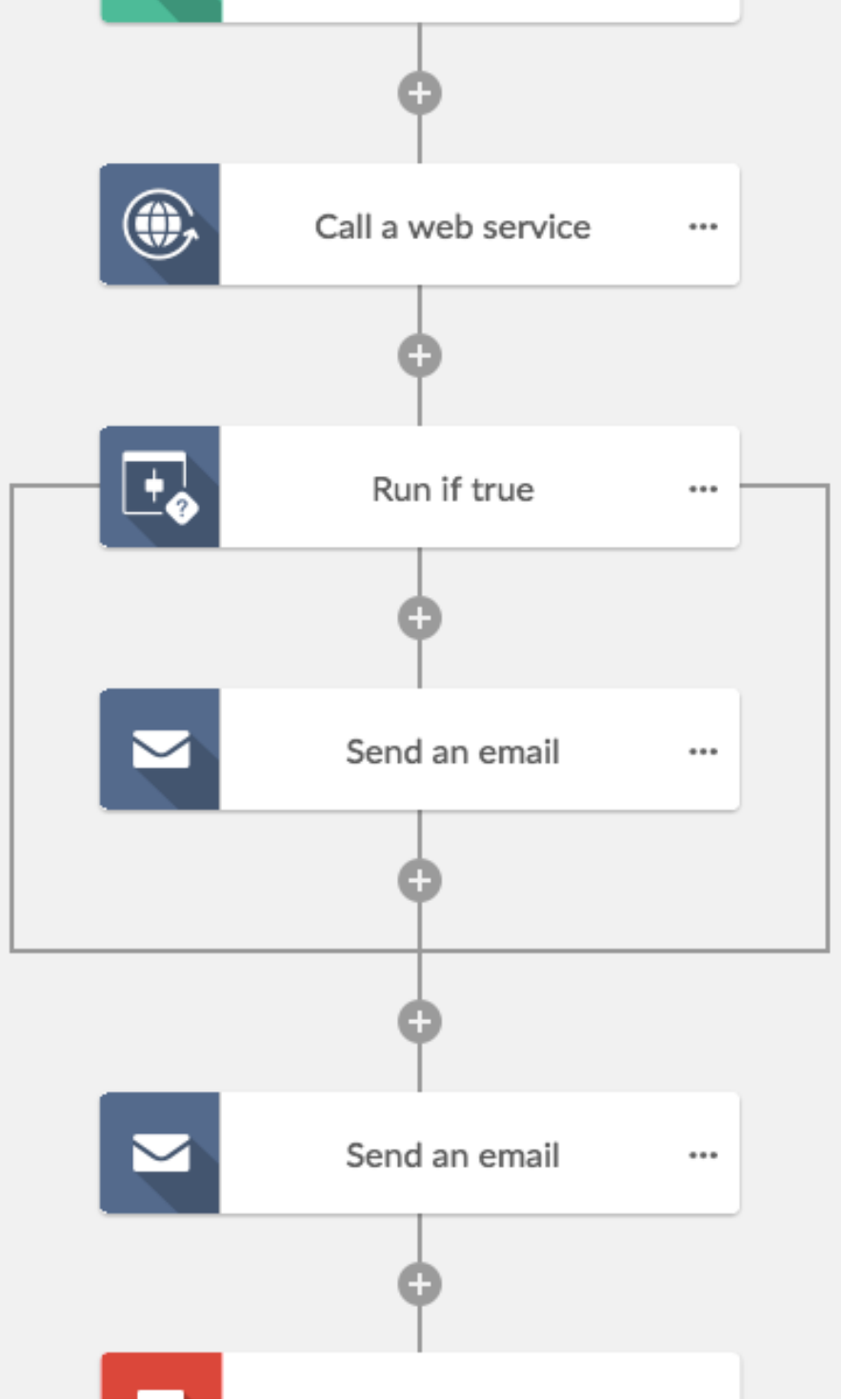
Host: yourtenant.workflowcloud.com

Content-Type: application/json

Cache-Control: no-cache

Postman-Token: e7474ba4-5555-f963-9d65-2d2dce29fb6c

```
{
  "startData": {
    "se_word1": "rabbit"
  },
  "options": {
    "callbackUrl": "<optionally add a callback URL
here. Must be HTTPS>"
  }
}
```





External Start with NWC

Vadim Tabakman

Ecosystem of Functions

Next Steps...

From a console-style app focused on a function, to the function placed within a RESTfully-accessible Nintex workflow handling:

- state
- validation
- monitoring
- logging
- errors

You can rapidly develop robust solutions, handling solutions in agile iterations.

They Can Grow Wildly Tangled

Handling a bottom-up architecture:

- Register and track your inventory of functions in a central tool
- Map functional dependencies as part of your deployment workflow
- Instrument your functions at the workflow level for performance and usage monitoring
- Use a tool such Swagger.IO to manage your API endpoints.
- Review the organic growth and state of your architecture at regular intervals
- Be mindful of your contracts: a RESTful endpoint is an interface and a contract





Questions and Discussion

NINTEX

inspire



Vadim Tabakman

vadim.tabakman@nintex.com

[@vadim_tabakman](#)

[linkedin.com/in/vadimtabakman](https://www.linkedin.com/in/vadimtabakman)

Matt Briggs

matt.briggs@nintex.com

[@seedcake](#)

[linkedin.com/in/mattbriggs](https://www.linkedin.com/in/mattbriggs)