2.11 - Operational Test Data with DPLs

Operational Test Data with DPLs — Appvance IQ documentation.

- Operational Test Data with DPLs
- AIQ DPL Types
- Configuring Appvance DPLs
- Providing Data to Distributed Tests
- Configure Dynamic DPL Data
- Array Generator DPL (Test Scenario, Functional Test)
- WhiteBoard DPL Usage in AIQ
- Use Scenario-Wide DPL Data
- Use Hash DPL Data in Scenario Editor Tests
- Configure JavaScript DPL Data
- Use DPL Data Sources in AIQ
- Communication Intra-Steps Using TheLocker
- The RSC Resource
- Creating Your Own DPL Using JavaScript in AIQ.
- Specific Types of DPLs
- Use DPL Data in Scenario Editor Tests
- Synthetic DPL Data in AIQ
- Encrypt/Decrypt DPL
- Operational Test Data FAQ

Operational Test Data with DPLs

Contents

- Appvance DPL Types
- Configuring Appvance DPLs
- Providing Data to Distributed Tests
- Dynamic DPL
- Array Generator DPL
- WhiteBoard DPL
- Scenario Wide DPL
- Hash DPL
- JavaScript DPL
- Using the DPLs
- Communication Intra-Steps Using TheLocker
- The RSC Resource
- Creating Your Own DPL
- Specific Types of DPLs
- DPL Usage with Appvance Designer Script
- Synthetic DPL
- Encrypt/Decrypt DPL
- Operation Test Data FAQ

Appvance IQ's Data Production Library (DPL) is a powerful system to deliver data to tests at runtime. For instance, a test of an online registration service stores name and password values in a database and uses a DPL to access the database as the test operates. In a second instance, a test of a reporting service uses a DPL to access a comma-separated-value (CSV) file of acceptable responses to validate service responses.

Software developers, QA technicians, and IT managers with a variety of technical skills use Appvance IQ. It is important to provide DPLs that are ready to use and require no programming

skills. But also offer a framework for building your own DPLs. Appvance Unified Test

Appvance IQ (AIQ) Data Product Libraries (DPLs) delivers operational data to tests at test run-time. For instance, a recorded Appvance Designer Script or Selenium test of a Web-based log-in page retrieves the user name and password values from a comma-separated value (CSV) data file as AIQ executes the test. Tests can use DPL provided data not only as input values but also to validate the application's function.

The AIQ DPL system provides a simple and easy way to data-enabled tests. A DPL does not require programming or scripting.

Last updated: April 11, 2024

If you have any feedback on this topic, please contact us at documentation@appvance.ai

AIQ DPL Types

Recommended guidance

DPLs provide operational test data for AIQ scenarios and services. Choose the DPL type based on data ownership, repeatability, volume, security, and whether values are static, generated, or retrieved from an external system.

Common DPL considerations

- Static data for repeatable regression tests.
- CSV or file-backed data for controlled row-based execution.
- Dynamic or generated data for unique values per run.
- Service-backed data when the test needs values from an API or system of record.
- Secure handling for credentials, personal data, or customer-sensitive values.

Selection checklist

1. Define whether the scenario needs unique, reusable, random, or environment-specific data.
2. Confirm how data is reset or cleaned up after execution.
3. Document which scenario steps consume each DPL value.
4. Do not store secrets or production customer data in unsecured DPL sources.

Last updated: October 6, 2026

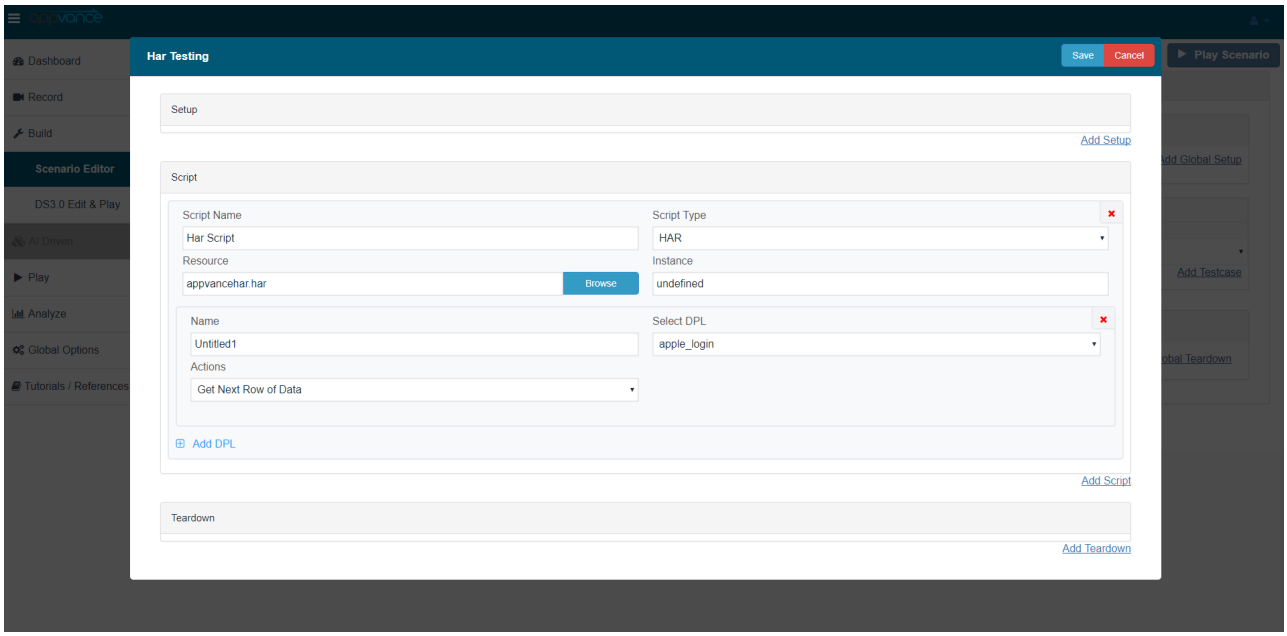
Configuring Appvance DPLs

AIQ implements a Data Production Library (DPL) as an extensible framework to provide operational test data to test script runners at test runtime. The DPL framework is general in the sense that it can get data from a variety of sources, including a relational database, comma-separated-value file, and data generating objects. A DPL also provides a safe way to store data between test steps. AIQ comes with pre-built DPLs and makes it easy for software developers to add their own custom DPLs.

DPLs work in conjunction with script runners at test runtime. Test scenarios define the DPL and script runner interaction. The AIQ Tutorials give instructions on using the pre-built DPLs.

AIQ users configure Test Use Cases in the Appvance Scenario Builder to use operational test data from a DPL. To see this in action follow these steps:

1. Start AIQ
2. Click Build from the Start Screen. The Scenario Builder opens with a blank page.
3. Click File, then New to create a new empty scenario. AIQ sets the default name of the new scenario to New-Scenario.
4. In the Info tab, which is opened by default, select Functional Test for this example.
5. Click the Resources tab.
6. Click Add Data Source.
7. Choose "HashDPL - Reads CSV" in the DPL Type drop-down menu.
8. Click Browse to select a CSV data file.
9. Provide a name for the DPL in the Name box.
10. Click Test Cases on the left-hand side.
11. Under Iteration Name: Iteration 1, click Add a Test Case Execution
12. Provide a Test Case Name at the top of the pop-up window.
13. Select Type Selenium, provide a Script Name and then click Browse to find your appropriate Selenese file.
14. Click Add DPL and under Select DPL choose the DPL you created earlier.
15. Choose Get Next Row of Data from the Actions menu.



16. Provide a name for the DPL action and then click Save at the top.
17. Click File at the top and then Save and choose a location to save the Test Scenario
    This configuration tells the AIQ Selenium script runner to look into the Selenium test and replaces values in the test with values from the data.csv file.
    The DPL provides data from the data.csv file as A IQ runs the Selenium script, at test run time.
    The DPL uses the first row in the CSV file to identify field mapping names. Insert these mapping names into the Selenium file where you want AIQ to inject the test data from the CSV file.
    After the mapping name row comes the rows of the data file containing the actual data.
    The DPL Action determines the row of data AIQ provides to the test.
    In a load test, setting the DPL Action to Get Next Row of Data tells AIQ to advance to the next row of data.
    When the DPL reaches the bottom of the table, the DPL starts at the top.

Sheets | Charts | Smar

	A	B	C	D	E
1	name	password	product	validate	
2	bert	biz	1000	1000	
3	bert	biz	1001	1001	
4	bert	biz	1002	1002	
5	bert	biz	1003	1003	
6	bert	biz	1004	1004	
7	bert	biz	1005	1005	
8	bert	biz	1006	1006	
9	bert	biz	1007	1007	
10					
11					

The Selenium, SoapUI, and Command-Line script runners provide support for data mapping from DPLs.

Script runners for Appvance Designer Script, Java, Jython, JRuby, VB, .NET, Groovy, and the other support scripting languages receive DPL data in a dictionary structure (map).

For TestNG engine to work with DPLs, it is necessary to need to define the @DataProvider annotation inside the TestNG, an AIQ will override that method providing the DPL values as arguments for the @Test method:

```
public class TestNG {
@DataProvider(name = "DPLDataProvider")
public Object[][] ValidDataProvider3() throws Exception {
return new Object[][] {
{'C', 65}, {'c', 97},
{'D', 66}, {'d', 98}
};
}
@Test(dataProvider = "DPLDataProvider")
public void basicTest(final char character, final int ascii) throws Exception {
System.out.println("character " + character + " ascii " + ascii);
}
}
```

In this case, you need to use CSV DPL in your Scenario. AIQ will do the casting of the arguments to the ones expected in the @Test method.

If you want to use HashMap instead (sending arguments as map dictionaries) then you need to add an AIQ provided annotation @TestNGParams (tools.jar):

```
public class TestNG {
@DataProvider(name = "DPLDataProvider")
public Object[][] ValidDataProvider3() throws Exception {
return new Object[][] {
{'C', 65}, {'c', 97},
{'D', 66}, {'d', 98}
};
}
@Test(dataProvider = "DPLDataProvider")
@TestNGParams(params = {"character","ascii"})
public void basicTest(final char character, final int ascii) throws Exception {
System.out.println("character " + character + " ascii " + ascii);
}
}
```

DPLs have the ability to share data from one test to the next. For example, TheLocker DPL takes the results from a test step authored in SoapUI as input to a second test step authored in Appvance Designer Script.

SoapUI support was removed in release 5.2.0 of AIQ.

The above example shows how to use data from a CSV file. DPLs work in the same way with data from a Relational Database (RDBMS DPL, RDBMS Map DPL) and other data-generating objects.

Last updated: July 25, 2024

If you have any feedback on this topic, please contact us at documentation@appvance.ai

Providing Data to Distributed Tests

AIQ is a distributed test system. Multiple virtual users run in a TestNode and tests run across one or more TestNodes. Tests often need unique data for each virtual user. For example, a test of a banking system needs a separate and unique customer number for each virtual user running in the test. AIQ offers 3 solutions to meet your testing needs.

Common Test Data, No Coordination

AIQ's CSV DPL sends a copy of a single CSV file to each TestNode running a test. Each virtual user uses data in the CSV file. Within a TestNode each user shares the CSV file and shares a single pointer to the row of data in the CSV file.

CSV files are distributed to each TestNode

To use this configuration, use the Appvance Scenario Editor to configure a Test Use Case that uses a CSV DPL /Hash DPL. Set the DPL Action setting to Get Next Row of Data. When you run the test AIQ will send a copy of the defined CSV file to each TestNode. AIQ creates a single DPL for all virtual users in the TestNode. The DPL has a raw pointer. The pointer is initially set to the first row of data. The DPL reads the first row of data and injects the data fields into the test script. The DPL moves the pointer to the next row of data. At the last row of data, the DPL moves the pointer to the first row of data.

Unique Provided Data

Many times the test architecture requires each virtual user to use unique data. For example, when testing a banking system each virtual user needs to log in to the banking application using its own user-id and password. AIQ provides the Unique DPL to provide each virtual user's DPL with its own data. The data comes from a single CSV file. Each virtual user uses its local DPL to communicate over a TCP/IP request to a single Unique DPL service. The Unique DPL reads from a single CSV file.

Unique data is distributed to each virtual user

The communication overhead for Unique DPL is minimal. Appvance staged tests up to 100,000 virtual users and observed less than 1% network bandwidth overhead and less than 20-millisecond wait times when using UniqueDPL.

Running a Unique DPL scenario using a CSV file which has duplicate rows

Consider the following scenario to understand the unique DPL functionality.

Suppose the user is creating an Appvance Designer script with DPL enabled. The created DPL contains duplicate values for the field that was data enabled and considers the scenario for 2 executions and 2 users and it should have multiple test Nodes. The Unique DPL works in such a way that it will not validate whether the CSV file has repeated rows or not, that is, it does not check for the content of the rows. In AIQ, the Unique DPL functionality is that it will make sure that among all running users in all running TestNodes in a single execution, each row of the CSV data file will be used only once until all rows available in the DPL are used.

Note: Please make sure the CSV file has unique data rather than duplicated rows.

Transactional Data Provided To Each Virtual User

AIQ's RDBMS DPL enables tests to receive operational test data from a relational database. For example, configure a MySQL instance to serve transactional data to the virtual users. One Appvance customer stored invoice data for a test of an Enterprise Resource Planning (ERP) system. A stored procedure (StoredProc) in MySQL provides invoice data to the virtual users, including part numbers, part costs, quantities, and shipping details. The StoredProc enables each virtual user to receive its own unique data from the common data store.

Transactional data is distributed to each virtual user

The above example showed how to use data from a CSV file. DPLs work in the same way with data from a Relational Database (RDBMS) and other data-generating objects. See the DPL Recipes section for details.

If you have any feedback on this topic, please contact us at documentation@appvance.ai

Last updated: October 15, 2026

Configure Dynamic DPL Data

Configure Dynamic DPL Data

Use a Dynamic DPL when a Scenario Editor test needs generated values at runtime, such as random numbers, generated strings, timestamps, or values within a defined range. Dynamic DPLs are useful when the test data does not need to come from a fixed file or database source.

When to use it

- Generate unique values during each execution.
- Avoid hard-coded test data in Scenario Editor steps.
- Keep data rules in one reusable DPL instead of duplicating logic across scenarios.

Implementation notes

1. Define the dynamic data rule in the DPL configuration.
2. Map the DPL output to the Scenario Editor input that needs the generated value.
3. Run a short validation execution and confirm the generated values appear in the output or reports.

Keep generated values within realistic ranges for the application under test. If a test requires a stable, repeatable data set, use a static source DPL instead.

Last updated: September 18, 2026

Array Generator DPL (Test Scenario, Functional Test)

To run Java code with array arguments in Appvance, use the **Array Generator DPL**. To execute Java code that has a `main` method in Scenario Editor, select the **Array Generator DPL** type and `String[]` from the **Actions** dropdown. Add the argument name and pass a value by clicking the **Add Val** link.

The main method receives the array from the generator.

With the **Array Generator DPL**, you can send an array of values as a single parameter containing multiple entries, for example a String array:

```
[ "string1", "string2", "string3" ]
```

This allows you to provide an input parameter for a method that requires an array-like parameter:

```
public void myMethod(String[] args) {
}
```

You can also use Object arrays. Example:

```
[
  { name: 'Honda', models: [
      { name: 'Accord', features: ['2dr', '4dr'] },
      { name: 'CRV', features: ['2dr', 'Hatchback'] },
      { name: 'Pilot', features: ['base', 'superDuper'] }
  ] },
  { name: 'Toyota', models: [
      { name: 'Prius', features: ['green', 'superGreen'] },
      { name: 'Camry', features: ['sporty', 'square'] },
      { name: 'Corolla', features: ['cheap', 'superFly'] }
  ] }
]
```

Types available for arrays: `Object`, `String`, `Integer`, `int`, `Byte`, `byte`, `Double`, `double`, `Short`, `short`, `Float`, `float`, `Character`, `Boolean`.

Sample Java Code with an int Array

```
package com.package1;


import java.util.Arrays;


public class Array {
    public static void main(String[] args) throws Exception {
    }


    public void printArray(String message, int array[]) {
        Arrays.sort(array);
        System.out.println(message + ": [length: " + array.length + "]");
        for (int i = 0; i < array.length; i++) {
            if (i != 0) {
                System.out.print(", ");
            }
            System.out.print(array[i]);
        }
        System.out.println();
        int index = Arrays.binarySearch(array, 2);
        System.out.println("Found 2 @ " + index);
    }
}
```

Create a **.jar** file from the Java code above. Go to Scenario Editor and select the **Functional Test** test type. Add a test case using the **Java Test** type and browse to the `.jar` file you created. The code above takes two arguments: the first is a `String`, and the second is an `int` array. Pass the arguments in the same order in Appvance so the scenario runs successfully.

Two arguments using array generator

Set the log level to 100% in the Options panel. Save and play the scenario in a Controller window.

Output:

```
Two arguments using array generator
```

If you have any feedback on this topic, please contact us at documentation@appvance.ai

Last updated: October 15, 2026

WhiteBoard DPL Usage in AIQ

Use this article when test data must be shared or manipulated through a WhiteBoard DPL workflow.

Validation workflow

1. Confirm the test needs WhiteBoard-style shared or generated data rather than a static CSV.
2. Validate produced values across multiple iterations and users.
3. Confirm cleanup or reset behavior before using the DPL in a repeatable test run.

Last updated: September 29, 2026

Use Scenario-Wide DPL Data

Use this article when a Scenario Editor test needs one DPL data source available across multiple steps or actions in the same scenario.

Validation workflow

1. Choose scenario-wide DPL only when multiple steps need the same data context.
2. Validate row selection and iteration behavior with a small run before scaling.
3. Confirm logs and reports expose enough row context for debugging without exposing sensitive values.

Last updated: September 30, 2026

Use Hash DPL Data in Scenario Editor Tests

Use this article when a Scenario Editor test needs key-value style test data supplied through a Hash DPL.

Validation workflow

1. Define the key field and confirm every required test case has a matching value.
2. Reference the Hash DPL from the Scenario Editor test and validate both a matching-key and a missing-key case.
3. Keep sensitive values out of plain test data or mask them before logging results.

Last updated: September 30, 2026

Configure JavaScript DPL Data

Use a JavaScript DPL when test data needs lightweight transformation before it is consumed by a Scenario Editor test. Keep the script focused on data preparation; avoid placing test flow, browser automation, credentials, or environment-specific paths in the DPL.

Recommended use

- Normalize or format values before sending them to the application under test.
- Combine fields from an approved data source into a value required by a step.
- Generate simple derived values that are safe to calculate at runtime.

Validation checklist

1. Confirm the JavaScript returns the expected value for a small sample.
2. Handle empty or malformed input values predictably.
3. Verify the DPL mapping in a short Scenario Editor execution before scaling the test.

Do not embed secrets in the DPL script. Store credentials and environment-specific settings in approved configuration locations.

Last updated: September 18, 2026

Use DPL Data Sources in AIQ

Use DPL Data Sources in AIQ

DPLs provide reusable data to AIQ tests. They keep test data and data-generation rules separate from Scenario Editor steps so scenarios remain easier to maintain.

Basic workflow

1. Create or select the DPL that represents the required data source or generated data rule.
2. Map the needed DPL fields to the Scenario Editor inputs.
3. Choose the execution behavior, such as using the next available row or reusing a value.
4. Run a short validation execution and review the output before using the DPL in a larger run.

Good practices

- Use clear field names that match the business meaning of the data.
- Keep credentials and environment-specific values out of the DPL article and scenario content.
- Document data assumptions, such as required formats, uniqueness, and reset behavior.

Last updated: September 18, 2026

Communication Intra-Steps Using TheLocker

If you have any feedback on this topic, please contact us at documentation@appvance.ai

Data flow and scope

TheLocker is the hand-off point for values produced by one tool or step and consumed by another. Define the value name and expected format before the producing step runs, then retrieve and validate it in the consuming step. Keep sensitive values scoped to the test and avoid logging secrets.

Last updated: October 15, 2026

The RSC Resource

When working with distributed testing nodes, tests may require additional resources such as CSV files, data files, or other external files. These resources must be transferred to the remote testing node before the test executes.

Because the resource may be stored at a different location on the remote node, a path that was hardcoded when the test was created may no longer be valid. This can cause the test to fail because the required file cannot be found.

Appvance IQ provides the **RSC DPL** to solve this problem. Resources can be declared in the test's `<resources>` section, and the RSC DPL can then be used to retrieve the actual path of a resource at runtime, regardless of where the resource is stored on the testing node.

Declaring Resources

Resources required by the test, including resources used by other DPLs, should be declared in the `<resources>` section.

For example, suppose a `HashDPL` requires a CSV file. The CSV file can exist anywhere on the local machine, but it must be included in the resources section so that Appvance IQ can transfer it to the remote testing node.

```
<resources>
    <data path="c:\mydata\data.csv"/>
</resources>
```

The `<data>` element represents a file resource.

Resources are assigned an index based on their order in the `<resources>` section. The first resource has index **0**.

For example:

```
<resources>
    <data path="c:\mydata\data.csv"/>
    <data path="c:\mydata\users.csv"/>
</resources>
```

In this example:

- `data.csv` is resource **0**
- `users.csv` is resource **1**

Using RSC with another DPL

Once a resource has been declared, the **RSC DPL** can be used to obtain its actual runtime path.

For example, if a `HashDPL` requires the path to a CSV file:

```
<DataSources>
    <dpl name="mydpl" type="HashDPL">
        <argument name="file"
                  dpl="rsc"
                  value="getDataByIndex"
                  index="0"/>
    </dpl>
</DataSources>
```

Here, the `file` argument of the `HashDPL` receives its value from the RSC DPL.

The following attributes are used:

Attribute	Description
`dpl="rsc"`	Specifies that the RSC DPL is being used.
`value="getDataByIndex"`	Specifies that the resource path should be retrieved by its index.
`index="0"`	Identifies the resource declared at index 0 in the `<resources>` section.

The value returned by RSC is the **actual path of the resource on the testing node**. This means the test does not need to know or depend on the path used by the remote node.

Using RSC in Test Steps

RSC can also be used directly in test step definitions.

For example, a Jython test step may require the path of a resource:

```
<run name="CheckInventory"
     module="JythonModule"
     testclass="testgen4web"
     method="run"
     langtype="jython">
    <argument name="frozenfile"
              dpl="rsc"
              value="getDataByIndex"
              index="0"/>
    <argument name="ownerinfo"
              dpl="values"
              value="getData"/>
</run>
```

In this example, the `frozenfile` argument receives the runtime path of resource **0**.

The test step can therefore access the file without knowing where Appvance IQ stored the resource on the remote testing node.

Recommended Usage

Use the following approach when a test depends on an external file:

1. Add the file to the `<resources>` section.
2. Determine its resource index based on its position in the section. Remember that indexing starts at **0**.
3. Use the RSC DPL with `getDataByIndex` whenever the file path is required.
4. Avoid hardcoding paths to resources that will be used on distributed testing nodes.

This approach allows the same test to run reliably across different testing nodes and environments without requiring changes to file paths.

Summary

The RSC DPL provides a runtime-safe way to access resources transferred to distributed testing nodes. Instead of relying on a hardcoded local or remote path, the test references the resource by its index in the `<resources>` section.

For example:

```
<resources>
    <data path="c:\mydata\data.csv"/>
</resources>
```

The resource can then be referenced anywhere a file path is required:

```
<argument name="file"
          dpl="rsc"
          value="getDataByIndex"
          index="0"/>
```

The RSC DPL resolves the resource to its actual location on the testing node at runtime.

Creating Your Own DPL Using JavaScript in AIQ.

Creating your own DPL requires Java programming skills and a familiarity with the DPL interface definition. It involves a two-step process: first, you need to implement the DPL main class; second, you need to define a new DPL type for Appvance IQ.

REMEMBER: This is only one of the many ways of creating DPLs (simulated data that can be used for Scripts, Scenarios, and more). If you want to learn more about how to create DPLs and integrate them, visit our LMS and watch these videos:

- https://academy.appvance.ai/mod/resource/view.php?id=211
- https://academy.appvance.ai/mod/resource/view.php?id=206

Also, if you are interested in reading external data (Excel, CSV, and others):

- https://academy.appvance.ai/mod/resource/view.php?id=206

The DPL Java class implements the standard DPL methods including getNextData, getDataByIndex, and more. It needs to implement the com.Appvance.tool.dpl.DataProduction interface. Here is an example:

```
import com.Appvance.tool.dpl.DataProduction;
public class SimpleDPL implements DataProduction {
public SimpleDPL() { }
public void setup(String data) {
System.out.println("This is a setup using data "+data);
}
public Object getData() { return null; }
public Object getRandomData() { return "RANDOM DATA"; }
public Object getNextData() { return "Simple data: getNextData"; }
public Object getDataByIndex(int index) {
return "Simple data: get data by index "+index;
}
public void tearDown() { }
```

```
}
```

Once you define and compile a custom DPL packaged into a .jar file, you should use the <DataSources> in a TScenario document to identify your new DPL type.

If you have any feedback on this topic, please contact us at documentation@appvance.ai

Last updated: October 15, 2026

Specific Types of DPLs

Using the DPL categories

Choose a DPL type based on how its data is produced and consumed at runtime. Confirm the DPL’s schema, mutability, and distribution behavior in the related Operational Test Data documentation before referencing it from a scenario. This page had no technical body content; the overview now directs customers to the supported DPL workflow without inventing additional product behavior.

Last updated: October 15, 2026

Use DPL Data in Scenario Editor Tests

Use DPL Data in Scenario Editor Tests

Use DPL data sources to provide repeatable test data to Scenario Editor executions. Store source files and database connection details in approved locations and avoid embedding credentials in scripts.

1. Create or select the required data source.

2. Map the needed fields to the relevant test inputs.

3. Configure execution behavior, such as advancing to the next available data row.

4. Run a small validation execution before scaling the test.

Refer to the release-specific DPL documentation for supported source types and configuration fields.

Last updated: September 14, 2026

Synthetic DPL Data in AIQ

Recommended guidance

Use Synthetic DPL data when AIQ tests need generated values that are safe, repeatable enough for validation, and independent from production customer records. Synthetic data should match the format and constraints of the application under test.

Good use cases

- Generating names, IDs, addresses, emails, or other non-sensitive values.
- Creating unique values per run to avoid duplicate-data conflicts.
- Testing validation rules without exposing real customer data.
- Supporting performance or regression runs that need large data volume.

Quality checks

1. Confirm generated values match field length, format, and business rules.
2. Document randomness, uniqueness, and cleanup expectations.
3. Do not use synthetic values that accidentally resemble real sensitive data.
4. Validate failures against both scenario logic and generated data quality.

Last updated: October 6, 2026

Encrypt/Decrypt DPL

The Data Production Libraries (DPLs) that are supported in AIQ are CSV files in a readable format. This allows AIQ to encrypt the contents of the file so that the file is not in a readable format. It is encrypted with a password so that can be decrypted during run time from the Scenario Editor.

Support for Encrypted DPL was deprecated in release 5.3.0 of AIQ. Scripts using encrypted DPL files will continue to run, but descriptions will not be possible.
This option has been removed from the **Test Data** page. You can migrate encrypted DPL to HASH DPL files before upgrading to release 5.3.0.

See Encrypt/Decrypt DPL in Scenario Editor for more information.

Considerations

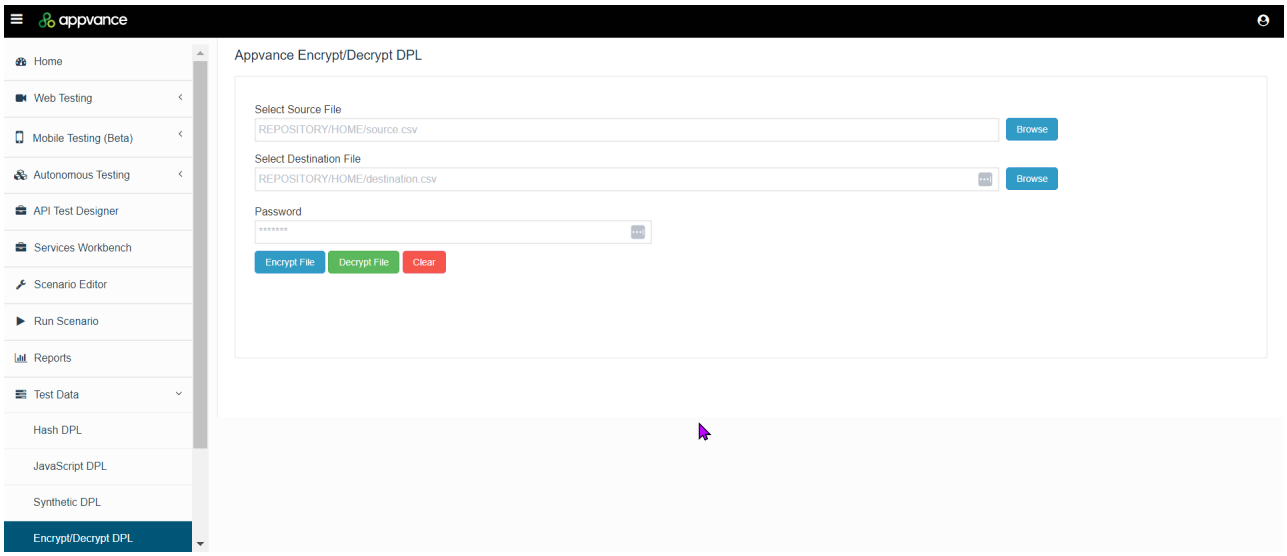
- This feature is implemented using the SHA-256 Algorithm as the secure option.
- For this encryption to work as expected, it is required to have Java 8 build 251 installed and configured to use for AIQ.
- For any other build that is lesser than Java 8 build 251, there are certain files that need to be added to the JDK build. See Fixing java.security.InvalidKeyException: Illegal key size exception.
- However, it is strongly recommended that you use Java 8 build 251 so that you do not have to use other jars.
- Since JDK is paid version as of April 2020, there is support for OpenJDK as well, The file encryption works using OpenJDK with version 1.8.0_252.

Video explaining encryption-decryption of DPL

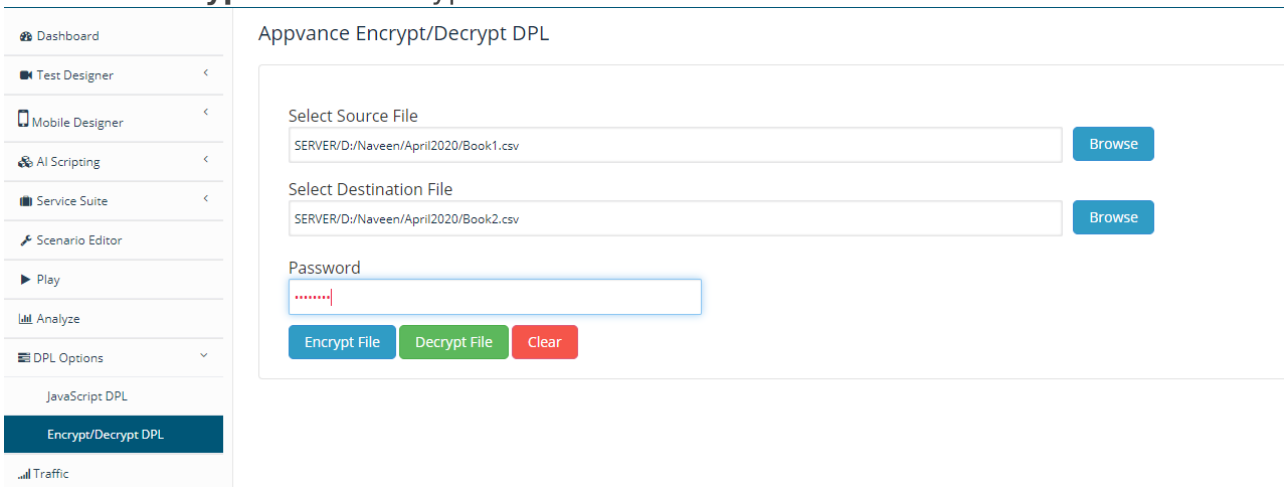
Encrypting a DPL

This option is available for users and administrators.

1. From AIQ, navigate to **Test Data** > **Encrypt/Decrypt DPL**.

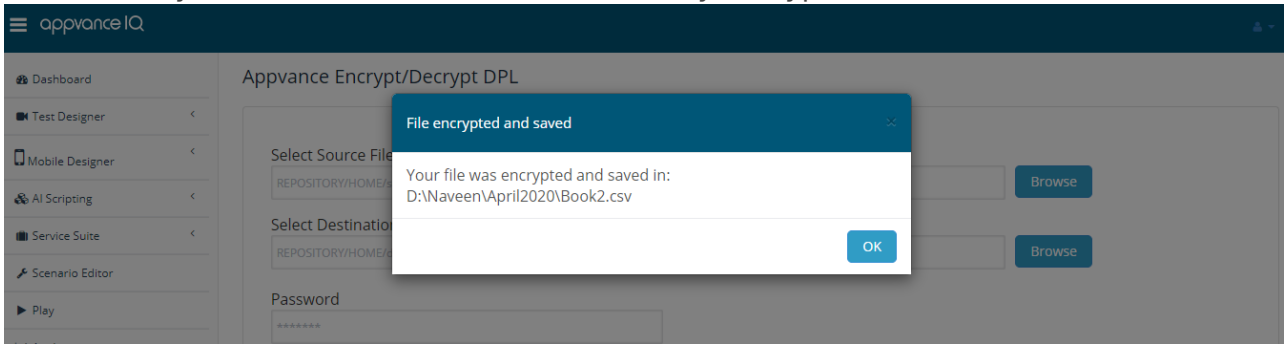


2. Click **Browse** to select the **Source File** from the local directory or from the configured repository.
   NOTE: As of now this works only for HASH DPL and not for JS DPL yet.
3. Select the **Destination File**. Provide a new name to the CSV file used for decrypting the data. If you choose an existing file, the file will be replaced with new contents.
4. Provide a valid **Password**.
5. Click the **Encrypt File** to encrypt the file.

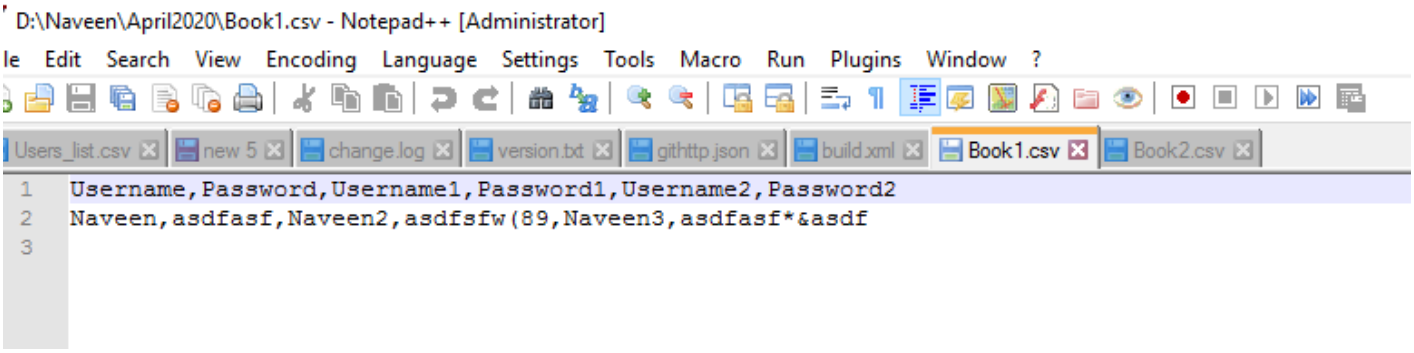


Make sure the password is remembered, because the same password should be provided to decrypt the password when running from the scenario as well.

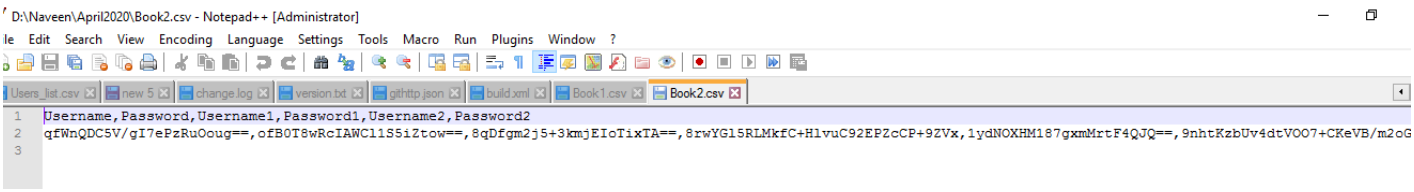
6. AIQ notifies you that the file has been successfully encrypted and saved.



Before Encryption

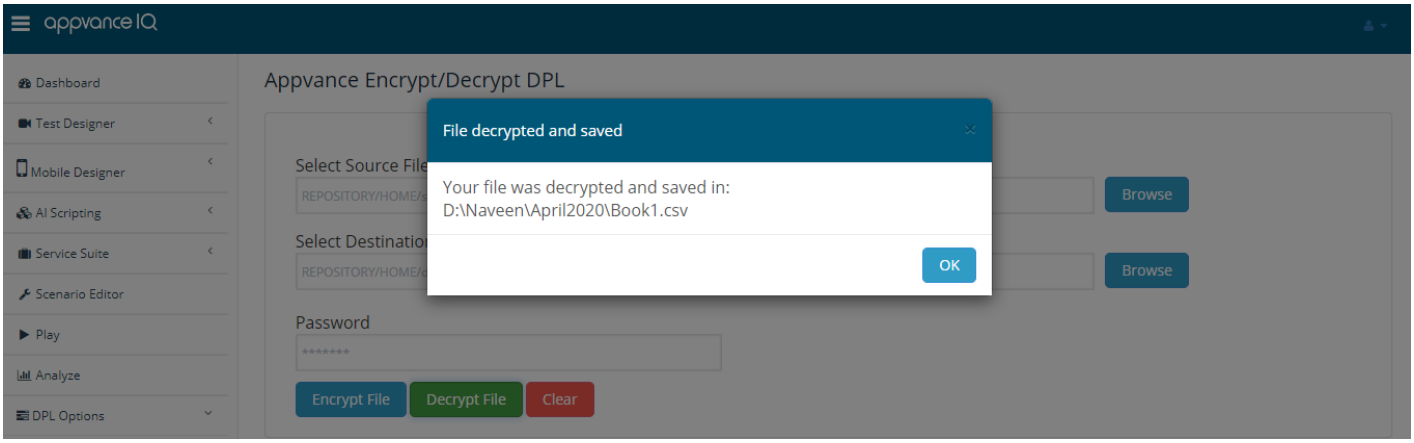


After Encryption



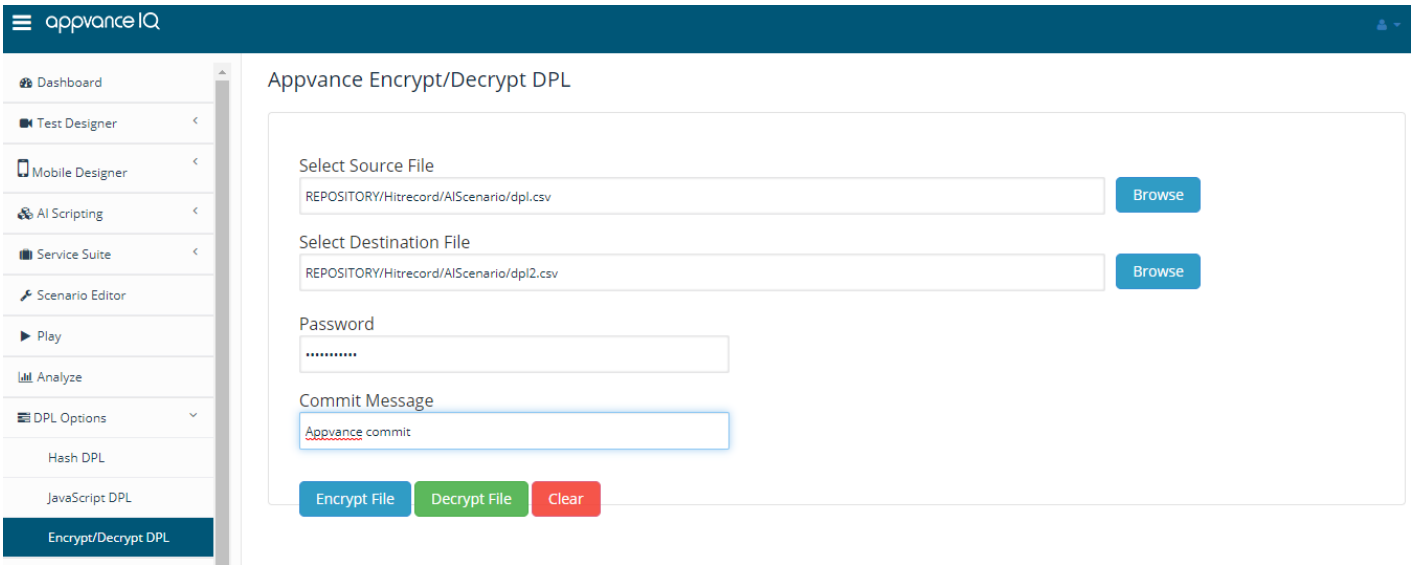
Decrypting a DPL

The files can be decrypted by choosing the decrypted file and again choosing a different file to decrypt and providing the same password.



Saving File to Repository

When saving the encrypted files to the repository, you will also have an option to provide the commit message.



Last updated: May 06, 2025

If you have any feedback on this topic, please contact us at documentation@appvance.ai

Operational Test Data FAQ

Operational Test Data FAQ

Operational test data is the set of values a test consumes during execution.

Last updated: September 18, 2026