Introduction to Platform SDK

alteryx

inspire

Forward Looking Statements

This presentation includes “forward-looking statements” within the meaning of the Private Securities Litigation Reform Act of 1995. These forward-looking statements may be identified by the use of terminology such as “believe,” “may,” “will,” “intend,” “expect,” “plan,” “anticipate,” “estimate,” “potential,” or “continue,” or other comparable terminology. All statements other than statements of historical fact could be deemed forward-looking, including any projections of product availability, growth and financial metrics and any statements regarding product roadmaps, strategies, plans or use cases. Although Alteryx believes that the expectations reflected in any of these forward-looking statements are reasonable, these expectations or any of the forward-looking statements could prove to be incorrect, and actual results or outcomes could differ materially from those projected or assumed in the forward-looking statements, including, but not limited to, as a result of: the impact to the economy, our customers and our business due to the COVID-19 pandemic; our ability to manage our growth and the investments made to grow our business effectively; our ability to retain and expand our talent base, particularly our sales force and software engineers, and increase their productivity; our history of losses; our dependence on our software platform for substantially all of our revenue; our ability to attract new customers and expand sales to and retain existing customers; our ability to develop and release product and service enhancements and new products and services to respond to rapid technological change in a timely and cost-effective manner; intense and increasing competition in our market; the rate of growth in the market for analytics products and services; our ability to establish and maintain successful relationships with our channel partners; our dependence on technology and data licensed to us by third parties; risks associated with our international operations; our ability to develop, maintain, and enhance our brand and reputation cost effectively; litigation and related costs; security breaches; and other general market, political, economic and business conditions. Additionally, these forward-looking statements involve risk, uncertainties and assumptions, including those related to the impact of COVID-19 on our business and global economic conditions. Many of these assumptions relate to matters that are beyond our control and changing rapidly, including, but not limited to, the timeframes for and severity of the impact of COVID-19 on our customers’ purchasing decisions and the length of our sales cycles, particularly for customers in certain industries highly affected by COVID-19. Alteryx’s future financial condition and results of operations, as well as any forward-looking statements, are subject to risks and uncertainties, including but not limited to the factors set forth above, in Alteryx’s press releases, public statements and/or filings with the Securities and Exchange Commission, especially the “Risk Factors” section of Alteryx’s most recent Annual Report on Form 10-K. These documents and others containing important disclosures are available at www.sec.gov or in the “Investors” section of Alteryx’s website at www.alteryx.com. All forward-looking statements are made as of the date of this presentation and Alteryx assumes no obligation to update any such forward-looking statements.

This presentation also contains estimates and other statistical data made by independent parties and by us relating to market size and growth and other data about our industry. This data involves a number of assumptions and limitations, and you are cautioned not to give undue weight to such estimates. In addition, projections, assumptions, and estimates of our future performance and the future performance of the markets in which we operate are necessarily subject to a high degree of uncertainty and risk. In addition to the financials presented in accordance with U.S. generally accepted accounting principles (GAAP), this presentation includes certain non-GAAP financial measures. The non-GAAP financial measures have limitations as analytical tools and you should not consider them in isolation or as a substitute for the most directly comparable financial measures prepared in accordance with GAAP. There are a number of limitations related to the use of these non-GAAP financial measures versus their nearest GAAP equivalents. Other companies, including companies in our industry, may calculate non-GAAP financial measures differently or may use other measures to evaluate their performance, all of which could reduce the usefulness of our non-GAAP financial measures as tools for comparison. We urge you to review the reconciliation of our non-GAAP financial measures to the most directly comparable GAAP financial measures set forth in the Appendix, and not to rely on any single financial measure to evaluate our business.

Any unreleased services or features referenced in this or other presentations, press releases or public statements are only intended to outline Alteryx’s general product direction. They are intended for information purposes only and may not be incorporated into any contract. This is not a commitment to deliver any material, code, or functionality (which may not be released on time or at all) and customers should not rely upon this presentation or any such statements to make purchasing decisions. The development, release, and timing of any features or functionality described for Alteryx’s products remains at the sole discretion of Alteryx.

Agenda

01 Session Overview

02 Basics of Alteryx
Developer Experience

03 Walkthrough #1
API Connections and Data Manipulation

04 Tool Development:
Step-by-step Recap

05 Walkthrough #2
Data Processing at Scale

06 Closing Thoughts and Office Hours

Session Overview

Training Speakers

David Ung

Senior Tech. Product Manager

david.ung@alteryx.com

Linkedin.com/in/davidu1/

Bryan Ashby

Lead Software Engineer

bryan.ashby@alteryx.com

Linkedin.com/in/bryan-ashby/

Paul England

Lead Software Engineer

paul.england@alteryx.com

Linkedin.com/in/paul-bradley-england/

Special Attendance

Cassandra Clark

Technical Lead (Ace Emeritus)

cassandra.clark@alteryx.com

Linkedin.com/in/cassandra-clark-/

Peter Ott

Manager of Product Management

peter.ott@alteryx.com

Linkedin.com/in/peter-ott/

Orion Ou

Senior Software Engineer

orion.ou@alteryx.com

Linkedin.com/in/orionou/

Extensibility at Alteryx

Who are we?

The Developer Experience and Extensibility product group **provides products and experiences for both Cloud and On-premise** users across numerous applications under the Alteryx portfolio.

What do we own?

We build SDKs and Client Libraries that allow technical users to **customize and extend Alteryx**.

Additionally, we provide infrastructure and services that support a smooth Developer Experience.

Learning Outcomes

01 **Understand the fundamentals of Alteryx Extensibility.**

- Scaffold, package, and deploy custom tools
- Custom tool development lifecycle
- Core functionality of the Alteryx CLI and SDK

02 **Ability to develop custom tools on top of Alteryx products. (Hands-on practice)**

- Make API data connections and integrations
- Perform customized data transformations
- Perform customized data analysis and data science
- Process large data at scale
- Integrate with 3rd-party libraries.

Required Materials for Training

- Alteryx Designer 21.4+
  - Recommended: Latest Preferred
- MiniConda (Install MiniConda)
- Git (Install Instructions)
- Python compatible IDE or Editor
  - Recommended: VSCode, PyCharm

Note: Python and Node are required but will be installed via MiniConda.

Walkthrough Material:

ayx-developer-sdk/inspire

- Code
- Examples
- Guides
- Test data
- Training walkthrough

Alteryx SDK and CLI Installation

Once you've installed all the required materials, perform the following steps via Terminal/PowerShell:

1. The recommended way to install Python or Node on Windows is through Conda.
    1. Run this command: `conda create -n MyEnvName python=3.8.5 doit`
    2. For Python and Node: `conda create -n MyEnvName python=3.8.5 nodejs=14 doit`
2. Activate the Conda environment using: `conda activate MyEnvName`
    1. Check the python version with `python --version`
    2. Ensure that the environment is fresh by running `pip list`
3. Install the CLI and SDK: `pip install ayx-plugin-cli ayx-python-sdk`

Basics of Alteryx Developer Experience

inspire

General Terminology

Software Development Kit (SDK)

Bundled functions, modules, APIs, and CLIs that enable the active ability to build technology or interactions with specific systems and services.

SDKs often come with a suite of developer necessities such as testing suites, documentation, examples, guides, and other enablement to support a smooth development experience.

At Alteryx, SDKs allow individuals to **build, scaffold, package, and deploy custom extensions/tools on top of Alteryx Products.**

Command Line Interface (CLI)

A command line interface provides a series of operations and guides you through a specific development process or system. CLIs are often bundled companions within a SDK product.

At Alteryx, our CLI allows for a guided experience from scaffolding to deployment for an Alteryx plugin.

Alteryx SDKs allow Developers and Partners to **extend the power of Alteryx** with tools that Designer does not support out of the box and thus opening the gate for **infinite possibilities and solutions**.

Common Use Cases for Custom Tools

1

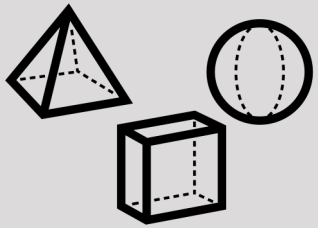
New Data Connections

Build new connectors to reach otherwise inaccessible data

2

Advanced Transforms

Augment workflows with custom Python functions, ML models, and more



3

Workflow Simplification

Reduce the burden of complex workflows and macros with a single custom tool

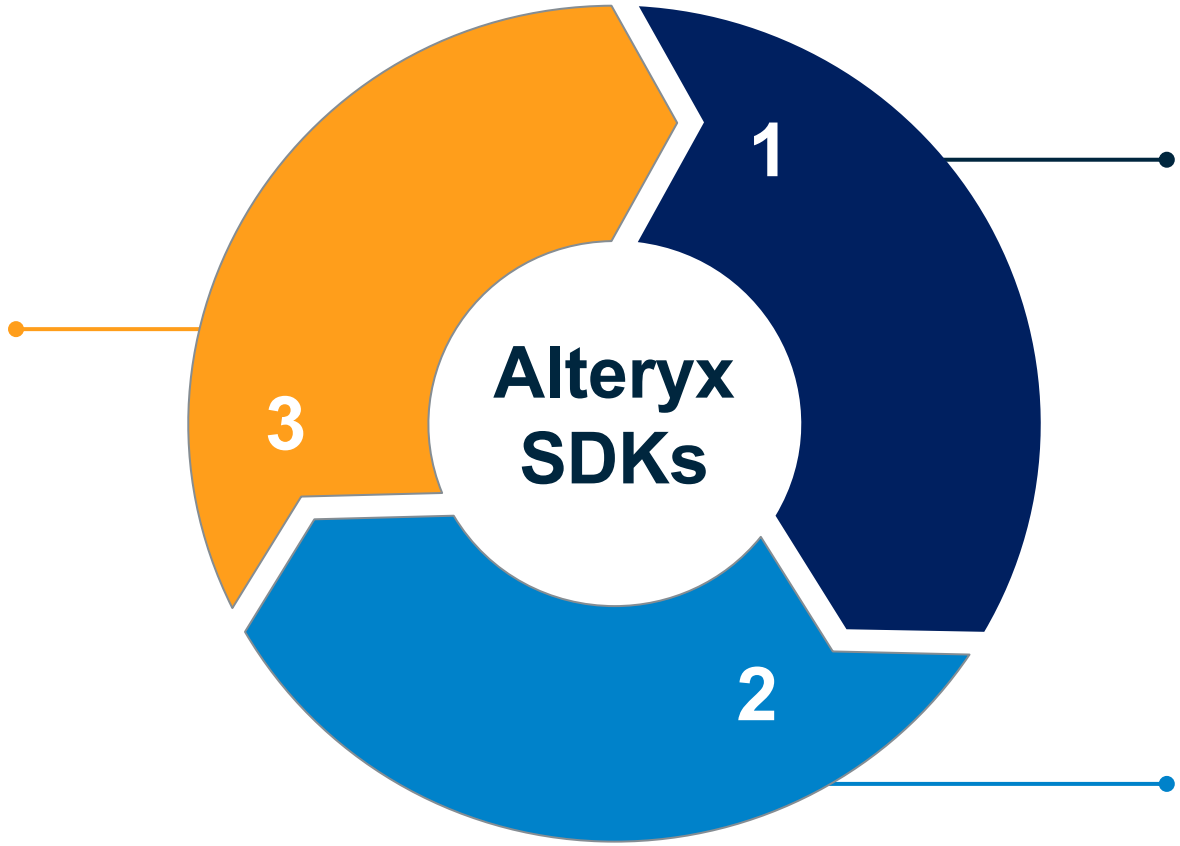
4

Proprietary Integrations

Integrate Alteryx with proprietary code and executables

Our process is easy!

The typical development lifecycle of an Alteryx extension/plugin Developer:



Scaffold and Code

Use the Alteryx Platform SDK and CLI to scaffold your project and begin coding!!!

Add your business logic!

Testing Client

Use the Alteryx Testing Client to scaffold and perform unit and client-side testing.

Test your plugin backend without having to launch Alteryx Designer.

Package and Deploy

Use the Alteryx Platform SDK and CLI to package your tool after testing and install into Alteryx Designer.

Common Development Steps

01

Create Workspace

Users need to create a workspace to house all of their tool code.

sdk-workspace-init: This command initializes a workspace that contains all of the source code and configuration details for a set of Alteryx SDK plugins.

02

Scaffold Tool

Once your workspace is ready, you will need to define the tool that will be scaffolded.

create-ayx-plugin: This command generates the boilerplate code, configs, and UI for a new Alteryx tool in the workspace. (Tool Name, Tool Type, Description, Tool Version)

03

Add Business Logic

Once the base tool and configs are scaffolded into your workspace, you can now begin coding your business logic.

We scaffolded four core functions for you to add your business logic:

- __Init__()
- on_incoming_connecti on_complete()
- on_record_batch()
- on_complete()

04

Generate Plugin YXI

When your tool is ready for usage, you will need to package your tool into a YXI.

YXI is an archive containing everything needed to install a new tool.

create-yxi: This command packages your workspace (plugin, configs, code, metadata) and saves it as a YXI format.

05

Install Tool to Designer

The final step is to install into Designer.

You can use these two commands to install tools directly into Alteryx Designer.

install-yxi installs ANY arbitrary YXI given a path.

designer-install creates and installs the YXI from the current workspace.

alteryx

inspire

Core Concepts: Where do I add my code?

PluginV2 Class

The Plugin class is the basis. The Plugin class provides the **required abstract operations that need to be implemented so that a tool can interact with Alteryx Designer.** These interactions are mediated by the "**Providers**", which provide simplified interfaces for Designer functionality and drive the execution of the Alteryx Plugin Tools.

Four Core Functions:

__Init__()

on_incoming_connection_complete()

on_record_batch()

on_complete()

Core Concepts: Auto-generated Basic Skeleton

Created during the `create-ayx-plugin` command step.

After this command finishes, you will see a file named `<YOUR_TOOL>.py` under ~/backend/ayx_plugins/ with the boilerplate code.

When you open the file, you should see something like this.

```python
class APITool(PluginV2):
    """Concrete implementation of an AyxPlugin."""

    def __init__(self, provider: AMPProviderV2) -> None:
        self.provider = provider
        # truncated code

    def on_incoming_connection_complete(self, anchor: namedtuple) -> None:
        # truncated code

    def on_record_batch(self, batch: "Table", anchor: namedtuple) -> None:
        # truncated code

    def on_complete(self) -> None:
        import pandas as pd
        import pyarrow as pa

        df = pd.DataFrame(
            {
                "x": [1, 2, 3],
                "y": ["hello", "world", "from ayx_python_sdk!"],
                "z": [self.config_value, self.config_value, self.config_value],
            }
        )

        packet = pa.Table.from_pandas(df)

        self.provider.write_to_anchor("Output", packet)
        self.provider.io.info("APITool tool done.")
```

alteryx

Walkthrough #1

API Connections and Data Manipulation

Training Material:
https://github.com/alteryx/ayx-developer-sdk/tree/main/inspire

Tool Development: Step-by-step Recap

alteryx

inspire

Step 0: Help and Versioning

How do I find command references?

```
ayx_plugin_cli --help
```

How do I find arguments/default-values?

```
ayx_plugin_cli <command> --help
```

How do I check the CLI version?

```
ayx_plugin_cli version
```

```
(MyEnvName) ~$ ayx_plugin_cli version
Alteryx CLI Version: 1.0.4
(MyEnvName) ~$
```

```
(MyEnvName) ~$ ayx_plugin_cli --help
Usage: ayx_plugin_cli [OPTIONS] COMMAND [ARGS]...

  The Alteryx CLI for SDK Development.

Options:
  --install-completion [bash|zsh|fish|powershell|pwsh]
                                  Install completion for the specified shell.
  --show-completion [bash|zsh|fish|powershell|pwsh]
                                  Show completion for the specified shell, to
                                  copy it or customize the installation.

  --help                          Show this message and exit.

Commands:
  create-ayx-plugin      Create a new Alteryx plugin in this workspace.
  create-yxi             Create a YXI from the tools in this workspace.
  designer-install       Install the tools from this workspace into Alteryx...
  generate-config-files  Generate the config files for the tools in this...
  generate-tests         Generate the test files for tools in this...
  install-yxi            Install a YXI into Designer.
  sdk-workspace-init     Initialize the current directory as an Alteryx SDK...
  test                   Run the tests command for the language in question.
  version                Display the version of the Alteryx CLI.
```

Step 1: Creating a Workspace

The first step of tool creation is to make a workspace.

Steps:

1. To initialize the tool workspace, we first create a new, empty directory.
2. Run the command inside that directory and answer the prompts, which starts the workspace initialization process.

```
ayx_plugin_cli sdk-workspace-init
```

Note: The **only required** arguments for this command are the package name and the backend language.

```
~$ mkdir sdk-api-tool
~$ cd ./sdk-api-tool/
~/sdk-api-tool$ ayx_plugin_cli sdk-workspace-init
Package Name: API Tool
Tool Category [Python SDK Examples]: Python SDK Examples
Description []: API Tool
Author []: Alteryx
Company []: Alteryx
Backend Language (python): python
```

```
[Workspace initialization] started
[Workspace initialization] .  Create configuration directory
[Workspace initialization] .  Create DCM Schemas directory
[Workspace initialization] .  Create .gitignore
[Workspace initialization] .  Create README.md
[Workspace initialization] .  Initialize backend
[Workspace initialization] Creating ~\sdk-api-tool\backend\ayx_plugins
[Workspace initialization] Creating file ~\sdk-api-tool\backend\requirements-local.txt
[Workspace initialization] Creating file ~\sdk-api-tool\backend\requirements-thirdparty.txt
[Workspace initialization] Creating file ~\sdk-api-tool\backend\setup.py
[Workspace initialization] Creating file ~\sdk-api-tool\backend\ayx_plugins\__init__.py
[Workspace initialization] .  Create tests directory
[Workspace initialization] .  Initialize UI
[Workspace initialization] finished
Created Alteryx workspace in directory: ~\sdk-api-tool
Workspace settings can be modified in: ayx_workspace.json
[Generating config files] started
[Generating config files] .  generate_config_files:generate_config_xml
[Generating config files] Generating top level config XML file...
[Generating config files] finished
```

alteryx

Step 2: Create a Plugin

The next step is to **add a plugin to the workspace**.
Plugins are the individual tools that show up in Designer.
Workspaces can have one or many plugins.

Steps:

1. Run the create plugin command in the Workspace.
2. Answer the prompts and then you will have the template code for your SDK Tool.

```
ayx_plugin_cli create-ayx-plugin
```

Notes:

- The **only required** arguments for this command is the tool name.
- When a YXI is installed into Designer, all the tools from the workspace are installed at once.

```
~/sdk-api-tool$ ayx_plugin_cli create-ayx-plugin
Tool Name: API tool
Tool Type (input, multiple-inputs, multiple-outputs, optional, output,
Description []: My API Tool
Tool Version [1.0]: 1.0
DCM Namespace []:
Creating input plugin: API tool
```

```
[Create plugin] started
[Create plugin] Downloading UI components via git
[Create plugin] Cloning into '.ayx_cli.cache\ui_tool_template'...
[Create plugin] .  Create plugin
[Create plugin] Installing UI components via npm
[Create plugin] Creating Alteryx Plugin...
[Create plugin] Copying example tool to ~\sdk-api-tool\backend\ayx_plugins...
[Create plugin] Added new tool to package directory: ~\sdk-api-tool\backend\ayx_plugins\a_p_i_tool.py
[Create plugin] finished
[Generating config files] started
[Generating config files] .  generate_config_files:generate_config_xml
[Generating config files] Generating top level config XML file...
[Generating config files] .  generate_config_files:generate_tool_config_xml
[Generating config files] Generating tool configuration XMLs...
[Generating config files] Generating APItool XML...
[Generating config files] Done!
[Generating config files] .  generate_config_files:generate_manifest_jsons
[Generating config files] Generating manifest.json for APItool...
[Generating config files] Done!
[Generating config files] finished
[Generating test files for APItool] started
[Generating test files for APItool] .  Generate tests
[Generating test files for APItool] finished
```

Step 2 (Cont.): Anchor Permutations Supported (1)

Single-input-single-output

A tool with a single input anchor and a single output anchor. The default code acts as a passthrough tool (receiving data from the input anchor and writing it to the output anchor).

Input

A tool with no input anchors and a single output anchor. The default code generates a small amount of data and writes to the output anchor.

Optional

A tool with one optional input anchor and one output anchor. The default code combines the functionality of the input and single-input-single-output code (generating data if no input is provided, and passing data through if input is provided).

Output

A tool with one input anchor and no output anchors. The default code takes in a record batch and prints its metadata within Designer.

Step 2 (Cont.): Anchor Permutations Supported (2)

Multiple-inputs

A tool with two input anchors, and one output anchor. The default code acts like a union tool, combining the data from the two input anchors.

Multiple-outputs

A tool with one input anchor, and two output anchors. The default code acts like a filter tool, filtering input data by an integer “Value” column and writing odd rows to the first anchor, and even rows to the second.

Multi-connection-input-anchor

A tool with one multi-connection input anchor, and five output anchors. The default code passes data from the first four connections to the first four output anchors, then performs a union operation on the data from all of the other input connections and writes the union to the last output anchor.

Step 3: Add Business Logic

Once the base plugin and configs are scaffolded into your workspace, **you can now begin coding your business logic.**

We scaffolded four core functions for you to add your business logic:

- `__Init__()`
- `on_incoming_connection_complete()`
- `on_record_batch()`
- `on_complete()`

Note: You will see the generate file named `<TOOL_NAME>.py` inside the workspace directory `~/backend/ayx_plugins/`

```python
class APITool(PluginV2):
    """Concrete implementation of an AyxPlugin."""

    def __init__(self, provider: AMPProviderV2) -> None:
        self.provider = provider
        # truncated code

    def on_incoming_connection_complete(self, anchor: namedtuple) -> None:
        # truncated code

    def on_record_batch(self, batch: "Table", anchor: namedtuple) -> None:
        # truncated code

    def on_complete(self) -> None:
        import pandas as pd
        import pyarrow as pa

        df = pd.DataFrame(
            {
                "x": [1, 2, 3],
                "y": ["hello", "world", "from ayx_python_sdk!"],
                "z": [self.config_value, self.config_value, self.config_value],
            }
        )

        packet = pa.Table.from_pandas(df)

        self.provider.write_to_anchor("Output", packet)
        self.provider.io.info("APITool tool done.")
```

alteryx

Step 3 (Cont.): Where should I add my code?

__Init__()

Initializes relevant properties. It is the **access point** to the Plugin.

The “init” is also the point when anchors are set from the provider.

on_incoming_connection_complete()

Handles any **additional work for a completed anchor**.

The method is called when the **connection is finished** sending records for an anchor.

on_record_batch()

Called for each input connection on an anchor in **batches**.

In this method, the plugin writer can manipulate the data before writing the data to the output anchor.

on_complete()

Called at the **end of the runtime execution**. Do any required cleanup here.

If the plugin is an Input tool, this method is used to read and push to the output anchor.

Step 3 (Cont.): Working with AMPProviderV2

Provider Functions	Usage
`write_to_anchor()`	The method you will use to write data to an output anchor. This takes in the name of the output anchor, and the data in the form of a PyArrow RecordBatch.
`io()`	A set of methods for displaying messages in Designer's Results pane. There are three message types (info, warn, and error) and all can be accessed as methods of provider.io().
`push_outgoing_metadata()`	The method you would use to set metadata on an output anchor. For the most part, Arrows can infer this information from a Table/RecordBatch, but you could set this information manually.
`tool_config()`	The data sent over from the tool's UI component, if applicable, in the form of a Python dictionary.
`environment()`	Designer environment variables (designer version, workflow directory, Designer installation directory, Designer-managed temporary directory, proxy configurations, etc.)
`dcm()`	The methods used to work with Alteryx's Data Connection Manager.

Step 4: Creating a YXI

When your plugin is ready for usage, you will need to package your plugins into a YXI.

Steps:

1. Return to the Workspace.
2. Run the yxi creation command. This command packages your Workspace (plugin, configs, code, metadata) and saves it as a YXI format.

```
ayx_plugin_cli create-yxi
```

Notes: This argument takes no parameters, and **the created YXI will live under "build/yxi/" directory.**

```
~/sdk-api-tool$ ayx_plugin_cli create-yxi
[Creating YXI] started
[Creating YXI] -- generate_config_files:generate_config_xml
[Creating YXI] -- generate_config_files:generate_tool_config_xml
[Creating YXI] .  generate_config_files:generate_manifest_jsons
[Creating YXI] Generating manifest.json for APItool...
[Creating YXI] Done!
[Creating YXI] .  generate_artifact:build_artifacts
[Creating YXI] Creating APItool.yxi...
[Creating YXI] Creating shiv artifact...
[Creating YXI] [Installing local dependencies]: python -m pip install -r
[Creating YXI] [Compiling shiv artifact]: shiv --compile-pyc --reproducib
[Creating YXI] Created shiv artifact at: ~\sdk-api-tool\main.pyz
[Creating YXI] .  create_yxi:create_yxi
[Creating YXI] finished
```

What is a YXI?

YXI is Alteryx specific packaging archive containing everything needed to install a new tool into Designer

Step 5: Install Plugin to Designer

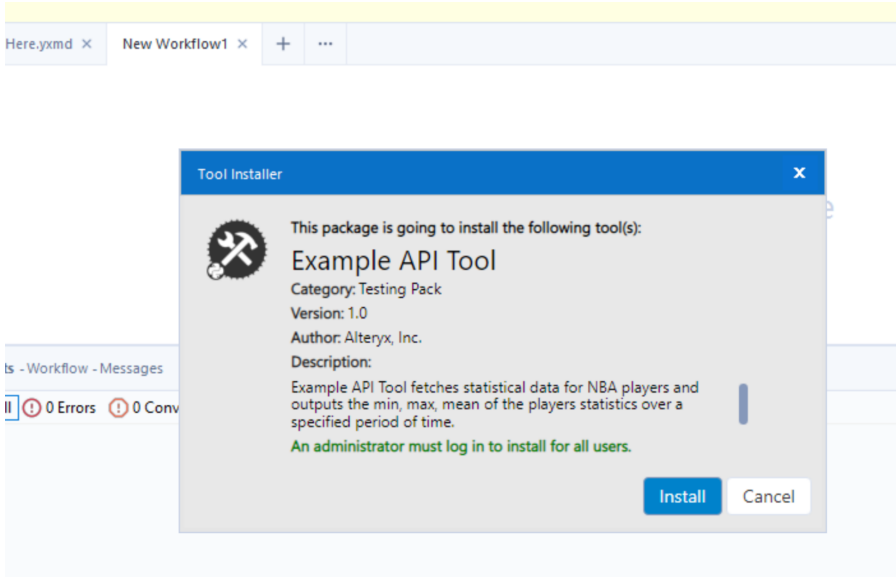
The final step is to install into Designer!!!

Several options:

- install-yxi installs ANY arbitrary YXI given a path. (For Sharing)
- `ayx_plugin_cli install-yxi`
- designer-install creates and installs the YXI from the current workspace. (Composite of create-yxi and install-yxi)
- `ayx_plugin_cli designer-install`
- Double clicking the YXI: After you create a .yxi, you can double-click the .yxi to install it into Designer. This opens Designer and prompts you to install the package in a new dialog box.

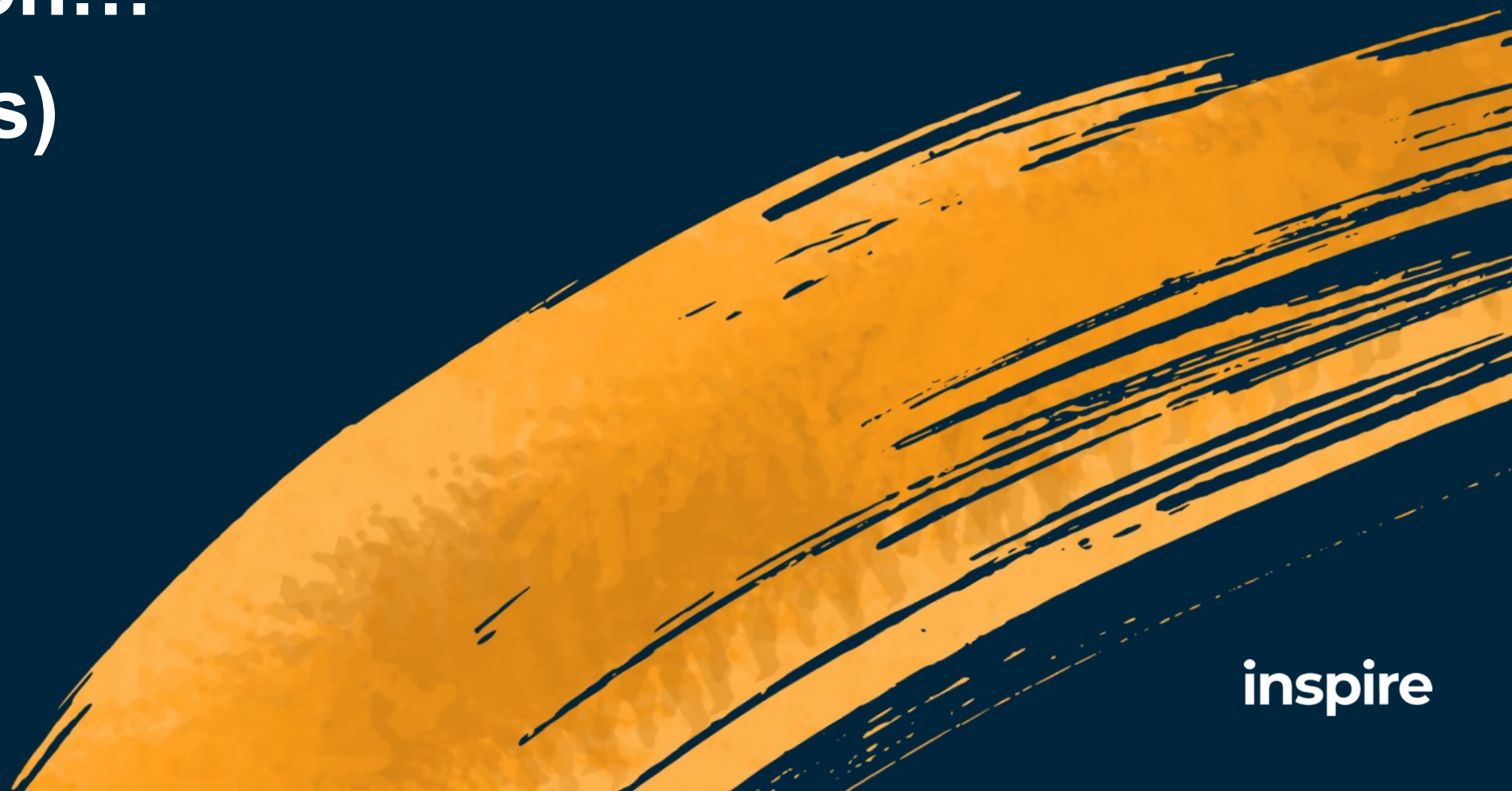
Warning: First time tools installations require a close and relaunch of Designer. For subsequent reinstallations, tools dragged onto Designer's canvas should reflect the most up-to-date state

```
~/sdk-api-tool$ ayx_plugin_cli designer-install
Install Type (user, admin) [user]: user
[Creating YXI] started
[Creating YXI] -- generate_config_files:generate_config_xml
[Creating YXI] -- generate_config_files:generate_tool_config_xml
[Creating YXI] .  generate_config_files:generate_manifest_jsons
[Creating YXI] Generating manifest.json for APItool...
[Creating YXI] Done!
[Creating YXI] .  generate_artifact:build_artifacts
[Creating YXI] Creating APItool.yxi...   # <-- .yxi generated here
[Creating YXI] Creating shiv artifact...
[Creating YXI] [Installing local dependencies]: python -m pip install -r requirement
[Creating YXI] [Compiling shiv artifact]: shiv --compile-pyc --reproducible --extend
[Creating YXI] Created shiv artifact at: ~\sdk-api-tool\main.pyz
[Creating YXI] .  create_yxi:create_yxi
[Creating YXI] finished
[Installing yxi ~\sdk-api-tool\build\yxi\APItool.yxi into designer] started
[Installing yxi ~\sdk-api-tool\build\yxi\APItool.yxi into designer] .  install_yxi
[Installing yxi ~\sdk-api-tool\build\yxi\APItool.yxi into designer] finished
If this is your first time installing these tools, or you have made modifications to
```



alteryx

Intermission!!!
(10 Minutes)



alteryx

inspire

Walkthrough #2

Data Processing at Scale

Training Material:
https://github.com/alteryx/ayx-developer-sdk/tree/main/inspire

Closing Thoughts and Office Hours

“We are committed to empowering Community Developers and Partners to extend the latest Alteryx innovations”

alteryx

Continuing at Inspire

Alteryx Extensions, SDKs, and Exchange, Oh My!

Learn how easy it is to extend multiple Alteryx products. Then discover new types of extensions coming to the cloud soon and take a peek at an upcoming concept for distributing and sharing Alteryx extensions with others on the Alteryx Exchange.

Peter Ott, Manager of Product Management
Magen Harron, Principal Product Manager

Solution Center

Visit us at the Solution Center and we can directly help you. You can find us in the booth labeled “SDK”.

- Installation and Step up.
- Solutioning.
- Troubleshooting and Support.
- Want to just chat!!!

Continue the Alteryx Journey

Alteryx.IO

Centralized place for **developer products, education, documentation, guides, and inspiration** necessary to enhance your Alteryx experience through SDKs, APIs, Extensions, and Custom Applications at developer.alteryx.com

Alteryx Community

Post your questions, search discussion forums, read Knowledge Base articles and contribute your knowledge at community.alteryx.com

Alteryx Academy

Complete Learning Paths and Interactive Lessons, participate in the Weekly Challenge and learn about product certification at academy.alteryx.com (*Developer Academy Coming Soon)

Now please join us for a brief survey

Survey located in this session's description in the Inspire app

What will **you** build with Alteryx SDKs?

alteryx

Thank You

alteryx

inspire

References

Additional Functionality: Building Tools with UI

For custom tools that require frontend components (icons, forms, input boxes, dropdowns, etc.), Alteryx offers a UI-SDK that is a companion to the Platform SDK.

UI SDK

- The AYX UI SDK is a React-based software development kit that lets you build custom applications to use within the Alteryx platform.
- The UI SDK provides many product-specific messaging systems. You can use these across the Alteryx platform, as well as the entire core segment of the Alteryx Components library.
- The UI SDK consists of 3 components:
  - The **Communication Bridge**
  - The **Alteryx Component Library**
  - All associated walk-throughs and help guides.
- For the real-time Dev Harness that assists in developing a tool UI, check out **this repository**.

Additional Functionality: Unit Testing

The CLI autogenerates a set of unit tests alongside the plugin code.

These tests can help you quickly iterate on your business logic by running your code's individual methods and allowing you to compare them against expected output, without having to rebuild the YXI every time.

```python
@pytest.mark.parametrize("record_batch_set", ["small_batches", "medium_batches", "large_batches"])
@pytest.mark.parametrize("anchor", [
    Anchor("Input", "1"),
])
def test_on_record_batch(scratch_passthrough_plugin_service, anchor, record_batch_set, request):
    """
    This function is where you should test your plugin's on_record_batch method.
    Use scratch_passthrough_plugin_service.run_on_record_batch to run the specified record batch
    through the specified input anchor.

    Once the method has run, you can compare the output data against expected values,
    by accessing the corresponding data from scratch_passthrough_plugin_service.data_streams.
    Use the output anchor name as the dictionary key.
    If no data was written, scratch_passthrough_plugin_service.data_streams will be an empty dictionary.

    You can also compare IO calls made to designer through scratch_passthrough_plugin_service.io_stream.
    The message type (INFO, WARN, ERROR) will be prepended to the message's text with a colon.
    If no provider.io methods were called, scratch_passthrough_plugin_service.io_stream will be an empty list.
    """
    input_record_batch, expected_output_record_batch = request.getfixturevalue(record_batch_set)
    scratch_passthrough_plugin_service.run_on_record_batch(input_record_batch, anchor)
    assert scratch_passthrough_plugin_service.data_streams["Output"] == [expected_output_record_batch]
    assert scratch_passthrough_plugin_service.io_stream == []
```

Additional Functionality: Testing Client

Maximize development cycles by **testing your tools** ahead of time **without Designer.**

The standalone **ayx-sdk-cli** Test Client offers the ability to **simulate the "run"** of a plugin while allowing passing of input and output testing data for supported custom tools formats and anchor permutations while producing comprehensive debug logs of the simulated "run".

Notes:

- Download the Test Client
- The Test Client does NOT use Engine/AMP at all.
- The Test Client "runs" the plugin and **mimics** passing data as if the Designer backend was doing it.

Additional Functionality: DCM

Improve **security** by moving your credentials outside the workflows and **synchronizing** them across the Alteryx product suite.

Securely Store

Secure and store all your credentials. Credential storage can be used to securely store databases keys, cloud services tokens and more.

Synchronize

Connection Objects join data sources to credentials and are used by one or many tools in your Workflows. DCM stores your credentials and syncs them across the Alteryx product suite.

Management

Centralize location to create, read, update, and delete your credentials across Alteryx product suite.

Arrows or Pandas?

Arrows is built to be very fast and very memory-efficient.

It's excellent at working with large datasets and has great documentation and support available.

PyArrows has integrations with Pandas:

```
# Convert from pandas to Arrow
table = pa.Table.from_pandas(df)
# Convert back to pandas
df_new = table.to_pandas()
```

These conversions can be rather slow. Try to work within PyArrow as much as you can, and if you do need to convert back and forth between PyArrow and Pandas, do so sparingly.

alteryx

scratch_multi_output.py

```
49          metadata = batch.schema
50          if not any([field_name == "Value" for field_name in metadata.names]):
51              raise RuntimeError(
52                  "Incoming data must contain a column with the name 'Value'"
53              )
54          if not pa.types.is_integer(metadata.field("Value").type):
55              raise RuntimeError("'Value' column must be of 'int' data type")
56          input_dataframe = batch.to_pandas()
57
58          grouped = input_dataframe.groupby("Value")
59          odds = grouped.filter(lambda row: (row["Value"] % 2 == 1).any())
60          evens = grouped.filter(lambda row: (row["Value"] % 2 == 0).any())
61
62          odd_batch = pa.RecordBatch.from_pandas(odds, preserve_index=False)
63          even_batch = pa.RecordBatch.from_pandas(evens, preserve_index=False)
64          self.provider.write_to_anchor("Output1", odd_batch)
65          self.provider.write_to_anchor("Output2", even_batch)
66
```

ScratchMultiOutput › on_record_batch()

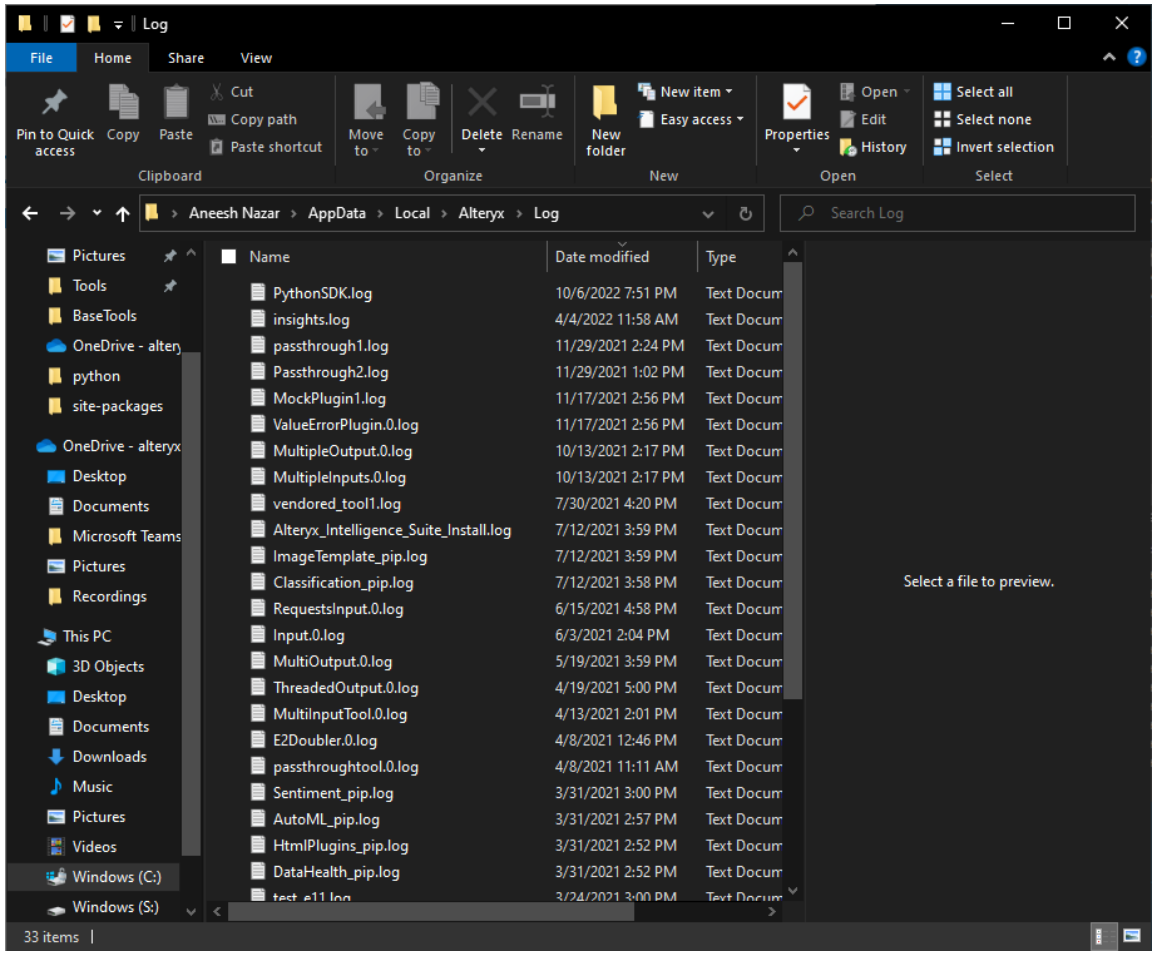
scratch_multi_output_no_pandas.py

```
48          metadata = batch.schema
49          if not any([field_name == "Value" for field_name in metadata.names]):
50              raise RuntimeError(
51                  "Incoming data must contain a column with the name 'Value'"
52              )
53          if not pa.types.is_integer(metadata.field("Value").type):
54              raise RuntimeError("'Value' column must be of 'int' data type")
55
56          even_batch = batch.filter(pc.equal(pc.bit_wise_and(batch[0], 1), 0))
57          odd_batch = batch.filter(pc.equal(pc.bit_wise_and(batch[0], 1), 1))
58
59          self.provider.write_to_anchor("Output1", odd_batch)
60          self.provider.write_to_anchor("Output2", even_batch)
61
62
```

Where do I find developer/tool logs?

You can find your log files under %LOCALAPPDATA%\Alteryx\Log

Look for <plugin name>.log and PythonSDK.log



alteryx

inspire

Running the plugin artifact

Sometimes, you may see a “Internal error: Failed to read port assignment” issue in Designer. Sometimes, this can be due to simple syntax errors

If you navigate to %APPDATA%\Alteryx\Tools after installing, then find the folder with your artifact in it (use the manifest.json file to locate the main.pyz artifact), you can run

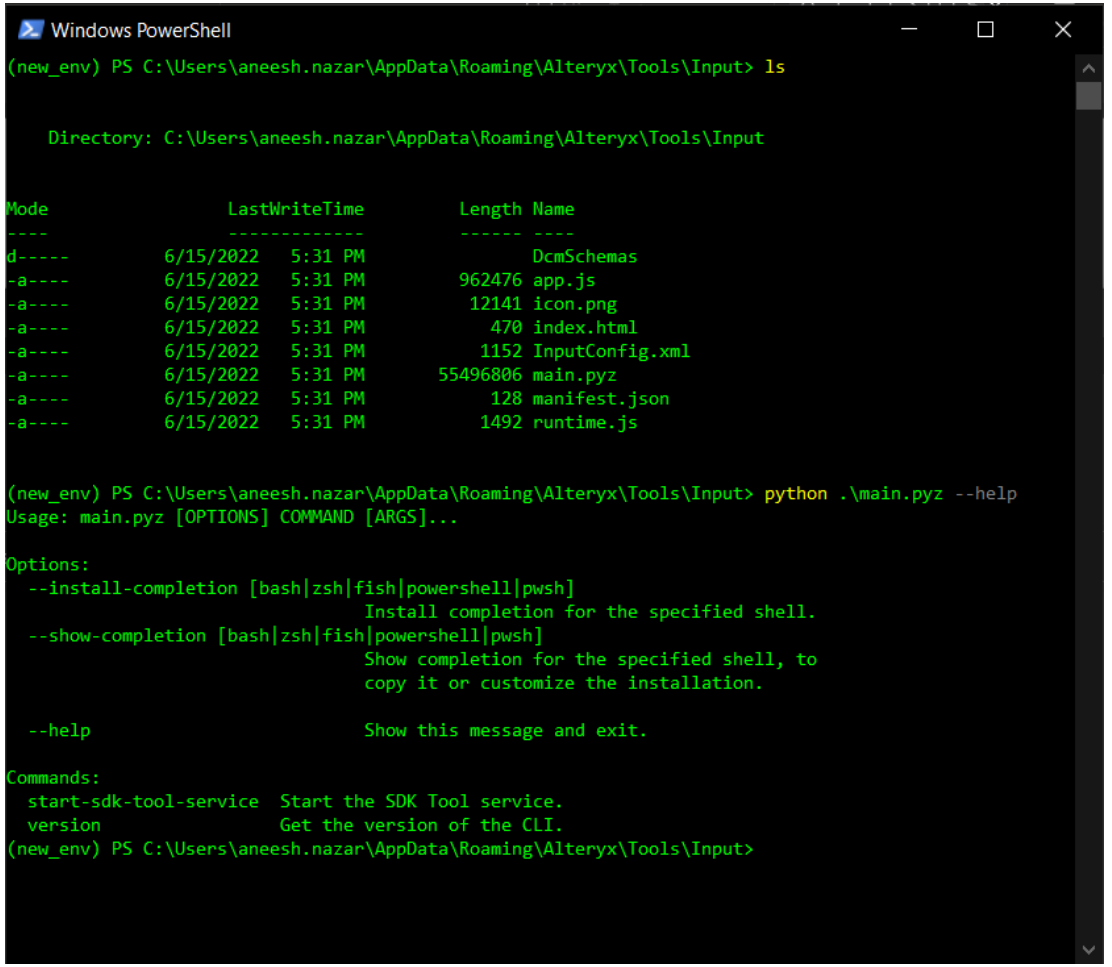
```
python .\main.pyz --help
```

In order to ensure that your python code is being imported properly.

If this works, then try running

```
python .\main.pyz start-sdk-tool-service
ayx_plugins <Tool you want to test>
```

To ensure the service is starting up properly.



alteryx

inspire

Core Concept: PluginV2 Class (Detailed Version)

PluginV2 Class

The Plugin class is the basis. The Plugin class provides the **required abstract operations that need to be implemented so that a tool can interact with Alteryx Designer**. These interactions are mediated by the "Providers", which provide simplified interfaces for Designer functionality and drive the execution of the Alteryx Plugin Tools.

Registering the Plugin

Every plugin must be registered with the SDK after the new Tool class is defined. The Tool must implement the base PluginV2 class so that the SDK can acknowledge the registered plugin. The registration process indicates to the SDK that the Plugin exists, what the name of the class is, and provides a means of execution of the Alteryx Plugin Tools.

Four Core Functions:

- **__Init__()**
- **on_incoming_connection_complete()**
- **on_record_batch()**
- **on_complete()**

Core Concept: Four Functions (Detailed Version)

__Init__()

The __init__ method in the Ayx Plugin Tool class **initializes relevant properties**. It is also the access point for the BaseProvider object to all of the Plugin methods, so the provider is typically stored as a class variable in the init method.

The init is also the point when anchors are set from the provider.

on_incoming_connection_complete()

The on_incoming_connection_complete **method is called to handle any additional work for a completed anchor**. **The method is called when there are no more records left.** It notifies the plugin that the connection is done sending data.

This method receives an Anchor object that contains the anchor name (anchor.name) and the corresponding input connection (anchor.connection).

on_record_batch()

The on_record_batch method is **called for each input connection on an anchor.** This method also receives an Anchor object in the form of a PyArrows Table.

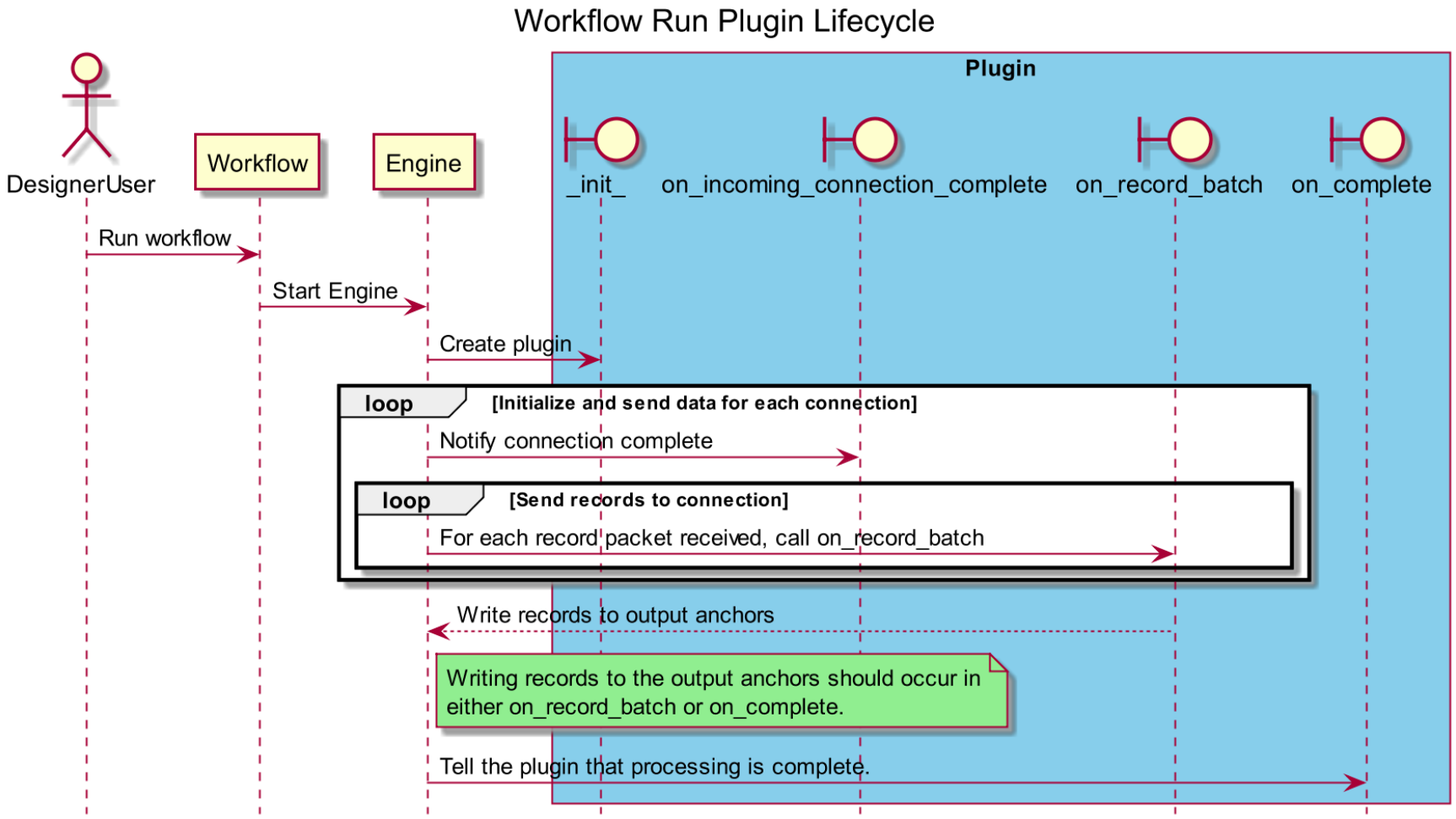
In this method, the plugin writer can manipulate the data before writing the data to the output anchor
using self.provider.write_to_anchor(self.output_anchor_name, table)

on_complete()

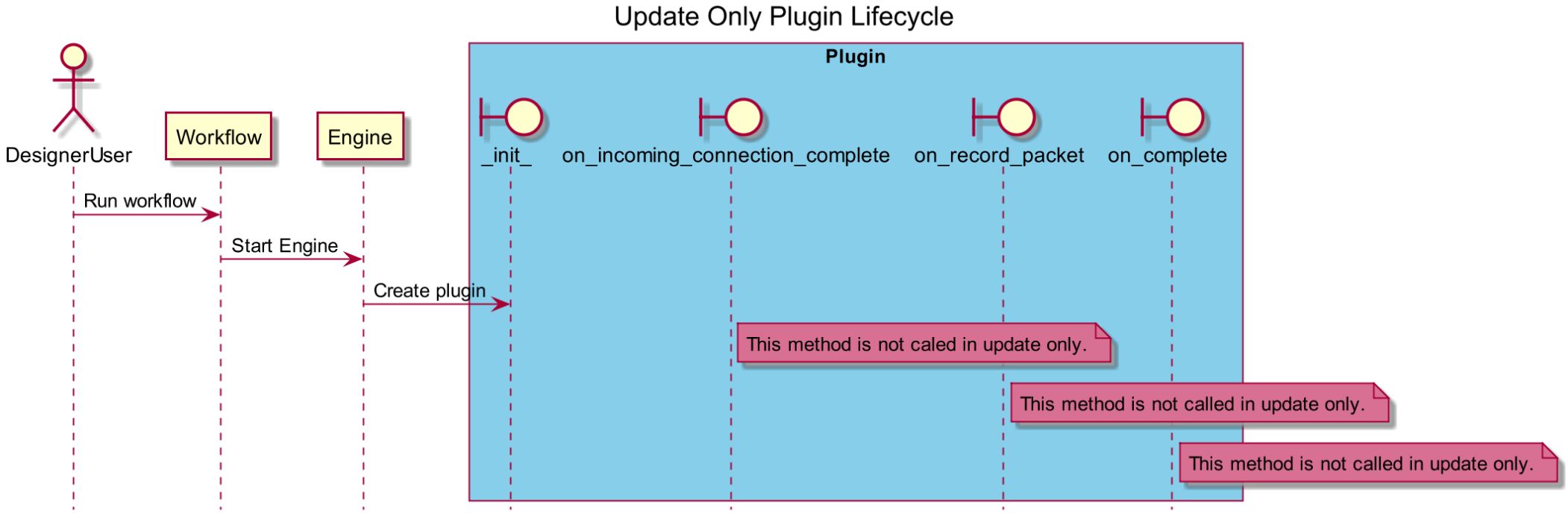
The on_complete method is called at **the end of the runtime execution**.

This typically does any cleanup required for the Plugin. If the plugin is an Input tool-type, this method is used to read in the data from the datasource and push it to the output anchor.

Core Concept: Lifecycle of a Plugin



Core Concept: Lifecycle of a Plugin (for Update Only)



Update only “Run” is a mode that runs in Designer all the time. The purpose of this mode is to generate the metadata that each tool will output during the next time a workflow runs. This allows new tools on the canvas to know what columns they can operate on.

Update only “Run” occurs when:

- New tool is added to the canvas.
- Event on Designer interface effects a tool.
- Change in the configuration panel.
- Tool on the canvas has a change in configuration.