IOblend

IOblend Developer Handbook

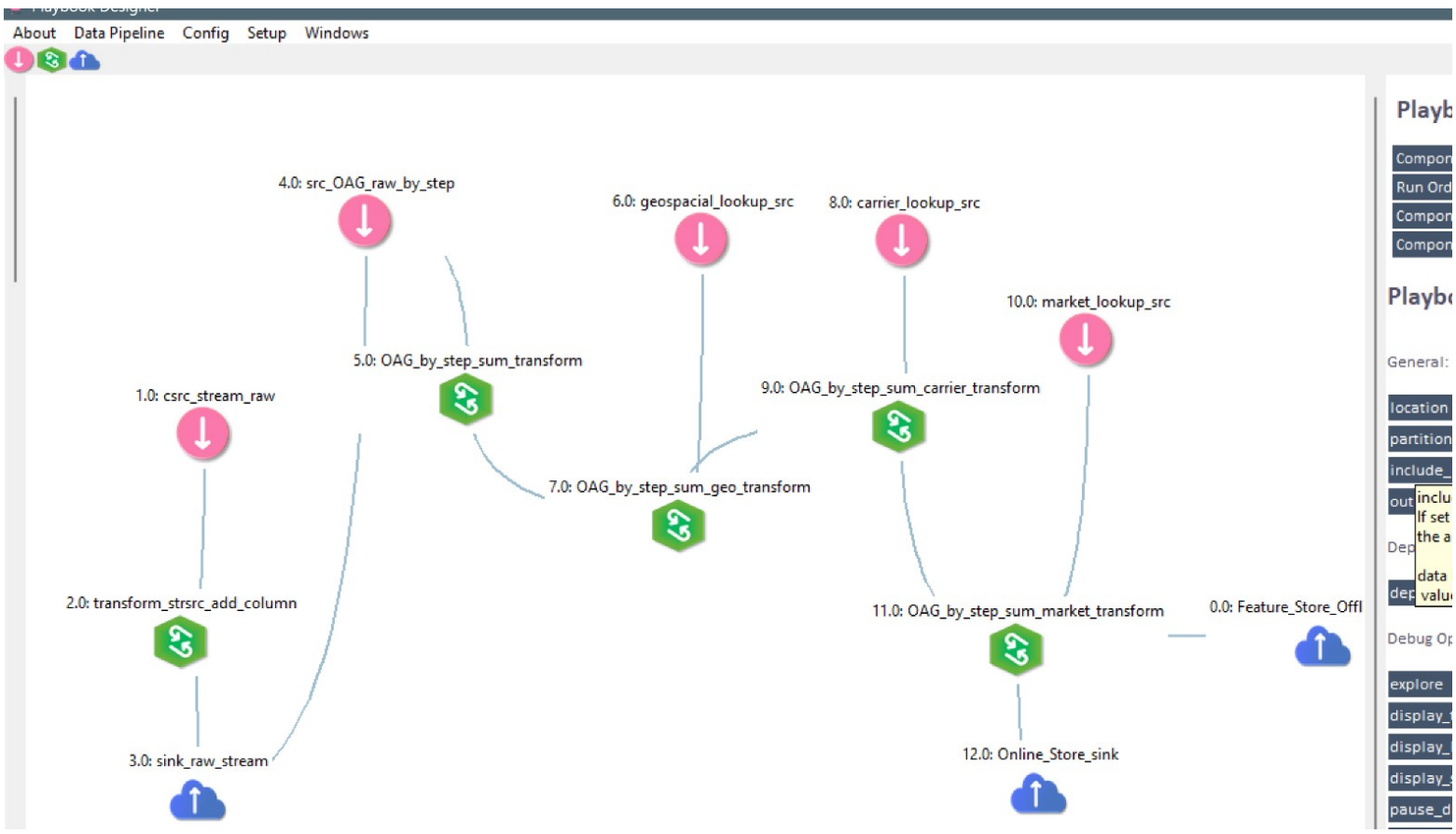
INSTALLATION / SETUP & CONFIGURATION
PIPELINE BUILD

Developer Edition • Integrated reference and practical labs

Edition v3 | November 2026

TECHNICAL ENABLEMENT / DEVELOPER REFERENCE

About this handbook



Playbook Designer: the canvas shows the pipeline; the selected component is configured in the right-hand panel.

ABOUT THIS HANDBOOK

One practical reference for installing IOblend on Windows or macOS, configuring the local Engine, building and debugging a pipeline, connecting an S3 source or sink, and exploring streaming, lineage and JDBC. Start with the local CSV exercise before using cloud resources.

For developers, analysts and engineers getting started with IOblend. Basic SQL and file-system familiarity are useful; no Spark coding is required for the first lab.

Environment: Developer Edition on the developer's own workstation. Production workloads use Enterprise Edition and the organisation's approved deployment environment.

How to use this handbook

Start with Part I to install IOblend, then follow Part II to set up your project and runtime. Part III contains the guided labs and technical references, beginning with a local CSV pipeline. You can return to any part independently once your development environment is ready.

Phase	What you will do	Completion check
1. Install	Extract the distribution using 7-Zip and initialise the workstation	Designer and Engine terminal launch
2. Configure	Create folders and project Run Parameters	Saved configuration with a named playbook directory
3. Build	Connect a local source, SQL transform and sink	First result file with checked calculations
4. Inspect	Read logs, playbooks and lineage metadata	Can explain a record from input to output
5. Extend	Use S3, a stream and a lookup	A repeatable experiment with isolated test resources

USE A TRAINING PROJECT

Keep exercises separate from important data and systems. Create dedicated input, output, checkpoint and state locations. Your development machine is not a production container, but a pipeline can still read from or write to remote services if the credentials allow it.

Contents

PART I | INSTALLATION

1.1 Windows installation

1.2 macOS installation

PART II | SETUP & CONFIGURATION

2.1 Create a development project

2.2 Complete Run Parameters setup

2.3 Windows: Spark startup troubleshooting

PART III | PIPELINE BUILD

3.1 Lab 1: your first working pipeline

3.2 Logs and playbooks: what the Engine uses

3.3 Lab 2: connect to Amazon S3

3.4 Lab 3: streaming and event management

3.5 Lab 4: lookup joins and lineage

3.6 JDBC database destinations

3.7 Explore the aviation training project

3.8 Troubleshooting and good habits

For rare Windows timeouts while initialising Spark, work through Part II, Section 2.3 first. The checks below focus on pipeline components once the Engine is running.

REFERENCE APPENDICES

IOblend

DEVELOPER HANDBOOK / REFERENCE GUIDE

PART I

Installation

Install the Developer Edition and initialise the local Designer and Engine. Choose the procedure for your workstation.

01
Windows installation

02
macOS installation

PART I | INSTALLATION

Install the Developer Edition and initialise the local Designer and Engine. Choose the Windows or macOS procedure for your workstation.

1.1 Windows installation

Before you start

IOblend Developer Edition runs the Designer and Engine on the developer workstation, and initialises a local Spark environment. It can reach cloud and on-premises data sources and sinks for which your machine has the necessary access.

Item	Recommended / minimum
Processor	Intel or AMD; 3 GHz recommended (2 GHz minimum).
Memory	16 GB or more recommended (4 GB listed as minimum).
Storage	30 GB recommended (10 GB minimum). Use a fast local SSD where possible.
Display	1024 × 768 or higher; Windows DirectX 9/10 compatible graphics.
Connectivity	Internet access for downloading the software and connecting to remote services.
Archive extractor	7-Zip. Required for the IOblend distribution in this handbook.

Step-by-step Windows setup

1. Download the Windows Developer Edition from the IOblend registration/download page. Submit the form and use the download link shown in the confirmation message. Save the complete ZIP on your workstation.
2. Install 7-Zip if it is not already available. Do not unpack the IOblend package with the regular Windows compressed-folder extractor. That method may omit files from this distribution, leaving an incomplete installation.
3. Right-click the downloaded ZIP, choose 7-Zip > Test archive (optional but useful), then 7-Zip > Extract to "<archive-name>\". Wait for the extraction to finish, with no reported errors.
4. Place the fully extracted IOblend folder on your local C: drive, in a short, ordinary path such as C:/IOblend/ioblend_designer. Use the path appropriate for your release and do not run it directly from inside the ZIP or a remote synchronised folder.
5. Open the extracted folder in File Explorer and locate the ioblend_designer application. Create a desktop shortcut if convenient, then launch it.
6. Keep both windows open: the Script/terminal window and the Playbook Designer window. They are part of the local development experience.
7. In the Designer select Setup > Setup Environmental Vars. The dialog warns that environment variables will be updated. Back up the current environment first, then confirm the setup.
8. To back up environment variables, open Command Prompt, run set, and save the output to a private text file. Environment listings can contain sensitive values; retain the backup locally with suitable access controls.

9. Return to the setup dialog, choose OK and check the Script window. The setup sequence reports six success messages when the environment-variable update is complete.
10. Close and reopen Designer if required for the environment change to take effect. Confirm the GUI and Engine terminal start without missing-file errors.

```
:: Windows Command Prompt - backup a copy of the current environment
set > "%USERPROFILE%\Documents\ioblend_env_before.txt"
```

POWERSHELL ON DEVELOPER MACHINES

If local PowerShell script execution is restricted, apply the execution-policy setting approved for your development workstation. The standard setup command is Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope LocalMachine where permitted; Windows developers or workstation administrators can choose the appropriate scope. This setting relates to the developer workstation, not the production runtime.

```
# Windows PowerShell - installation procedure where approved
Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope LocalMachine
```

If endpoint protection challenges a bundled Spark, Java or Python component, work with the workstation configuration to trust the verified IOblend files and required processes. Make only the exclusions necessary for this development installation.

Check that Windows setup succeeded

- ☐ The archive was extracted with 7-Zip and no extraction errors were reported.
- ☐ The entire ioblend_designer folder is on the local drive.
- ☐ Designer and Script windows open.
- ☐ Environmental setup completed and reported six success messages.
- ☐ A new project can be created without missing components or modules.

1.2 macOS installation

Prerequisites

Use a modern Mac with approximately 16 GB RAM and sufficient SSD space. The macOS developer package is approximately 2 GB to download. You will need internet connectivity, Terminal access, Python 3.10.11 and the complete IOblend archive.

Step-by-step macOS setup

1. Download the macOS Developer Edition from IOblend's registration/download page and save the ZIP locally.
2. Install Python 3.10.11 using the macOS 64-bit universal installer. Allow the installer to update the relevant shell profile if prompted.
3. Open Terminal and verify that the active interpreter is Python 3.10.11. Confirm its installation location before proceeding.

```
python3 --version
python3 -c "import os,sys; print(os.path.dirname(sys.executable))"
```

4. Create and activate a Python virtual environment dedicated to IOblend. Replace <username> with your macOS user account name or use the $HOME form shown below.

```
python3 -m venv "$HOME/python_venvs/p310"
source "$HOME/python_venvs/p310/bin/activate"
python3 --version
```

5. With that environment active, install the required Python modules. The module list below is the Developer Edition dependency set for this package.

```
python3 -m pip install pyinstaller Pillow Tornado psutil boto3 \
  retrying tzlocal py4j tabulate pandas cryptography
python3 -m pip install --upgrade google-cloud-storage
python3 -m pip install azure-storage-blob azure-identity \
  azure-common azure-storage-file-datalake pyarmor
```

The macOS package specifies these module names, including Google Cloud Storage and Azure storage/identity libraries. Keep the environment active when preparing to run IOblend. To leave it later, use deactivate.

6. Use the 7-Zip command-line extractor (7zz) or a 7-Zip-based archive tool to extract ioblend_designer_mac.zip completely. Do not rely on the default unzip/archive tool for this IOblend distribution. The example below assumes the 7zz command is installed.

```
mkdir -p "$HOME/IOblend"
7zz t "$HOME/Downloads/ioblend_designer_mac.zip"
7zz x "$HOME/Downloads/ioblend_designer_mac.zip" -o"$HOME/IOblend"
```

7. Navigate to the extracted ioblend_designer directory and open the ioblend_designer executable. Keep the terminal it launches open; the Playbook Designer starts alongside it.
8. In Designer choose Setup > Setup Environmental Vars. This generates io_env_setup.sh in the IOblend Designer directory.
9. Inspect io_env_setup.sh, then add its export statements to your active shell profile so the required component directories and environment variables are available. For a typical zsh login configuration, the profile may be ~/.zprofile; use the profile that applies to your workstation.

```
# Inspect the generated file before applying it to your shell profile
cat /path/to/ioblend_designer/io_env_setup.sh

# After copying the required export statements into your shell profile:
source "$HOME/.zprofile"    # if this is the profile you edited
```

10. Close both IOblend windows, then reopen ioblend_designer to pick up the configured environment.

FIRST LAUNCH ON MACOS

macOS may ask you to confirm an application from an unidentified developer. Review the downloaded package and use the system's approved app-specific opening procedure (for example, Open Anyway in Privacy & Security, where available). If local policy requires another path, follow the developer workstation procedure and restore normal protection after setup. No production security setting needs to change.

Check that macOS setup succeeded

- ☐ python3 reports 3.10.11 in the intended environment.

- [ ] Required Python modules installed without errors.
- [ ] The complete IOblend archive was extracted using a 7-Zip-based tool.
- [ ] Designer and terminal windows open.
- [ ] io_env_setup.sh exports are present in the selected shell profile.
- [ ] Designer starts successfully after a restart.

IOblend

DEVELOPER HANDBOOK / REFERENCE GUIDE

PART II

Setup & configuration

Prepare project folders, credentials, paths and run parameters before creating pipeline components.

01
Development project

02
Run Parameters

03
Spark startup checks

DEVELOPER EDITION • NOVEMBER 2026 • V3

PART II | SETUP & CONFIGURATION

Prepare the project folders, credentials, paths and run parameters before creating any pipeline components.

What you are building

Term	Meaning
Playbook Designer	Desktop GUI used to design, configure, debug and save source, transformation and sink components.
Playbook	The JSON component definition that preserves logic and operational options.
IOblend Engine	Runtime that interprets the playbooks and executes Spark/Python processing.
Run Parameters	The project-level settings that tell the Engine where to find the playbooks, state, credentials and runtime.
Epoch	One execution cycle; setting Epochs to 0 requests continuous execution until stopped.

2.1 Create a development project

Projects organise the playbooks and the paths the Engine will use. Create a new project before adding components. The layout below follows the structure of a working aviation training project; your first project can start with empty directories.

```
C:/IOblendLabs/sensor-lab/
  config/          Run Parameters, credential templates, JDBC mapping
  playbooks/       One JSON file per playbook component
  transforms/      External Python transformations
  jars/            JDBC driver JARs
  source_data/     Your local input CSV files
  output/          Output from the local sink
  checkpoint/      Stream checkpoint files
  state/           Execution state, logs, event archives
  interaction/     Designer/Engine interactive files
```

On macOS replace C:/IOblendLabs/sensor-lab with a suitable local path such as /Users/<username>/IOblendLabs/sensor-lab.

What lives where

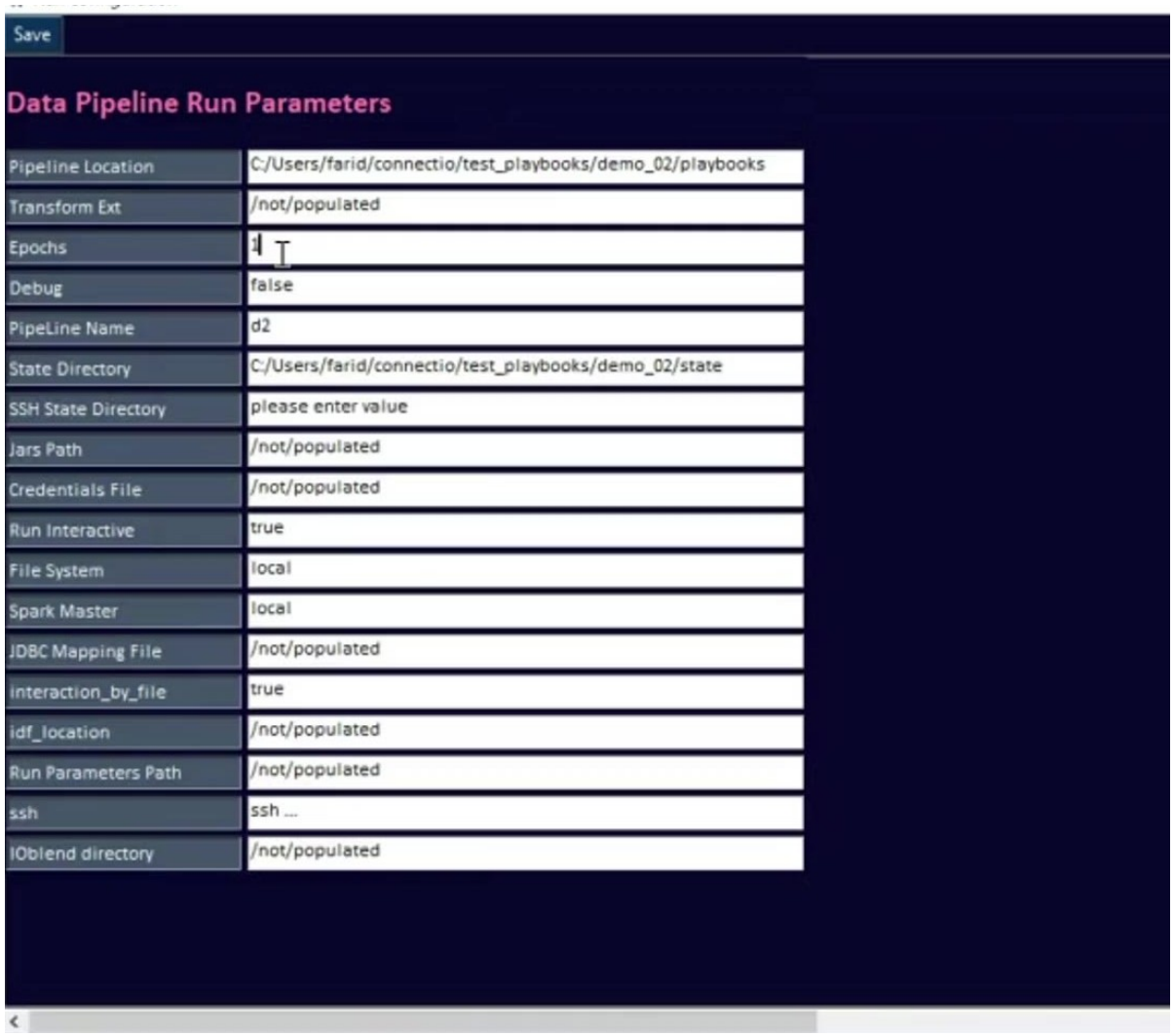
Location	Developer action
playbooks/	Use Designer to create and edit components. Inspect the JSON to understand what gets saved.
config/	Create project Run Parameters and connector settings. Keep any credential file private.
source_data/	Place test input files here. Keep it separate from every sink output path.
state/ and checkpoint/	Let the Engine manage runtime artefacts; do not reuse checkpoints across unrelated tests.
interaction/	Provides interactive communication between Designer and Engine.
output/	Open the output data and verify counts, headers, calculations and metadata fields.

FOLDER SEPARATION MATTERS

Never point a streaming source at the same directory in which its sink writes data unless that feedback is deliberately part of the design. Otherwise generated files can be read again as new inputs.

2.2 Complete Run Parameters setup

Run Parameters are project-specific. Set them first: Config > Project Run Params > New Run Params. Populate the required fields in the Run Configuration window and save. Hover over field labels to see their tooltips.



Run Configuration in the Designer. All paths in the exercises below refer to your own project, not the machine in this example.

2.2.1 Core paths and identity

Field	Example / value	What it controls
Pipeline Location	C:/IOblendLabs/sensor-lab/playbooks	Directory containing component JSON playbooks.
PipeLine Name	sensor_lab_01	Unique project/pipeline identity reported in output metadata and logs.
Transform Ext	C:/IOblendLabs/sensor-lab/transforms	Directory for external Python transformation scripts.
State Directory	C:/IOblendLabs/sensor-lab/state	Location for generated internal execution state.
Run Parameters Path	C:/IOblendLabs/sensor-lab/config	Directory in which project Run Parameters are stored.

Field	Example / value	What it controls
File System	local	File system for the run configuration artefacts. Local for this exercise.
Credentials File	C:/IOblendLabs/sensor-lab/config	Location of the security configuration/credentials file; see Section 7.
Jars Path	C:/IOblendLabs/sensor-lab/jars	Directory for JDBC and related JAR dependencies.
JDBC Mapping File	C:/IOblendLabs/sensor-lab/config	Directory holding jdbc_db_sql.json, the database SQL-template mapping.
IDF location	C:/IOblendLabs/sensor-lab/interaction	Folder used for interactive Designer/Engine exchange files.

The Pipeline Location, Transform Ext, Jars Path, Credentials File, JDBC Mapping File and Run Parameters Path must use the file system selected by File System. The supplied package may display directory paths while the Engine resolves individual filenames within them.

2.2.2 Execution and interactive debugging

Field	Beginner value	Meaning / action
Epochs	1	Run one cycle; set to 0 for continuous operation until terminated.
Debug	true	Enable development-time debugging and step-by-step inspection.
Run Interactive	true	Display intermediate pipeline results as the pipeline runs.
interaction_by_file	true	Use interaction files to communicate between Designer and Engine.
Spark Master	local	Run Spark on the workstation. local[*] is another local option.
SSH	local	Use the local Engine; remote SSH execution is not enabled in Developer Edition.
IOblend Directory	not required for local	Absolute path to Engine binaries used for SSH deployment in Enterprise Edition.

For unattended Enterprise processing the three development-time flags Debug, Run Interactive and interaction_by_file are set to false. Developer Edition remains a development and testing environment.

2.2.3 Additional parameters you may see

Field	What to know
idf_source	Specifies the file system for interactive files; local in the aviation example.
Cache	Pipeline caching switch; the aviation example uses true.
test_pandas_dir	Optional test directory field; not populated in the aviation training run.
pickle_reads	Present in the sample project; leave the supplied value unless your workflow uses it.
sql_scripts_path	Additional SQL script directory path, not populated in the sample run.
parent_parent_path	State parent path as recorded in the sample run parameters.
spark_submit_command	Optional configured path to a Spark submission-command file.
SSH State Directory / SSH remote OS	Remote execution settings; not needed for local Developer Edition.
Spark submit overrides	Use matched driver-memory and JVM initial-heap values. See Section 4.4 for copy-ready spark-submit flags.

2.2.4 Spark submit: driver and Java heap memory

Some workloads need a larger initial Java heap. Set it alongside a compatible Spark driver-memory limit: -Xms must not exceed the JVM maximum heap established by Spark driver memory. Choose a profile appropriate to the available workstation RAM.

Larger-memory developer workstation (typically 48 GB RAM or more):

```
--conf spark.driver.memory=24g --conf spark.driver.extraJavaOptions=-Xms20g
```

Standard developer workstation (typically 16 GB RAM or more):

```
--conf spark.driver.memory=8g --conf spark.driver.extraJavaOptions=-Xms4g
```

Use one profile. An 8g/20g combination fails because -Xms exceeds the maximum heap. Apply these settings before Spark starts, and reserve RAM for the OS and Python.

FIRST-RUN SETTINGS TO CONFIRM

Before building components, reopen the saved project configuration and check: PipeLine Name, Pipeline Location, State Directory, Spark Master = local, SSH = local, Epochs = 1, and all three debugging flags = true.

2.3 Windows: Spark startup troubleshooting

On a small number of Windows development workstations, the Engine may appear to pause while starting Spark. The usual investigation is the local environment: archive extraction, Java, free memory, endpoint protection, file access or Windows networking. Follow these checks in order. They do not require changing your playbooks or business SQL.

FIRST RULE

Use the simple local CSV lab for diagnosis. Change only one workstation setting at a time, then retry. Keep security protections in place; where a policy blocks a trusted IOblend component, work with the workstation administrator to approve a targeted exception.

What you observe	Where to start
No Java process appears	Steps 2 and 3: extracted files, launch path, Java installation and environment
Java starts, then closes	Steps 3 and 4: Java compatibility, JVM memory and captured launch errors
Java remains running; Designer waits	Steps 5 and 6: interaction-file writes, endpoint protection, hostname and loopback
Spark session exists, but the pipeline stops later	Part III, Section 3.8: playbook, source, transformation and sink troubleshooting

2.3.1 Step 1: Record the exact point of failure

Launch IOblend normally and keep both the Script/terminal window and the Designer visible. Record the time, the first error (not just the final timeout), and the last successful startup message. Look for “setting up spark session”, “spark session created”, Java gateway errors, and the interaction-file path. Do not delete logs or state until they have been copied.

```
Get-Date -Format "yyyy-MM-dd HH:mm:ss"
```

If “spark session created” appears, move directly to the interaction-file and communication checks in Step 5. If the Java gateway exits or the Spark session never appears, proceed through Steps 2–4.

2.3.2 Step 2: Confirm installation and project directories

Make sure the IOblend package was extracted using 7-Zip, not the standard Windows ZIP extractor. Keep the installation on a local drive (preferably C:) rather than a network share. Confirm the project playbooks, config, state and interaction directories exist, and avoid synchronised or redirected folders for the first diagnostic run.

```
$Project = 'C:\IOblendLabs\sensor-lab'  # use your project path
Test-Path $Project
Test-Path (Join-Path $Project 'playbooks')
Test-Path (Join-Path $Project 'config')
Test-Path (Join-Path $Project 'interaction')
```

Expected: each command returns True. If any returns False, correct the Run Parameters path or recreate the missing folder as appropriate, then restart the test. Re-extract with 7-Zip if bundled JARs, executables or Python files are missing.

2.3.3 Step 3: Check Java and Windows environment resolution

Run the following commands in PowerShell. They identify the Java executable visible to the shell and the configured JAVA_HOME value. Compare these with the Java environment used when the IOblend launcher starts Spark.

```
java -version
where.exe java
$env:JAVA_HOME
Get-Command java -ErrorAction SilentlyContinue | Select-Object Source
```

Expected: Java resolves to the intended supported installation. Multiple conflicting Java locations, an invalid JAVA_HOME, or a missing executable are leads for the workstation administrator. The Java used by IOblend may differ from the Java returned by an unrelated shell, so confirm the launcher environment before making changes.

2.3.4 Step 4: Check available RAM and Spark/JVM settings

Confirm that there is enough free memory to start the driver, JVM, local Spark components, Python and Windows. If the developer workstation has limited free RAM, temporarily test without an initial heap override or use the standard profile below. Keep -Xms smaller than the effective maximum heap.

```
$os = Get-CimInstance Win32_OperatingSystem
"Available physical RAM (GB): {0:N1}" -f ($os.FreePhysicalMemory / 1MB)
```

Standard local profile (16 GB+ workstation, provided enough memory is actually available):

```
--conf spark.driver.memory=8g --conf spark.driver.extraJavaOptions=-Xms4g
```

Larger-memory local profile (48 GB+ workstation, provided enough memory is actually available):

```
--conf spark.driver.memory=24g --conf spark.driver.extraJavaOptions=-Xms20g
```

Expected: Java stays running and Spark progresses beyond session initialisation. Do not combine spark.driver.memory=8g with -Xms20g; the initial heap would exceed the driver maximum heap. Restart the Engine between configuration changes.

2.3.5 Step 5: Verify writes and review endpoint protection

Designer and Engine exchange files under the project interaction directory. Test whether your Windows user can create and read a temporary file there. This is a workstation diagnostic only; it does not modify the playbooks.

```
$Project = 'C:\IOblendLabs\sensor-lab'  # use your project path
$TestFile = Join-Path $Project 'interaction\ioblend_write_test.tmp'
'OK' | Set-Content -Path $TestFile
Get-Content $TestFile
Remove-Item $TestFile
```

Expected: the file is written, read back as OK and removed without an access-denied error. If this fails, check permissions, OneDrive/redirected folders, and endpoint-protection rules. A successful manual write does not exclude application-specific Controlled Folder Access restrictions.

Where Microsoft Defender Controlled Folder Access is enabled, the following command can reveal blocked or audited writes around the failed startup:

```
Get-WinEvent -FilterHashtable @{
  LogName   = 'Microsoft-Windows-Windows Defender/Operational'
  Id        = 1123,1124,1127,1128
  StartTime = (Get-Date).AddHours(-2)
} -ErrorAction SilentlyContinue |
  Select-Object TimeCreated, Id, Message
```

An event 1123/1127 is a block; 1124/1128 is an audit event. No events does not rule out other endpoint tools. Where protection software identifies a trusted IOblend, Java or Python component, ask workstation IT to approve the relevant executable or directory using the organisation's normal process rather than disabling protection globally.

2.3.6 Step 6: Test local Spark networking

Spark uses local TCP networking even in local mode. VPNs, virtual adapters, firewall policies or hostname resolution can sometimes interfere with driver communication. On an affected developer workstation only, try forcing Spark to use loopback for one diagnostic run.

```
$env:SPARK_LOCAL_IP = '127.0.0.1'
& 'C:\path\to\ioblend_designer.exe'  # replace with actual path
```

Start the Designer from that same PowerShell window so it inherits SPARK_LOCAL_IP. Alternatively, add the following to the local Spark submit configuration for a separate test (do not change both methods at once):

```
--conf spark.driver.host=127.0.0.1 --conf spark.driver.bindAddress=127.0.0.1
```

Expected: the Spark session starts and the Designer receives execution output. If this resolves the timeout, check Windows hostname resolution, VPN/virtual network adapters and local firewall rules. These loopback settings are for local Developer Edition troubleshooting, not a default for a remote Spark cluster.

2.3.7 Step 7: See whether Java is starting or stopping

Run this command while IOblend is attempting to start Spark, then run it again after the timeout:

```
Get-CimInstance Win32_Process |
  Where-Object { $_.Name -match "^(java|javaw|python|pythonw)(\.exe)?$" } |
  Select-Object Name, ProcessId, ParentProcessId
```

Result	Next action
No Java process at any point	Recheck extracted files, Java resolution, launch permissions and endpoint blocks.
Java appears then vanishes	Capture the Script window and stderr; recheck memory, Java compatibility and JAR availability.
Java persists, but Designer times out	Focus on the interaction directory, local ports, hostname resolution and endpoint restrictions.
Spark starts, but components later fail	Move to Part III 3.8 and investigate the relevant component output.

2.3.8 Step 8: Collect a useful diagnostic bundle

If the local CSV lab still cannot start, collect the evidence below and send it to your internal workstation support or the IOblend technical team. Keep the exact failure timestamp so event logs can be matched to the failed run.

- ☐ Windows version, installed RAM and approximate free RAM at startup.
- ☐ Java version and resolved executable path; active Spark memory settings.
- ☐ Engine version, last successful Script/terminal message and full error.
- ☐ Whether Java was absent, exited early or remained running.
- ☐ Project paths and results of the interaction-directory write test.
- ☐ Relevant Controlled Folder Access/endpoint logs and whether VPN or redirected folders were in use.
- ☐ The result of a loopback-only Spark test, if permitted.

BEFORE SHARING LOGS
Remove credentials, tokens, private-key paths and any sensitive paths or data. Do not send the full credentials.json file. Preserve the error lines and startup timestamps so the investigation remains reproducible.

IOblend

DEVELOPER HANDBOOK / REFERENCE GUIDE

PART III

Pipeline build

Build, run and troubleshoot real pipelines, progressing from a local CSV exercise through S3, streaming, joins and JDBC.

01 Local CSV

02 Logs and playbooks

03 Amazon S3

04 Streaming and lineage

05 JDBC destinations

06 Troubleshooting

DEVELOPER EDITION • NOVEMBER 2026 • V3

PART III | PIPELINE BUILD

Build, run and troubleshoot real pipelines, starting with local CSV files and progressing to S3, streaming, joins and JDBC.

3.1 Lab 1: your first working pipeline

Objective: import six sensor readings from CSV, calculate a summary for each device with Spark SQL, and write the result to a local CSV sink. This is the quickest way to learn the Source → Transformation → Sink workflow without needing network credentials.

3.1.1 Make the input file

Create source_data/sensor_readings.csv in your new project, using these exact headers and sample values:

```
device_id,created_date,temp,heart_rate
A1,2026-11-09 09:01,36.5,70
A1,2026-11-09 09:02,37.2,80
A1,2026-11-09 09:03,36.8,90
B2,2026-11-09 09:01,36.3,65
B2,2026-11-09 09:02,36.9,75
C3,2026-11-09 09:01,37.8,100
```

3.1.2 Create the source

1. Add a Source component from the Designer palette. Name it sensor_source and set Run Order = 1.0.
2. Set Component Type = source_driver, Component Sub-type = file-local-static and Source Type = csv.
3. Set location to your project's source_data directory. Under load_options, use sep = , and header = true.
4. Under Debug Options enable explore, display_table and display_schema; set display_limit = 10. Save the component.

The source reads the CSV and makes its data available to downstream components. The Engine may also add internal source and lineage fields.

3.1.3 Create the SQL transformation

1. Add a Transformation component named sensor_summary, with Run Order = 2.0 and internal SQL transformation subtype.
2. Set the output tableName to sensor_summary_tbl and dependency to sensor_source.
3. Paste the SQL below into sql statement and save. SQL table names must match the registered source/transform tables in your project.

```
SELECT
  device_id,
  COUNT(*) AS readings,
  ROUND(AVG(CAST(heart_rate AS DOUBLE)), 1) AS avg_hr,
  MAX(CAST(temp AS DOUBLE)) AS max_temp
FROM sensor_source
GROUP BY device_id
```



Internal SQL transform properties in the Designer. The screenshot illustrates the editor; use the lab names and SQL above.

3.1.4 Create the local CSV sink

1. Add a Sink component named sensor_output, Run Order = 3.0, Component Sub-type = local-csv.
2. Set dependencies to sensor_summary and location to C:/IOblendLabs/sensor-lab/output (or your own equivalent).
3. Set CSV load_options: sep = , and header = true. Set include_meta_fields = true so you can inspect the generated record-level lineage fields.
4. For this small test you may enable output_single_file. Save the component and save the pipeline.

3.1.5 Run and verify

If Spark does not start on Windows, use the workstation checklist in Part II, Section 2.3 before changing the pipeline.

1. Click Run. The Engine should start its Spark session, load the playbooks and display schemas and sample data in the terminal/interactive output.
2. Use the interactive Continue controls when prompted. Confirm the source has six rows and the aggregated result has three rows.
3. Open output/ and inspect the CSV (or Spark part file). Confirm your business columns match the expected results below; extra ioblend_* metadata columns may be present.

device_id	readings	avg_hr	max_temp
A1	3	80.0	37.2
B2	2	70.0	36.9
C3	1	100.0	37.8

Row order may differ because Spark does not guarantee ordering without ORDER BY.

A SMALL BUT IMPORTANT TEST

Add WHERE CAST(heart_rate AS DOUBLE) >= 75 before GROUP BY and rerun into a clean lab output. The new results should be A1 = 2 readings / avg 85.0, B2 = 1 / avg 75.0, C3 = 1 / avg 100.0. A successful execution is only half the test: check the arithmetic.

3.2 Logs and playbooks: what the Engine uses

3.2.1 Read a run from beginning to end

For a typical local run, the terminal reports the Engine version, initialised Spark session, loaded Run Parameters, playbook paths, schemas, component execution, record counts, checkpoint operations and sink output. Each stage can be inspected while Debug and Run Interactive are enabled.

At this stage	Check
Initialisation	Engine version, Spark Master and which playbook location was loaded.
Source	Record count, expected column names and source-file provenance.
Transformation	SQL registration, output tableName, sample rows and calculation correctness.
Sink	Dependency resolution, location, new-record counts and successful output.
Event management	When enabled, event windows, forwarding, regression, archive and checkpoint messages.

3.2.2 What a playbook JSON actually contains

Designer saves a separate JSON metadata file for each component. The actual aviation project uses a top-level list containing a component object, with keys such as comp_type, name, run_order, comp_sub_type, source_type, conf, dependencies, interactive_shell and x/y canvas positions.

```
[
 {
  "comp_type": "source_driver",
  "name": "sensor_source",
  "run_order": 1.0,
  "comp_sub_type": "file-local-static",
  "source_type": "csv",
  "conf": {
   "location": "<your-project>/source_data",
   "load_options": {"sep": ",", "header": "true"}
  }
 }
]
```

This example is shortened to teach the structure, not a complete runnable playbook. Use Designer to build or edit components; it also writes schema history, debug settings and other operational properties that are not shown here.

3.2.3 What belongs in source control

JSON playbooks and non-secret project templates can be versioned alongside other application code. Keep credentials, access tokens, cloud private keys, developer-specific environment files, execution outputs and generated checkpoints outside broadly shared repositories.

3.3 Lab 2: connect to Amazon S3

IOblend supports S3 as both source and destination. S3 file locations use the s3a:// URI scheme in the playbook. Set up sources and sinks directly in Designer, including the source format, location, checkpoint and output options. All examples here use placeholders, not real account, bucket or credential values.

3.3.1 Prepare a dedicated training location

1. Obtain access to a non-production S3 bucket and two separate prefixes, for example s3://<training-bucket>/sensor-in/ and s3://<training-bucket>/sensor-out/. Set aside a third prefix for checkpoint data if you will test s3-stream.

2. Ensure your workstation/Engine process is authorised for the actions needed by the test: list/read objects in the input prefix, write objects to the output prefix and access its checkpoint prefix if enabled. Use the AWS identity and credential method approved for your environment.
3. Create a small CSV input file under sensor-in/ using the six-row dataset from Lab 1. Do not place output files inside the input prefix.
4. If the AWS CLI is available, test connectivity to the training bucket independently. This checks the AWS side, though it does not on its own prove that the Spark/Hadoop S3A connection has picked up the same credentials.

```
# Example AWS CLI checks - replace all placeholders
aws sts get-caller-identity
aws s3 ls s3://<training-bucket>/sensor-in/
aws s3 cp sensor_readings.csv s3://<training-bucket>/sensor-in/
```

HOW S3 AUTHENTICATION IS SUPPLIED

IOblend's playbook records the destination URI and connector settings; your AWS/Spark runtime provides the necessary S3 permissions and authentication. Use your environment's approved S3A credential provider or identity configuration. Do not put an access key, secret key or session token into the JSON playbook, shared screenshots or a training handout. Confirm the actual credential-loading route with the local Spark setup.

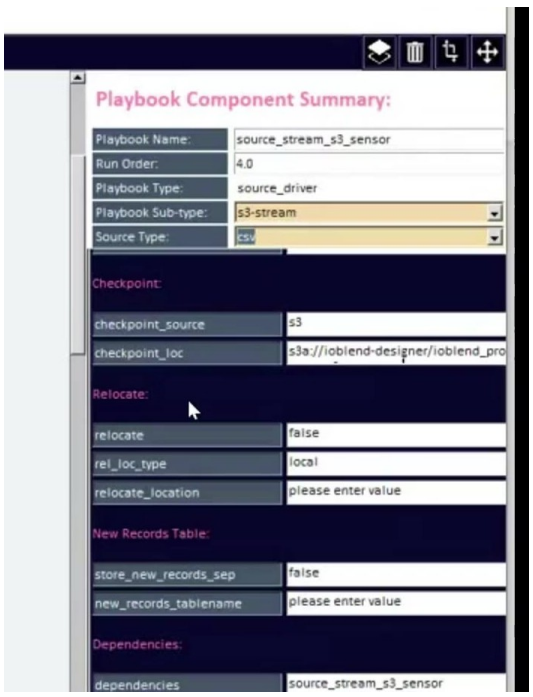
3.3.2 Read S3 using a static source

For a one-off file read, create a source named sensor_s3_static with Component Type = source_driver and Component Sub-type = file-s3-static. Set Source Type = csv, location = s3a://<training-bucket>/sensor-in/, and sep = , / header = true. Put it before sensor_summary and update the SQL source table reference accordingly.

3.3.3 Read new S3 files as a stream

For a streaming experiment, choose source_driver, Component Sub-type = s3-stream, Source Type = csv, and a dedicated S3 input prefix. Configure checkpoint_source = s3 and checkpoint_loc to a separate S3A checkpoint location. Start with managed-record and event regression options off until basic ingestion works.

Property	Example
name	sensor_s3_stream
run_order	1.0
comp_sub_type	s3-stream
source_type	csv
location	s3a://<training-bucket>/sensor-in/
checkpoint_source	s3
checkpoint_loc	s3a://<training-bucket>/checkpoints/sensor-s3-lab/
load_options	sep = , ; header = true
relocate	false for the first experiment



The S3 streaming source exposes checkpoint settings in the Designer. Use your own bucket and checkpoint path.

3.3.4 Write a CSV output to S3

Add a sink with Component Sub-type = s3-csv. The aviation playbook establishes the configuration pattern: destination location, CSV load options, partition keys, output_single_file, include_meta_fields and dependencies.

Sink property	Value for the exercise	Purpose
Component name	sensor_s3_output	Unique destination component.
Run Order	3.0 or next free order	Execute after the SQL transform.
Component Sub-type	s3-csv	Write CSV to S3.
location	s3a://<training-bucket>/sensor-out/	Output prefix.
dependencies	sensor_summary	Consume the transformation result.
include_meta_fields	true	Include generated lineage metadata in the output.
output_single_file	true (small tests)	Single-file option where supported.
load_options	sep = , ; header = true	CSV formatting.

Only set partition-keys when you deliberately want to partition by valid output columns. For a first exercise, leave them unset rather than entering a placeholder string.

```
[
  {
    "comp_type": "sink",
    "name": "sensor_s3_output",
    "run_order": 3.0,
    "comp_sub_type": "s3-csv",
    "conf": {
      "location": "s3a://<training-bucket>/sensor-out/",
      "include_meta_fields": true,
      "output_single_file": true,
      "load_options": {"sep": ",", "header": "true"}
    },
    "dependencies": ["sensor_summary"]
  }
]
```

This is a sanitised, shortened illustration of the sink metadata. Designer should generate the complete playbook JSON; do not use the excerpt as a replacement for a saved component.

3.3.5 Verify the S3 exercise

- ☐ The S3A source reads the expected input rows.
- ☐ The transformation produces the expected three device summaries.
- ☐ The sink writes objects under the output prefix only.
- ☐ You can read the destination CSV and check its result values.
- ☐ Changing include_meta_fields changes whether generated lineage columns appear in the sink.

IF S3 ACCESS FAILS

Start with the exact S3A URI, bucket region, runtime credentials, policy permissions and Spark/Hadoop S3A connector configuration. A successful AWS CLI listing is helpful but cannot substitute for an Engine read/write test. Inspect the first connector error in the Script window.

3.4 Lab 3: streaming and event management

Once a static source and sink work, introduce a fresh streaming source in a copy of the lab. The local file-stream option uses local checkpoint files; s3-stream supports an S3 checkpoint. Keep these setups separate so you can compare behaviour.

3.4.1 A bounded streaming test

1. Create a new input directory stream_in and a separate output directory stream_out. Add one small CSV file containing the original headers.
2. Configure source_driver, Component Sub-type = file-stream, Source Type = csv, a checkpoint_source of local, and a unique checkpoint_loc under checkpoint/.
3. Set Epochs to a small positive number for a bounded test. Epochs = 0 runs continuously until terminated; do not start with an indefinite run.
4. Run the pipeline, inspect the initial source count, then place another CSV file with new rows into stream_in during the next cycle.
5. Record the new-file behaviour, checkpoint log and sink count. Keep every output outside stream_in.

3.4.2 Event and managed-record options

Option	Meaning in the Designer
is_managed	Enable managed-record behaviour based on declared keys.
maintained_keys	Fields identifying records or groups of records.
version_keys	Fields used in the managed-record versioning configuration.
store_new_records_sep	Option to retain new-record results separately.
event_key	Source column used as the event timestamp.
event_time_format	Date/time format used to interpret event_key.
event_window_span_in_secs	Event-time window span in seconds.
latedata_event_window_epocs	Late-data event-window parameter in the saved playbook.
regress_events / apply_archive	Controls for event regression and archive behaviour.

For sensor readings, use created_date as the event_key only if the timestamps in your input match event_time_format. Change one event option at a time and observe event forwarding, archiving and checkpoints in the Engine log.

3.5 Lab 4: lookup joins and lineage

3.5.1 Join a static lookup to the sensor result

Create source_data/device_sites.csv containing three distinct device IDs:

```
device_id,site
A1,North
B2,West
C3,South
```

Add a static CSV source named device_sites. Create a new internal transform that depends on sensor_summary and device_sites. Set an output tableName such as sensor_sites_tbl and use the SQL below, adjusting the upstream table name if your saved transform differs.

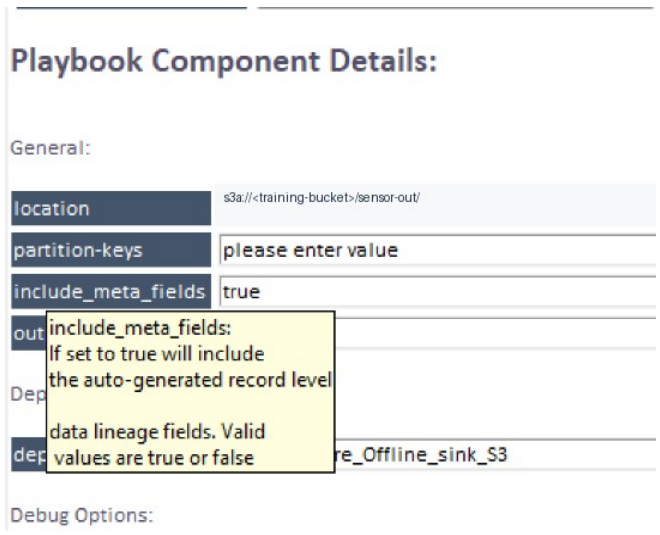
```
SELECT s.*, l.site
FROM sensor_summary_tbl s
LEFT JOIN device_sites l
  ON s.device_id = l.device_id
```

Expect three rows. If the count increases after the join, inspect duplicate values in the lookup key before assuming an Engine error.

3.5.2 Metadata and lineage

Field generated by IOblend	What it tells you
ioblend_source_file	The original source file/path where the record was loaded.
ioblend_created_date	Record creation/processing timestamp associated with the pipeline.
ioblend_dpName	Pipeline identifier.
ioblend_event_key	Configured event-time value when applicable.
ioblend_process_time	Processing time for the record at the current stage.
ioblend_latest_process_time	Latest recorded processing time for the record.
ioblend_rn	Engine row-number field observed in event-archive outputs.

The include_meta_fields option is configured independently on each sink. Set it to true when the target should expose the auto-generated record-level data-lineage fields; set it to false when the target should contain business columns only. This does not require changing your SQL to strip the metadata.



The sink tooltip identifies include_meta_fields as the switch for auto-generated record-level data-lineage fields.

For logical lineage reconstruction, inspect input record metadata, the SQL transformation definition, and the post-transformation records and metadata. Joins and aggregations can be reconstructed through the component relationships and SQL logic if the relevant inputs and audit trail have been retained. Automated end-to-end reverse-lineage visualisation is not currently a built-in feature.

3.6 JDBC database destinations

A JDBC sink is configured through the sink component, the correct driver JAR, the JDBC mapping file and the relevant connector credentials. For example, a pjdbc sink can deliver processed records into Google BigQuery, where the resulting tables can be inspected directly.

1. Use an isolated test database or schema. Confirm which DBMS and JDBC driver version the local Engine supports.
2. Place the required JDBC JARs in the directory specified by Jars Path. Confirm the actual JAR files are present before running.
3. Use the default jdbc_db_sql.json mappings from the installation package. The mapping contains database-specific SQL templates; if a database is not present, have the IOblend team extend it.
4. Create or update the credential configuration through Config > Security config > New credentials. Use the field names appropriate to your connector; keep secret values outside documentation and screenshots.
5. Create a sink with the appropriate JDBC subtype, set dependencies, destination table, maintenance/SCD options as required, and save the playbook.
6. Run in Debug/Interactive mode. Check the Engine messages, inspect the actual destination table, and confirm row counts and expected SCD records if you configured SCD.

CREDENTIAL-FILE STRUCTURE

The training package uses a JSON object with connection-profile entries such as {"dbtable":"<test_table>","user":"<db_user>","password":"<secret>"}. The exact keys depend on the connector. Never check a populated credential file into a shared playbook repository.

3.7 Explore the aviation training project

The Aviation_test_data_by_step_local archive is a richer example of the same principles. It contains component playbooks, config, source data, backup playbooks, generated state/logs, interactive files, and an offline model/report. It is a training project with paths tied to its original workstation; reconfigure the paths before experimenting.

Component / group	Practical point
csrc_stream_raw → transform_strsrc_add_column → sink_raw_stream	The first raw ingestion and intermediate CSV write.
src_OAG_raw_by_step → OAG_by_step_sum_transform	Second stream and aggregated routes/capacity.
geospacial_lookup_src / OAG_by_step_sum_geo_transform	Join airport coordinate data.
carrier_lookup_src / OAG_by_step_sum_carrier_transform	Enrich airline information.
market_lookup_src / OAG_by_step_sum_market_transform	Enrich markets and country names.
Online_Store_sink	CSV output with generated metadata enabled.
Feature_Store_Offline_sink_S3	Example of an S3 CSV sink; not active in the supplied run (Run Order 0).

The sample log shows 22,810 first-source records, 45,620 rows at the second source, 13,589 rows after aggregation and 13,599 after market enrichment. These values are specific to that test run, not expected counts for a fresh project. Follow the component dependencies and SQL to understand what each stage does to the data.

3.8 Troubleshooting and good habits

Symptom	Check first
Missing DLL, JAR or module	Re-extract the complete package with 7-Zip; confirm the extracted files exist.
Designer does not open	Start from the extracted local directory; check Script window and environment setup.
Run Parameters do not load	Check Config > Project Run Params and verify the config/playbooks path.
Spark or Java server connection error	Confirm Python/Java/Spark setup, local firewall/endpoint trusted components and compatible JARs.
S3 access denied	Verify S3A bucket URI, Engine identity/credentials, IAM actions and bucket region.
Source returns no records	Check input path, delimiter/header, file existence and checkpoint already marking files processed.
SQL table not found	Check exact source/output table names, registered tables and dependencies.
Unexpected duplicate rows	Check re-read input directories, duplicate lookup keys and event windows.
JDBC fails	Check JAR location, connection configuration, mapping and database permissions.
Interactive execution pauses	Read the terminal: it may be waiting for Continue/Reset/Quit or interaction file activity.

Before you share a project or ask for help

- ☐ Record the Engine version, OS and the name of the failing component.
- ☐ Keep a small reproducible input CSV and the expected output.
- ☐ Share only the relevant JSON playbooks and non-secret Run Parameters.

- ☐ Remove credentials, private-key paths, tokens and personal machine paths from screenshots/logs.
- ☐ Include the error and one preceding successful component from the log.

Before you move beyond the first lab

- ☐ Run Parameters are saved and correct.
- ☐ Source count matches the file you supplied.
- ☐ Transform output matches independently checked results.
- ☐ Sink location is separate from source inputs.
- ☐ You understand which generated metadata fields are shown or hidden at the sink.
- ☐ S3 paths and permissions are isolated to test prefixes.
- ☐ You know where the project JSON playbooks and Engine logs are stored.

IOblend

DEVELOPER HANDBOOK / REFERENCE GUIDE

REFERENCE

Appendices

Keep copyable examples, configuration patterns and developer reference terms close at hand.

01
Starter data

02
Configuration template

03
Developer terms

REFERENCE APPENDICES

Copyable starter data, configuration examples and terms to keep beside you while working in the Designer.

Appendix A. Copyable starter data and calculations

The six-row CSV and the three expected aggregated results appear in Part III, Lab 1, so the handbook is usable without another download. A companion practice ZIP can provide those files if preferred. The SQL assumes Spark SQL semantics; source fields imported from CSV may require explicit CAST for calculations.

Appendix B. Minimal configuration template

The following JSON is an example representation of the key project settings; Designer will write the full run configuration with release-specific fields and paths. Do not substitute this shortened example for the saved Run Parameters file.

```
{
  "Pipeline Location": "C:/IOblendLabs/sensor-lab/playbooks",
  "Transform Ext": "C:/IOblendLabs/sensor-lab/transforms",
  "Epochs": 1,
  "Debug": true,
  "PipeLine Name": "sensor_lab_01",
  "State Directory": "C:/IOblendLabs/sensor-lab/state",
  "Run Interactive": true,
  "File System": "local",
  "Spark Master": "local",
  "interaction_by_file": true,
  "idf_location": "C:/IOblendLabs/sensor-lab/interaction",
  "Run Parameters Path": "C:/IOblendLabs/sensor-lab/config",
  "ssh": "local"
}
```

Appendix C. Developer reference terms

Term	What it means in IOblend
Source driver	A playbook component that reads from a file, streaming location or database.
Internal transform	A SQL transformation defined in the playbook.
External transform	A Python transformation loaded from the configured external path.
Sink	A component writing data to a local, cloud or database destination.
Dependency	Component(s) that the current playbook relies upon.
Run Order	Numeric order used in the Designer playbook.
State directory	Project location for the Engine’s internal processing state.
Checkpoint	Source progress tracking used by certain stream configurations.
S3A URI	Spark/Hadoop form of an Amazon S3 object location: s3a://bucket/prefix.
include_meta_fields	Sink switch to include or omit generated record-level lineage fields.

IOblend documentation and downloads: https://ioblend.com/documentation/

Edition notes: Covers Developer Edition installation for Windows and macOS, project Run Parameters and the playbook conventions in the aviation training project. Local exercises use instructional data; use the fields and values appropriate to your installed release.

Final quick-reference checks

Question	Answer / place to look
Did the install extract completely?	7-Zip test and extracted ioblend_designer folder.
Is the correct project loaded?	Config > Project Run Params; check Pipeline Location and PipeLine Name.
What ran and how many rows?	Interactive Script window; component schemas, samples and counts.
Where is the logic stored?	One JSON playbook per component under playbooks/.
Where is lineage exposed?	Sink property include_meta_fields = true / false.
Can the Engine access S3?	Test the Engine with an s3a:// training prefix and authorised runtime credentials.

IOblend

PARTNER & CLIENT TRAINING PROGRAMME

Advanced capabilities

Further learning aligned with the training programme

01 EVENT-TIME INTELLIGENCE

- Chained aggregations
- Windowed & decayed metrics
- Event regressions & replay
- Late-arriving / out-of-order data

02 STATE, QUALITY & RECOVERY

- Checkpointing & state management
- CDC, deduplication & upserts
- SCD Type I/II & MDM
- Record-level lineage & error handling

03 OPEN INTEGRATION & CUSTOM LOGIC

- APIs, databases, files & event streams
- Cloud, on-prem & hybrid systems
- SQL and custom Python transformations
- Portable JSON playbooks

04 AI EMBEDDED IN THE PIPELINE

- Invoke Python-based AI models or agents
- Extract, classify, enrich & infer
- Validate AI outputs against business rules
- Govern and route results downstream

SOURCE › TRANSFORM › STATE › AI › GOVERN › DELIVER

Applied as advanced, use-case-led labs tailored to each partner or client environment.