Informatica® PowerExchange for HDFS 10.2.2

User Guide

Informatica PowerExchange for HDFS User Guide
10.2.2
February 2019

Publication Date: 2019-06-10

Table of Contents

Preface

The *Informatica PowerExchange® for HDFS User Guide* provides information about reading data from the Hadoop Distributed File System (HDFS) and writing data to HDFS. The guide is written for database administrators and developers.

This book assumes you have knowledge of HDFS and Informatica Developer.

Informatica Resources

Informatica provides you with a range of product resources through the Informatica Network and other online portals. Use the resources to get the most from your Informatica products and solutions and to learn from other Informatica users and subject matter experts.

Informatica Network

The Informatica Network is the gateway to many resources, including the Informatica Knowledge Base and Informatica Global Customer Support. To enter the Informatica Network, visit https://network.informatica.com.

As an Informatica Network member, you have the following options:

- Search the Knowledge Base for product resources.
- View product availability information.
- Create and review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Knowledge Base

Use the Informatica Knowledge Base to find product resources such as how-to articles, best practices, video tutorials, and answers to frequently asked questions.

To search the Knowledge Base, visit https://search.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Documentation

Use the Informatica Documentation Portal to explore an extensive library of documentation for current and recent product releases. To explore the Documentation Portal, visit https://docs.informatica.com.

Informatica maintains documentation for many products on the Informatica Knowledge Base in addition to the Documentation Portal. If you cannot find documentation for your product or product version on the Documentation Portal, search the Knowledge Base at https://search.informatica.com.

If you have questions, comments, or ideas about the product documentation, contact the Informatica Documentation team at infa_documentation@informatica.com.

Informatica Product Availability Matrices

Product Availability Matrices (PAMs) indicate the versions of the operating systems, databases, and types of data sources and targets that a product release supports. You can browse the Informatica PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services and based on real-world experiences from hundreds of data management projects. Informatica Velocity represents the collective knowledge of Informatica consultants who work with organizations around the world to plan, develop, deploy, and maintain successful data management solutions.

You can find Informatica Velocity resources at http://velocity.informatica.com. If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that extend and enhance your Informatica implementations. Leverage any of the hundreds of solutions from Informatica developers and partners on the Marketplace to improve your productivity and speed up time to implementation on your projects. You can find the Informatica Marketplace at https://marketplace.informatica.com.

Informatica Global Customer Support

You can contact a Global Support Center by telephone or through the Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
https://www.informatica.com/services-and-training/customer-success-services/contact-us.html.

To find online support resources on the Informatica Network, visit https://network.informatica.com and select the eSupport option.

CHAPTER 1

Introduction to PowerExchange for HDFS

This chapter includes the following topic:

PowerExchange for HDFS Overview

PowerExchange for HDFS provides connectivity to the Hadoop Distributed File System (HDFS). You can use PowerExchange for HDFS to read data from and write data to HDFS. You can also use PowerExchange for HDFS to read data from and write data to the local file system.

The Data Integration Service uses the Hadoop API infrastructure to connect to HDFS. It connects to the NameNode to read data from and write data to HDFS.

With PowerExchange for HDFS, you can read and write fixed-width text files, delimited text files, and the industry-standard file formats, such as Avro, Parquet, ORC, JSON, and XML files. You can read and write hierarchical data present in the Avro, Parquet, ORC, JSON, and XML files. In addition to the industry-standard file formats, you can also read from intelligent structure sources.

With PowerExchange for HDFS, you can read and write files as binary and write them to a target. When you select the file type as binary for a complex file source or target, PowerExchange for HDFS can process the mapping with or without using the Data Processor transformation.

You can read and write compressed files. You can configure custom formats to process data in input, output, and compression formats that Hadoop supports.

CHAPTER 2

PowerExchange for HDFS Configuration

This chapter includes the following topics:

PowerExchange for HDFS Configuration Overview

PowerExchange for HDFS is installed with Informatica Data Services. You enable PowerExchange for HDFS with a license key.
Note: To read or write data with a complex file data object, you will also need the Unstructured Data license key.

Prerequisites

Before you use PowerExchange for HDFS to access data in HDFS, perform the following tasks:

- Install and configure Informatica Services. Verify that the domain has a Data Integration Service and a Model Repository Service.
- Verify that a cluster configuration is created in the domain.
- Verify that a Metadata Access Service is created in the domain.
- To successfully preview data from a local complex file or run a mapping in the native environment with a local complex file, you must configure the INFA_PARSER_HOME property for the Data Integration Service in Informatica Administrator. Perform the following steps to configure the INFA_PARSER_HOME property:
  - Log in to Informatica Administrator.
  - Click the Data Integration Service and then click the **Processes** tab on the right pane.
  - Click **Edit** in the **Environment Variables** section.
  - Click **New** to add an environment variable.
  - Enter the name of the environment variable as **INFA_PARSER_HOME**.

- Set the value of the environment variable to the absolute path of the Hadoop distribution directory on the machine that runs the Data Integration Service. Verify that the version of the Hadoop distribution directory that you define in the INFA_PARSER_HOME property is the same as the version you defined in the cluster configuration.

CHAPTER 3

HDFS Connections

This chapter includes the following topics:

HDFS Connections Overview

Create an HDFS connection to read data from or write data to HDFS.

HDFS Connection Properties

Use a Hadoop File System (HDFS) connection to access data in the Hadoop cluster. The HDFS connection is a file system type connection. You can create and manage an HDFS connection in the Administrator tool, Analyst tool, or the Developer tool. HDFS connection properties are case sensitive unless otherwise noted.

Note: The order of the connection properties might vary depending on the tool where you view them.

The following table describes HDFS connection properties:

Property	Description
Name	Name of the connection. The name is not case sensitive and must be unique within the domain. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: ~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. The description cannot exceed 765 characters.
Location	The domain where you want to create the connection. Not valid for the Analyst tool.
Type	The connection type. Default is Hadoop File System.

Property	Description
User Name	User name to access HDFS.
NameNode URI	The URI to access the storage system. You can find the value for `fs.defaultFS` in the `core-site.xml` configuration set of the cluster configuration. **Note:** If you create connections when you import the cluster configuration, the NameNode URI property is populated by default, and it is updated each time you refresh the cluster configuration. If you manually set this property or override the value, the refresh operation does not update this property.

Accessing Multiple Storage Types

Use the NameNode URI property in the connection parameters to connect to various storage types. The following table lists the storage type and the NameNode URI format for the storage type:

Storage	NameNode URI Format
HDFS	`hdfs://<namenode>:<port>` where: - `<namenode>` is the host name or IP address of the NameNode. - `<port>` is the port that the NameNode listens for remote procedure calls (RPC). `hdfs://<nameservice>` in case of NameNode high availability.
MapR-FS	`maprfs:///`

WASB in HDInsight	`wasb://<container_name>@<account_name>.blob.core.windows.net/<path>`
	where:
	- `<container_name>` identifies a specific Azure Storage Blob container.
	Note: `<container_name>` is optional.
	- `<account_name>` identifies the Azure Storage Blob object.
	Example:
	`wasb://infabdmoffering1storage.blob.core.windows.net/infabdmoffering1cluster/mr-history`

ADLS in HDInsight	`adl://home`

When you create a cluster configuration from an Azure HDInsight cluster, the cluster configuration uses either ADLS or WASB as the primary storage. You cannot create a cluster configuration with ADLS or WASB as the secondary storage. You can edit the NameNode URI property in the HDFS connection to connect to a local HDFS location.

Creating an HDFS Connection

Create an HDFS connection before you import physical data objects.

1. Click **Window** > **Preferences**.
2. Select **Informatica** > **Connections**.

3. Expand the domain.
4. Select the connection type **File Systems** > **Hadoop File System**, and click **Add**.
5. Enter a connection name.
6. Optionally, enter a connection description.
7. Click **Next**.
8. Enter the user name to access HDFS.
9. Enter the NameNode URI to access HDFS based on the Hadoop distribution that you use.
10. Select the cluster configuration associated with the Hadoop environment.
11. Click **Test Connection**. If a default Metadata Access Service is not set, a message appears to configure the Metadata Access Service. Click **OK** and set one Metadata Access Service as default. After you set a default Metadata Access Service, the connection to HDFS is tested. If the Metadata Access Service does not exist, contact the Informatica administrator to create a new Metadata Access Service in the domain.
12. Click **Finish**.

Related Topics:

- "HDFS Connection Properties" on page 11

CHAPTER 4

HDFS Data Objects

This chapter includes the following topics:

HDFS Data Objects Overview

After you configure an HDFS connection, create a physical data object to read data from or write data to HDFS.

Depending on the file format, you can configure the following types of physical data objects:

- Flat file data object. Create or import a flat file data object and configure an HDFS connection for the data object. Use the flat file data object to read or write fixed-width or delimited text files.
- Complex file data object. Import a complex file data object with an HDFS connection. Use the complex file data object to read or write structured, semi-structured, unstructured data. For example, Avro, Parquet, Orc, XML, and JSON files have structured or semi-structured data. Binary files, such as PDF and Microsoft Word have unstructured data. Complex file data objects can also read intelligent structure sources.

Generate the Source File Name for HDFS Data Objects

You can add a file name column to the flat file data object. The file name column helps you to identify the source file that contains a particular record of data. You can configure the mapping with the file name column for both flat file and complex file data objects. When you read data from HDFS, you can extract the fully qualified path of the source file.

You can configure the mapping to write the source file name to each source row when you add a File Name Column port in the Overview view. The File Name Column port contains the name and the fully qualified path for each source file. The File Name Column port is a string port with a default precision of 256 characters.

If the file or directory is in HDFS, enter the path without the node URI. For example,`/user/lib/testdir` specifies the location of a directory in HDFS. The path must not contain more than 512 characters.

When you use a file name column in a Read transformation, the file name column returns the value in the following format for HDFS:

```
hdfs://<host name>:<port>/<file name path>
```

For example, the file name column returns `hdfs://irldv:5008/hive/warehouse/ff.txt`, where the host name is irldv and the port is 5008.

FileName Port Overview

A FileName port is a string port with a default precision of 1024 characters that contains the source path of a file.

You cannot configure the FileName port. You can delete the FileName port if you do not want to read or write the data in the FileName. You cannot create a folder name which contains more than 255 characters.

When you create a data object read or write operation for all the complex files, the FileName port is displayed by default.

FileName port appears when you run a mapping in the native environment or on the Spark engine to read or write an Avro, ORC, JSON, Parquet, or binary(native) file.

Working with FileName Port

You can use the FileName port in a complex file target to dynamically redirect the rows based on the value received by FileName port.

When you run a mapping in the native environment to read or write a flat file using the FileName port, the Data Integration Service creates separate directories for each value in the FileName port in the following format:

```
<CFW FileName>/<CFW FileName>=<valueFromMappingFlow>
```

When you run a mapping to read or write a complex file using the FileName port, the result varies based on the complex file format that you use and the engine where you run the mapping.

When you run a mapping in the native environment or on the Spark engine to read or write an Avro, JSON, ORC, Parquet or Binary(Native) file using the FileName port, the Data Integration Service creates separate directories for each value in the FileName port and adds the files within the directories.

Rules and Guidelines for Using FileName Port

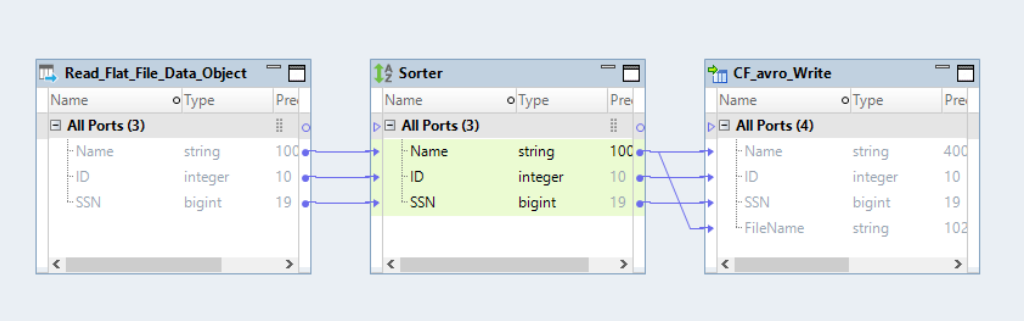
Use the following rules and guidelines when you use the FileName data in the FileName port:

- Do not use a colon (:) and forward slash (/) character in the file name data of the FileName port of the source or target object to run a mapping.
- If you connect the FileName port to the target empty zero KB files are created in the target folder.
- Do not connect FileName port to a FileName port because the FileName port in the source might contain colon (:) and forward slash (/) characters.

- In the Native environment use the Sorter transformation to sort the source port that you want to map to the FileName port of the Target transformation. After you sort the source port, map the port of the Sorter transformation to the FileName port of the Target transformation. The Data Integration Service creates only one file for each value with the same name. If you do not use the Sorter transformation, the Data Integration Service creates multiple files for each value with the same name.

  For example, Create a mapping in the Native environment or on the Spark engine to read or write an Avro file using the FileName port.

  The following image shows the sorter transformation mapping:



  If you want to Map the following source port name to the FileName port of the Target transformation and write the data to an Avro target file `target1`:

Name	ID	SSN
Anna	1	1
John	4	4
Smith	4	4
John	5	5
Anna	2	2

  Add a Sorter transformation to sort the source port and map the source port to the port of the Sorter transformation. Then, map the port of the Sorter transformation to the FileName port of the Target transformation. The Data Integration Service creates the following directories and single file per thread within the directories:

  ```
  target1.avro=Anna
  ```

  In this directory, the Data Integration Service creates a file with the following values: `1,1,1,2,2,2`.

  ```
  target1.avro=John
  ```

  In this directory, the Data Integration Service creates a file with the following values: `4,4,4,5,5,5`.

  ```
  target1.avro=Smith
  ```

  In this directory, the Data Integration Service creates a file with the following values: `4,4,4`.

  If you do not add a Sorter transformation, the Data Integration Service creates the following directories and multiple files within the directories:

  ```
  target1.avro=Anna
  ```

In this directory, the Data Integration Service creates two part files with the following values: `1,1,1` and `2,2,2`.

```
target1.avro=John
```

In this directory, the Data Integration Service creates two files with the following values: `4,4,4` and `5,5,5`.

```
target1.avro=Smith
```

In this directory, the Data Integration Service creates one file with the following values: `4,4,4`.

Flat File Data Objects

You can read data from and write data to HDFS through a fixed-width or delimited flat file data object that does not contain binary data.

You can create or import a flat file data object. The data object properties that you specify in the Developer tool must match the properties of the source file.

After you create a flat file data object, you can edit the following file properties:

- HDFS connection properties
- Compression formats

To read large volumes of data, you can connect a flat file source to read data from a directory of flat files.

You can use the flat file data objects as a source, target, or lookup transformation in mappings and mapplets. You can select the mapping environment and run the mappings in a native or Hadoop run-time environment. You can create and run profiles against flat file data objects.

When you configure a mapping that contains flat file data objects to run in the native environment, you can enable the mapping for partitioning. The Data Integration Service can use multiple partitions to read data from flat file sources with an HDFS connection. The Data Integration Service can also use multiple partitions to write data to flat file targets with an HDFS connection. When the Data Integration Service adds partitions, it increases the number of processing threads, which can increase mapping performance.

Compression and Decompression for Flat File Sources and Targets

File compression can increase data transfer rates and reduce space for data storage.

You can read and write compressed flat files, specify compression formats, and decompress files. You can compress and decompress files in compression formats such as Bzip2 and Lzo, or specify a custom compression format.

You can specify a file or a directory of files. When the Data Integration Service reads from a directory, it reads the files of the specified format only and ignores files of other formats.

For information about how Hadoop processes compressed and uncompressed files, see the Hadoop documentation.

The following table describes the compression options:

Compression Options	Description
None	The file is not compressed.
Auto	The Data Integration Service detects the compression format of the file based on the file extension.
Gzip	The GNU zip compression format that uses the DEFLATE algorithm.
Bzip2	The Bzip2 compression format that uses the Burrows–Wheeler algorithm.
Lzo	The Lzo compression format that uses the Lempel-Ziv-Oberhumer algorithm. **Note:** The JAR files for LZO compression are not available with the default Hadoop installation. You must place the JAR files for the LZO compression format in the `lib` folder of the distribution directory and verify the distribution directory properties.
Custom	Custom compression format. If you select this option, you must specify the fully qualified class name implementing the Hadoop `CompressionCodec` interface in the **Compression Codec** field.

Related Topics:

- "Custom Formats Configuration" on page 24

Rules and Guidelines for Flat File Data Objects

Use the following rules and guidelines when you use flat file sources with an HDFS connection:

- You cannot use a command to generate or transform flat file data and send the output to the flat file source at run time.
- You cannot use an indirect source type.

Use the following rules and guidelines when you use flat file targets with an HDFS connection:

- You cannot append output data to target files and reject files. The Data Integration Service truncates the target files and reject files before writing the data.
- You cannot use the command output type.
- When the flat file target is in a partitioned mapping, you cannot write to a merge file that contains the target output for all partitions. The Data Integration Service concurrently writes the target output to a separate file for each partition.

Configuring a Flat File Data Object with an HDFS Connection

Configure a flat file data object with an HDFS connection to read data from or write data to HDFS.

If you create an empty flat file, the file properties must match that of the file in HDFS. If you import a flat file data object, the file must reside in your local file system.

1. Click the **Advanced** tab of the flat file data object.
2. Navigate to the runtime properties for the flat file source in the **Runtime: Read** properties or the flat file target in the **Runtime: Write** properties.
3. Configure the HDFS connection properties.
4. Optionally, you can configure the compression properties.

Related Topics:

- "Flat File Data Object Read Properties" on page 26
- "Flat File Data Object Write Properties" on page 33

Naming Convention for Flat File Targets

When you run a mapping on the Blaze engine to write data to a flat file target, the Data Integration Service creates multiple target files with the following naming convention:

```
<FileName>-P1, <FileName>-P2,....<FileName>-P100....<FileName>-PN
```

The naming convention helps you to delete multiple split files generated from the previous mapping runs and to avoid the deletion of target files generated from another mapping with similar file names.

Complex File Data Objects

A complex file data object is a representation of a file in the Hadoop file system. Create a complex file data object to read or write structured, semi-structured, and unstructured data to HDFS.

You can read files from the local system or HDFS. Similarly, you can write files to the local system or HDFS. To read large volumes of data, you can connect a complex file source to read data from a directory of files that have the same format and properties. You can read and write compressed binary files.

When you read or write the industry-standard file formats, you may or may not use the Data Processor transformation based on the structure of the file and the engine you select to run the mapping.

You can use a complex file data object with an intelligent structure model to read and parse semi-structured or structured data from text, CSV, XML, or JSON files, as well as PDF forms, Microsoft Word tables, or Microsoft Excel. The output of the complex file data object is primitive and complex elements. You do not need to use a Data Processor transformation with a complex file data object that uses an intelligent structure model. The Data Integration Service can directly read intelligent structure model resources to HDFS or the local file system.

When you use a binary complex file data object as a source, you can use a Data Processor transformation to parse the file. The output of the binary complex file data object is a binary stream. Similarly, when you write binary data to a complex file, you must use a Data Processor transformation to convert the source data into a binary format. You can then use the binary stream to write data to the binary complex file.

When you create a complex file data object, a read and write operation is created. You can use the complex file data object read operation as a source in mappings and mapplets. You can use the complex file data object write operation as a target in mappings and mapplets. You can select the mapping environment and run the mappings in a native or Hadoop run-time environment.

Complex File Data Object Overview Properties

The Data Integration Service uses overview properties when it reads data from or writes data to a complex file.

Overview properties include general properties that apply to the complex file data object. They also include object properties that apply to the resources in the complex file data object. The Developer tool displays overview properties for complex files in the **Overview** view.

General Properties

The following table describes the general properties that you configure for complex files:

Property	Description
Name	The name of the complex file data object.
Description	The description of the complex file data object.
Access Method	The access method for the resource. - Connection. Select **Connection** to specify an HDFS connection. - File. Select **File** to browse for a file on your local system.
Connection	The name of the HDFS connection.

Objects Properties

The following table describes the objects properties that you configure for complex files:

Property	Description
Name	The name of the resource.
Type	The native data type of the resource.
Description	The description of the resource.
Access Type	Indicates that you can perform read and write operations on the complex file data object. You cannot edit this property.

Compression and Decompression for Complex File Sources and Targets

You can read and write compressed complex files, specify compression formats, and decompress files. You can use compression formats such as Bzip2 and Lzo, or specify a custom compression format. The compressed files must be of the binary format.

You can compress sequence files at a record level or at a block level.

For information about how Hadoop processes compressed and uncompressed files, see the Hadoop documentation.

The following table describes the complex file compression formats for binary files:

Compression Options	Description
None	The file is not compressed.
Auto	The Data Integration Service detects the compression format of the file based on the file extension.
DEFLATE	The DEFLATE compression format that uses a combination of the LZ77 algorithm and Huffman coding.

Compression Options	Description
Gzip	The GNU zip compression format that uses the DEFLATE algorithm.
Bzip2	The Bzip2 compression format that uses the Burrows–Wheeler algorithm.
Lzo	The Lzo compression format that uses the Lempel-Ziv-Oberhumer algorithm.
Snappy	The LZ77-type compression format with a fixed, byte-oriented encoding. **Note:** Default compression format is Snappy on the Spark engine.
Custom	Custom compression format. If you select this option, you must specify the fully qualified class name implementing the `CompressionCodec` interface in the **Custom Compression Codec** field.

Related Topics:

- "Custom Formats Configuration" on page 24

Parameterization of Complex File Data Objects

You can parameterize the complex file connection and the complex file data object operation properties.

You can parameterize the following data object read operation properties for complex data objects:

- Connection in the run-time properties
- File Format, Input Format, Compression Format, and Custom Compression Codec in the advanced properties.

You can parameterize the following data object write operation properties for complex file data objects:

- Connection in the run-time properties.
- File Name, Output Format, Output Key Class, Output Value Class, Compression Format, Custom Compression Codec, and Sequence File Compression Type in the advanced properties.

The following attributes support full and partial parameterization for complex file data objects:

- File Path in the advanced properties of the data object read operation.
  For example, to parameterize a part of the attribute value where the file path in the advanced property is `/user/adpqa/dynschema.txt`, create a parameter as *$str="/user/adpqa"*, and then edit the file path as `$str/dynschema.txt`. You can also parameterize the value of the entire file path.
- File Directory in the advanced properties of the data object write operation.
  For example, to parameterize a part of the attribute value where the file directory in the advanced property is `/export/home/qamercury/source`, create a parameter as *$param="/export/home"*, and then edit the file directory as `$param/qamercury/source`. You can also parameterize the value of the entire directory.

Complex File Data Object Output Parsing

You can use an Avro or Parquet format complex file data object as a source or target without using a Data Processor transformation. The Data Integration Service can directly read and write Avro and Parquet resources that contain flat structure to HDFS or local file system.

You can use a complex file data object with an intelligent structure model resource as a source in a mapping that runs over Spark. When you associate a complex file data object with an intelligent structure model, you can use any file input that the intelligent structure model applies to without using a Data Processor transformation.

When you use a binary complex file data object as a source, you can use a Data Processor transformation to parse the binary output of the complex file.

Configure the Data Processor transformation as follows:

- Set an input port to buffer input and binary data type. Specify the port size. The port size that you specify in the complex file properties and the Data Processor transformation must be the same.
- Set an output port to buffer output or set it for relational output. If you set the ports for relational output, specify the ports based on the number of relational groups of ports you want in the output. Specify the port size for the ports. You can use an XML schema reference that describes the XML hierarchy.
- Set a Streamer object as a startup component.

If you configure a binary complex file data object with an intelligent structure model, you do not need to use a Data Processor transformation to parse the output of the complex file.

When you use a complex file data object as a target, you must use a Data Processor transformation to convert the source data into a binary format. Set the Data Processor transformation port to binary. You can then use the binary stream as an input to the complex file data object.

Creating a Complex File Data Object

Create a complex file data object to read data from or write data to HDFS.

1. Select a project or folder in the **Object Explorer** view.
2. Click **File** > **New** > **Data Object**.
3. Select **Complex File Data Object** and click **Next**.

   The **New Complex File Data Object** dialog box appears.
4. Optionally, enter a name for the data object.
5. Click **Browse** next to the **Location** option and select the target project or folder.
6. In the **Resource Format** list, select any of the following formats:
   - Intelligent Structure Model: to read any format that an intelligent structure parses.
   - Binary: to read any resource format.
   - Avro: to read an Avro resource.
   - Parquet: to read a Parquet resource.
   - JSON: to read a JSON resource.
   - Orc: to read an Orc resource.
   - XML: to read an XML resource.

   Note: Intelligent structure model is supported only in Spark mode.
7. In the **Access Type** list, select **Connection** or **File**.
   - Select **Connection** to access a file on HDFS. Click **Browse** next to the **Connection** option and select an HDFS connection. Click **Add** next to the **Selected Resource** option to add a resource to the data object. If a default Metadata Access Service is not set, a message appears to configure the Metadata Access Service. Click **OK** and set one Metadata Access Service as default. After you set a default Metadata Access Service, the **Add Resource** dialog box appears. If the Metadata Access Service

does not exist, contact the Informatica administrator to create a new Metadata Access Service in the domain. Navigate or search for the resources to add to the data object and click **OK**.

- Select **File** to access a file on your local system. Click **Browse** next to the **Resource Location** option and select the file that you want to add. Click **Fetch**. The selected file is added to the **Selected Resources** list.

Note: To use an intelligent structure model, for the **Selected Resource** option, browse to and select the appropriate `.amodel` file.

8. From the **Available OS Profiles** list, select an operating system profile. You can use the **Available OS Profiles** to increase security and to isolate the design-time user environment when you import and preview metadata from a Hadoop cluster.

   Note: The Developer tool displays the **Available OS Profiles** list only if the Metadata Access Service is enabled to use operating system profiles. The Metadata Access Service imports the metadata with the default operating system profile assigned to the user. You can change the operating system profile from the list of available operating system profiles.

9. Click **Finish**.

   The data object appears under the Physical Data Objects category in the project or folder in the **Object Explorer** view. A read and write operation is created for the data object. Depending on whether you want to use the complex file data object as a source or target, you can edit the read or write operation properties. You can also create multiple read and write operations for a complex file data object. For a data object with an intelligent structure model, create a read operation. You cannot use a write transformation for a data object with an intelligent structure model in a mapping.

   Note: The complex file data object write operation goes through and the mapping runs successfully even if you have unconnected ports for required fields in the Parquet resource type. The NULL values are inserted in the target object when such a mapping runs. The complex file data object read operation results in an error while reading NULL values from the Parquet resource as Parquet Example Object Model does not support NULL read.

10. For a read operation with an intelligent structure model, specify the path to the input file. In the **Data Object Operations** panel, select the **Advanced** tab. In the **File path** field, specify the path to the input file.

Creating a Complex File Object Read or Write Operation

You can add an complex file data object read or write operation to a mapping or mapplet as a source.

Before you create an complex file data object read or write operation, you must create at least one complex file data object. You can create the data object read or write operation for one or more complex file data objects.

Perform the following steps to create an complex file data object read or write operation:

1. Select the data object in the **Object Explorer** view.
2. Right-click and select **New** > **Data Object Operation**.

   The **Data Object Operation** dialog box appears.
3. Enter a name for the data object read or write operation.
4. Select **Read** or **Write** as the type of data object operation.
5. Click **Add**.

   The **Select Resources** dialog box appears.
6. Select the complex file object for which you want to create the data object read or write operation and click **OK**.
7. Click **Finish**.

The Developer tool creates the data object read or write operation for the selected data object.

Rules and Guidelines for Creating a Complex File Data Object Operation

Use the following rules and guidelines when you create an complex file data object operation:

- When you create a data object read or write operation, you can add new columns or modify the columns in the **Ports** tab directly.
- To modify the columns of a complex file, you must reconfigure the column projection properties.
- To modify the columns of an Avro, JSON, ORC, or Parquet file, change the complex file file format in the **Schema** field of the column projection properties.
- When you create a mapping to read or write an Avro, JSON, ORC, or Parquet file, you can copy the columns of the Source transformations, Target transformations, or any other transformations from the **Ports** tab. Then, you can paste the columns in the data object read or write operation directly.
- When you copy the columns from any transformation to the data object read or write operation, you can change the data type of the columns. The Data Integration Service resets the precision value of the data type to the default value.
  However, the Data Integration Service does not change the precision value of the String data type to the default value.

Custom Formats

Custom formats provide flexibility with the input, output, and compression formats that you can use with PowerExchange for HDFS.

Apart from the input, output, and compression formats that PowerExchange for HDFS supports, you can use custom formats to read, write, and compress files. You can use the custom formats that Hadoop supports.

You can specify the following custom formats:

- Custom input format for complex file data objects
- Custom output format for complex file data objects
- Custom compression format for flat file and complex file data objects

Custom Formats Configuration

Before you use custom formats, you must complete configuration tasks in the Informatica environment.

To use custom formats in the native environment, copy the .jar files that implement the custom formats to the following directory:

```
<Informatica installation directory>/services/shared/hadoop/<hadoop distribution name>/
infaLib
```

To use custom formats in the Hadoop environment, see the Hadoop documentation for information about the prerequisite tasks.

If the custom compression includes native libraries, depending on the run-time environment, add the path of the native libraries to the environment variable $LD_LIBRARY_PATH or to the Hadoop connection. If you use

the native environment, add the path of the native libraries to the environment variable $LD_LIBRARY_PATH. If you use the Hadoop environment, add the path of the native libraries to the Hadoop connection.

Perform the following steps to add the path of the native libraries to the Hadoop connection:

1. Click the **Common Attributes** tab in the Hadoop connection.
2. Under the **Common Properties** section, click **Edit** next to the **Advanced Properties** field.
3. Add the infapdo.java.opts property and set its value to the path of the native libraries.
   For example, the following property specifies a native library path for a Cloudera distribution:

   ```
   infapdo.java.opts=-Djava.library.path=$HADOOP_NODE_INFA_HOME/services/shared/bin:
   $HADOOP_NODE_INFA_HOME/services/shared/hadoop/CDH_5.13/lib/native
   ```

   Note: If you use Hortonworks or MapR distributions, change the native library path based on the distribution.

CHAPTER 5

HDFS Data Extraction

This chapter includes the following topics:

HDFS Data Extraction Overview

You can use a flat file data object or a complex file data object to read data from HDFS.

Complete the following tasks to read data from HDFS by using PowerExchange for HDFS:

1. Create an HDFS connection.
2. Create a flat file data object or a complex file data object. Specify the data object properties such as the file location, compression format, and input format.
3. Create a mapping and use the flat file data object or the complex file data object read operation as a source.
4. If needed, configure a Data Processor transformation to parse the complex file.
5. Configure the validation and run-time environment type.
6. Run the mapping to read data from HDFS.

Flat File Data Object Read Properties

The Data Integration Service uses read properties when it reads data from a flat file. You can edit the format and runtime read properties on the **Advanced** tab.

The following table describes the HDFS connection and compression run-time properties that you configure for flat file sources:

Property	Description
Connection Type	The type of connection. Select from the following options: - None. The source file does not require a connection. - Hadoop File System. The source file resides in HDFS. Default is None.
Connection Name	The name of the connection. Select an HDFS connection or assign a mapping parameter that defines the connection details.
Compression Format	Optional. Specifies the compression format. Select from the following options: - None - Auto - Gzip - Bzip2 - Lzo - Custom
Compression Codec	Required for custom compression. Specify the fully qualified class name implementing the Hadoop `CompressionCodec` interface.

Complex Files Partitioning

When you run a mapping in a Hadoop environment to read data from sequence files and custom input format files that are splittable, the Data Integration Service uses multiple partitions to read data from the source. The Data Integration Service creates multiple Map jobs to read data in parallel, thereby resulting in high performance.

To read text files in parallel, specify the following input format in the complex file read properties:

```
com.informatica.adapter.hdfs.hadoop.io.InfaTextInputFormat
```

You can also specify the following input format to read text files in batches:

```
com.informatica.adapter.hdfs.hadoop.io.InfaBatchTextInputFormat
```

Typically, when you read complex files, the Data Processor transformation has a Streamer component and a Parser component. By default, the Data Integration Service calls the Data Transformation Engine for every record. You can modify this behavior by using the count property in the Streamer component. Set the count property to define the number of records that the Data Integration Service must treat as a batch. When you set the count property, the Data Integration Service calls the Data Transformation Engine for each batch of records instead of calling the Data Transformation Engine for every record. Since the Data Integration Service processes the text files in batches, the performance increases.

Complex File Data Object Read Properties

The Data Integration Service uses read properties when it reads data from a complex file. Select the Output transformation to edit the general, ports, sources, and run-time properties.

Note: The FileName port is displayed by default when you create a data object read operation. You can remove the FileName port if you do not want to read the FileName data.

Wildcard Characters for Reading Data from Complex Files

When you run a mapping in the native environment or on the Spark engine to read data from complex files, you can use wildcard characters to specify the source directory name or the source file name. You can use wildcard characters to specify the absolute path or relative path.

To use wildcard characters for the source directory name or the source file name, select the **Allow Wildcard Characters** option in the advanced read properties of the complex file data object. You can then use wildcard characters in the **File path** field.

You can use the following wildcard characters:

? (Question mark)

The question mark character (?) allows one occurrence of any character. For example, if you enter the source file name as `a?b.txt`, the Data Integration Service reads data from files with the following names:

- `a1b.txt`
- `a2b.txt`
- `aab.txt`
- `acb.txt`

*** (Asterisk)**

The asterisk mark character (*) allows zero or more than one occurrence of any character. If you enter the source file name as `a*b.txt`, the Data Integration Service reads data from files with the following names:

- `aab.txt`
- `a1b.txt`
- `ab.txt`
- `abc11b.txt`

Combination of * (Asterisk) and ? (Question mark)

The combination of asterisk mark character (*) and question mark character (?) allows zero or more than one occurrence of any character.

General Properties

The Developer tool displays general properties for complex file sources in the **Read** view.

The following table describes the general properties that you configure for complex file sources:

Property	Description
Name	The name of the complex file. This property is read-only. You can edit the name in the **Overview** view. When you use the complex file as a source in a mapping, you can edit the name in the mapping.
Description	The description of the complex file.

Ports Properties

Ports properties for a physical data object include port names and port attributes such as data type and precision.

Note: The port size specified in the source transformation and Output transformation must be the same.

The following table describes the ports properties that you configure for complex file sources:

Property	Description
Name	The name of the resource.
Type	The native data type of the resource.
Precision	The maximum number of significant digits for numeric data types, or the maximum number of characters for string data types.
Description	The description of the resource.

Sources Properties

The Developer tool displays the sources properties for complex file sources in the Output transformation in the **Read** view.

The sources properties list the resources of the complex file data object. You can add or remove resources in the data object.

Advanced Properties

The Developer tool displays the advanced properties for complex file sources in the Output transformation in the **Read** view.

The following table describes the advanced properties that you configure for complex file sources:

Property	Description
Allow Wildcard Characters	Indicates whether you want to use wildcard characters for the source directory name or the source file name. If you select this option, you can use wildcard characters ? and * for the source directory name or the source file name in the **File path** field. The question mark character (?) allows one occurrence of any character. The asterisk character (*) allows zero or more than one occurrence of any character. This option is applicable when you run a mapping in the native environment or on the Spark engine.
File Format	The file format. Select one of the following file formats: - Binary. Select Binary to read any file format. - Sequence. Select Sequence File Format for source files of a Hadoop-specific binary format that contain key and value pairs. - Custom Input. Select Input File Format to specify a custom input format. You must specify the class name implementing the `InputFormat` interface in the **Input Format** field. Default is Binary. **Note:** Binary file format is not supported for the complex file data object operation in a spark mapping.

Input Format	The class name for files of the input file format. If you select **Input File Format** in the **File Format** field, you must specify the fully qualified class name implementing the `InputFormat` interface.
	To read files that use the Avro format, use the following input format:
	`com.informatica.avro.AvroToXML`
	To read files that use the Parquet format, use the following input format:
	`com.informatica.parquet.ParquetToXML`
	You can use any class derived from `org.apache.hadoop.mapreduce.InputFormat`.

Input Format Parameters	Parameters for the input format class. Enter name-value pairs separated with a semicolon. Enclose the parameter name and value within double quotes. For example, use the following syntax: `"param1"="value1";"param2"="value2"`
Compression Format	Optional. The compression format for binary files. Select one of the following options: - None - Auto - DEFLATE - gzip - bzip2 - Lzo - Snappy - Custom Not applicable to Avro and Parquet formats.

Property	Description
Custom Compression Codec	Required for custom compression. Specify the fully qualified class name implementing the `CompressionCodec` interface.
File path	The location of the file or directory. If the path is a directory, all the files in the directory must have the same file format. If the file or directory is in HDFS, enter the path without the node URI. For example, `/user/lib/testdir` specifies the location of a directory in HDFS. The path must not contain more than 512 characters. If the file or directory is in the local system, enter the fully qualified path. For example, `/user/testdir` specifies the location of a directory in the local system. **Note:** The Data Integration Service ignores any subdirectories and their contents. If you select the **Allow Wildcard Characters** option, you can use wildcard characters ? and * for the source directory name or the source file name.

Column Projection Properties

The Developer tool displays the column projection properties for intelligent structure model, Avro, JSON, and Parquet complex file sources in the Properties view of the **Read** operation.

The following table describes the column projection properties that you configure for the complex file sources:

Property	Description
Enable Column Projection	Displays the column details of the complex files sources.
Schema Format	Displays the schema format that you selected while creating the complex file data object. You can change the schema format and provide respective schema. You can select one of the following complex file formats: - Avro - JSON - ORC - Parquet You can change the complex file format without losing the column metadata even after you configure the column projection properties for another complex file format. **Note:** You can switch from one schema format to another only once. If you change the schema format more than once, you might lose the original datatypes.
Schema	Displays the schema associated with the complex file. You can select a different schema. **Note:** If you disable the column projection, the schema associated with the complex file is removed. If you want to associate schema again with the complex file, enable the column projection and click Select Schema.
Use Intelligent Structure Model	Select this option to associate an intelligent structure model with the complex file. **Warning:** Do not associate an intelligent structure model with a Write data object operation. If you use a Write operation that is associated with an intelligent structure model in a mapping, the mapping will not be valid.

Property	Description
Model	Displays the intelligent structure model associated with the complex file. You can select a different model. **Note:** If you disable the column projection, the intelligent structure model associated with the data object is removed. If you want to associate an intelligent structure model again with the data object, enable the column projection and click **Select Model**.
Column Mapping	Displays the mapping between input and output ports. **Note:** If you disable the column projection, the mapping between input and output ports is removed. If you want to map the input and output ports, enable the column projection and click **Select Schema** to associate a schema to the complex file.
Project Column as Complex Data Type	Displays columns with hierarchical data as a complex data type, such as, array, map, or struct. Select this property when you want to process hierarchical data on the Spark engine. **Note:** If you disable the column projection, the data type of the column is displayed as binary type.

CHAPTER 6

HDFS Data Load

This chapter includes the following topics:

HDFS Data Load Overview

You can use a flat file data object or a complex file data object to write data to HDFS.

Complete the following tasks to write data to HDFS by using PowerExchange for HDFS:

1. Create an HDFS connection.
2. Create a flat file data object or a complex file data object. Specify the data object properties such as the file location and compression format.
3. Create a mapping and use the flat file data object or the complex file data object write operation as a target.
4. Configure the validation and run-time environment type.
5. Run the mapping to write data to HDFS.

Flat File Data Object Write Properties

The Data Integration Service uses write properties when it writes data to a flat file. You can edit the format and runtime write properties on the **Advanced** tab.

The following table describes the HDFS connection and compression properties that you configure for flat file targets:

Property	Description
Connection Type	The type of connection. Select from the following options: - None. The target file does not require a connection. The target file location is specified by the output file directory. - Hadoop File System. The target file is in HDFS. Default is None.
Connection Name	The name of the connection. Select an HDFS connection or assign a mapping parameter that defines the connection details.
Compression Format	Optional. Specifies the compression format. Select from the following options: - None - Gzip - Bzip2 - Lzo - Custom
Compression Codec	Required for custom compression. Specify the fully qualified class name implementing the Hadoop `CompressionCodec` interface.

Complex File Streaming

To write data to a complex file, include a Data Processor transformation in the mapping to convert the source data into a binary format. You can use the binary stream to write data to the complex file.

The Data Processor transformation continually streams and sends input to the complex file target. It sends end of file information after it fully streams a file. It sends end of streaming information when it streams the entire input fully.

When the Data Processor transformation sends portions of the input to the complex file target, PowerExchange for HDFS appends unique identifier information to the file name. The Data Integration Service uses the unique identifiers to recognize that the streaming is in progress and not complete. Therefore, the file name that you specify in the complex file write properties is not the same as the output file in HDFS. The output file name in HDFS contains the unique identifier information as well.

The unique identifier format depends on whether the file is not compressed or not. The following table describes the unique identifier format based on whether the file is compressed or not:

Run-time Environment Type	File Type	Unique Identifier Format
Native	Uncompressed File	<filename>_<unique identifier>_<seq>.<ext>
Native	Compressed File	<filename>_<unique identifier>_<seq>.<compression format extension>

If you do not include the compression format extension as part of the file name in the complex file write properties, PowerExchange for HDFS appends extensions based on the compression format.

The following table describes the extensions that PowerExchange for HDFS appends based on the compression format that you use:

Compression Format	File Name Extension that PowerExchange for HDFS Appends
DEFLATE	.deflate
Gzip	.gz
Bzip2	.bz2
Lzo	.lzo
Snappy	.snz

Complex Files Output Collection Mode

When you write data to complex files, you can choose to collect the input rows and write the output to a single file, or create an output row for each input row.

You can specify the output collection mode in the Data Processor transformation based on the complex file type.

To specify the output collection mode in the Data Processor transformation, open the Data Processor transformation and click the **Settings** view. In the **Binary output collection mode** section, specify the output collection mode.

The following table describes the options that you can select for the output collection mode:

Property Name	Property Description
Collect input rows to a single output	Select this option if you want to collect all input rows and write the output to a single file.
Split output when size exceeds	When you write the output to a single file, you can choose to split the output file when it exceeds a particular size. Enter the size in MB exceeding which the file must be split. Default is 100 MB.
Output row for each input row (do not collect)	Select this option if you want to write an output row for each input row.

Output Collection Mode for Binary Files

When you write to binary files in a native or Hadoop environment, you can specify the output collection mode in the Data Processor transformation.

Output Collection Mode for Sequence Files and Custom Output Format Files

When you write to sequence files or custom output format files in a native environment, PowerExchange for HDFS writes all the key-value pairs into one output file. The number of key-value pairs that PowerExchange for HDFS writes depends on the output collection mode that you specified in the Data Processor transformation.

Complex File Data Object Write Properties

The Data Integration Service uses write properties when it writes data to a complex file. Select the Input transformation to edit the general, ports, sources, and advanced properties.
Note: Though the FileName port is displayed by default when you create a data object write operation, the FileName port is not supported for the data object write operation.

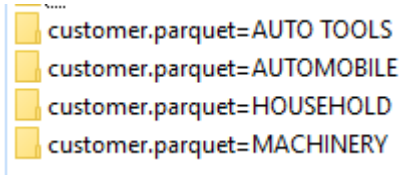
Overwriting Complex File Targets

When you run a mapping in the native environment or on the Spark engine to write data to a complex file target, you can choose to overwrite the target data. You can select the **Overwrite Target** option in the advanced write properties of the complex file data object.

If you select the **Overwrite Target** option, the Data Integration Service deletes the target data before writing data. If you do not select this option, the Data Integration Service creates a new file in the target and writes the data to the file.

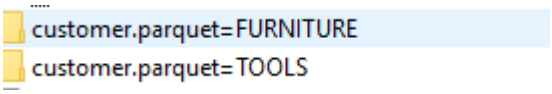
If you select the **Overwrite Target** option, the Data Integration Service deletes all the files and folders in the target directory that are prefixed with the target file name.

For example, the following image shows target data with Overwrite Fileport connected:



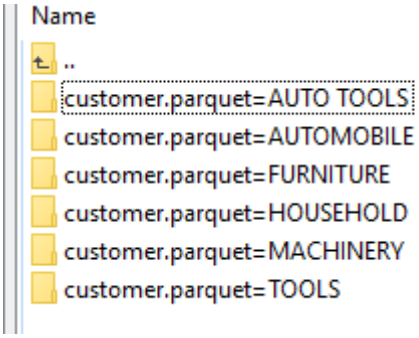
If you select the **Overwrite Target** option, the Data Integration Service deletes all the folders and files in the target directory that are prefixed with the target file name.

The following image shows the file structure after using the **Overwrite Target** option:



If you do not use the **Overwrite Target** option, the Data Integration Service adds the data to the existing files and folders in the target directory.

The following image shows the file structure when you append data:



To avoid unnecessary deletion of files and folders, you must verify that the target directory does not contain any folder or file with the same name as the target file name.

General Properties

The Developer tool displays general properties for complex file targets in the **Write** view.

The following table describes the general properties that you configure for complex file targets:

Property	Description
Name	The name of the complex file. This property is read-only. You can edit the name in the **Overview** view. When you use the complex file as a target in a mapping, you can edit the name in the mapping.
Description	The description of the complex file.

Port Properties

Port properties for a physical data object include port names and port attributes such as data type and precision.

Note: The port size specified in the target transformation and Input transformation must be the same.

The following table describes the ports properties that you configure for complex file targets:

Property	Description
Name	The name of the resource.
Type	The native data type of the resource.
Precision	The maximum number of significant digits for numeric data types, or the maximum number of characters for string data types.
Description	The description of the resource.

Target Properties

The Developer tool displays the sources properties for complex file targets in the Input transformation in the **Write** view.

The sources properties list the resources of the complex file data object. You can add or remove resources in the data object.

Advanced Properties

The Developer tool displays the advanced properties for complex file targets in the Input transformation in the **Write** view.

The following table describes the advanced properties that you configure for complex file targets:

Property	Description
File Directory	The directory location of the complex file target. If the directory is in HDFS, enter the path without the node URI. For example, `/user/lib/testdir` specifies the location of a directory in HDFS. The path must not contain more than 512 characters. If the directory is in the local system, enter the fully qualified path. For example, `/user/testdir` specifies the location of a directory in the local system. **Note:** The Data Integration Service ignores any subdirectories and their contents.
File Name	The name of the output file. PowerExchange for HDFS appends the file name with a unique identifier before it writes the file to HDFS. In spark mode PowerExchange for HDFS appends the file name with .avro extension.
Overwrite Target	Indicates whether the Data Integration Service must first delete the target data before writing data. If you select the **Overwrite Target** option, the Data Integration Service deletes the target data before writing data. If you do not select this option, the Data Integration Service creates a new file in the target and writes the data to the file. This option is applicable when you run a mapping in the native environment or on the Spark engine to write data to complex files.
File Format	The file format. Select one of the following file formats: - Binary. Select Binary to write any file format. - Sequence. Select Sequence File Format for target files of a Hadoop-specific binary format that contain key and value pairs. - Custom Output. Select Output Format to specify a custom output format. You must specify the class name implementing the `OutputFormat` interface in the **Output Format** field. Default is Binary. **Note:** Binary file format is not supported for the complex file data object operation in spark mode.
Output Format	The class name for files of the output format. If you select Output Format in the **File Format** field, you must specify the fully qualified class name implementing the `OutputFormat` interface.
Output Key Class	The class name for the output key. If you select Output Format in the **File Format** field, you must specify the fully qualified class name for the output key. You can specify one of the following output key classes: - BytesWritable - Text - LongWritable - IntWritable **Note:** PowerExchange for HDFS generates the key in ascending order.

Property	Description
Output Value Class	The class name for the output value. If you select Output Format in the **File Format** field, you must specify the fully qualified class name for the output value. You can use any custom writable class that Hadoop supports. Determine the output value class based on the type of data that you want to write. **Note:** When you use custom output formats, the value part of the data that is streamed to the complex file data object write operation must be in a serialized form.
Compression Format	Optional. The compression format for binary files. Select one of the following options: - None - Auto - DEFLATE - gzip - bzip2 - LZO - Snappy - Custom
Custom Compression Codec	Required for custom compression. Specify the fully qualified class name implementing the `CompressionCodec` interface.
Sequence File Compression Type	Optional. The compression format for sequence files. Select one of the following options: - None - Record - Block

Column Projection Properties

The Developer tool displays the column projection properties for Avro, JSON, and Parquet complex file targets in the Properties view of the **Write** operation.

The following table describes the advanced properties that you configure for Avro, JSON, and Parquet complex file targets:

Property	Description
Enable Column Projection	Displays the column details of the complex files sources.
Schema Format	Displays the schema format that you selected while creating the complex file data object. You can change the schema format and provide respective schema. You can select one of the following complex file formats: - Avro - JSON - ORC - Parquet You can change the complex file format without losing the column metadata even after you configure the column projection properties for another complex file format. **Note:** You can switch from one schema format to another only once. If you change the schema format more than once, you might lose the original datatypes.

Property	Description
Schema	Displays the schema associated with the complex file. You can select a different schema. **Note:** If you disable the column projection, the schema associated with the complex file is removed. If you want to associate schema again with the complex file, enable the column projection and click Select Schema.
Column Mapping	Displays the mapping between input and output ports. **Note:** If you disable the column projection, the mapping between input and output ports is removed. If you want to map the input and output ports, enable the column projection and click Select Schema to associate a schema to the complex file.
Project Column as Complex Data Type	Displays columns with hierarchical data as a complex data type, such as, array, map, or struct. Select this property when you want to process hierarchical data on the Spark engine. **Note:** If you disable the column projection, the data type of the column is displayed as binary type.

CHAPTER 7

HDFS Mappings

This chapter includes the following topics:

HDFS Mappings Overview

After you create a flat file or a complex file data object operation, you can create an HDFS mapping.

You can define the following objects in an HDFS mapping:

- A flat file data object or a complex file data object read operation as the input to read data from HDFS
- Transformations
- A flat file data object or a complex file data object write operation as the output to write data to HDFS

If you use a complex file data object as a source, you must use a Data Processor transformation to parse the file. Similarly, when you use a complex file data object as a target, you must use a Data Processor transformation to convert the source data into a binary format. You can then use the binary stream to write data to the complex file.

You can use complex file sources and targets as dynamic sources and targets in a mapping. For information about dynamic mappings, see the *Informatica Developer Mapping Guide*.

Validate and run the mapping. You can deploy the mapping and run it or add the mapping to a Mapping task in a workflow and run the workflow. You can also run the mapping in a Hadoop run-time environment.

Complex Files Target Creation

You can create a complex file target from an existing transformation in the mapping. The Developer tool can create an Avro, Parquet, ORC, or JSON complex file target.

You cannot use the **Create Target** option to create a complex file target from an existing transformation in the following scenarios:

- The existing transformation contains a port named FileName.
- The existing transformation points to a Hive table that contains a complex data type with hierarchical data.
- The existing transformation points to an Avro complex file that contains field names with special characters.

Creating a Complex File Target from an Existing Transformation

If one of the ports in the existing transformation contains a complex port, you must create a complex file target and link ports by name or link ports at run time based on the link policy.

1. Open a complex file mapping in the editor.
2. Right-click a transformation in the mapping editor and select **Create Target**.

   The **Create Target** window opens.
3. Choose the complex file data object type.
4. Choose a link type.

   You can choose from the following link types:

   Link ports by name

   Ports in the Write transformation correspond to those in the source and have the same names.

   Link dynamic port based on the mapping flow

   The Write transformation contains dynamic ports based on upstream objects in the mapping flow.

   Link ports at run time based on link policy

   Ports are created in the target at run time based on the link policy that you configure on the **Run Time Linking** tab of the Write transformation.

   For more information about dynamic ports and run-time link configuration, see the *Informatica Developer Mapping Guide.*
5. Name the new complex file data object.
6. Optionally, click **Browse** to select a location for the data object.
7. Select the complex file format as Avro, Parquet, Orc, or Json in the **Resource Format** list.
8. Click **Finish**.

The Developer tool performs the following tasks:

- Adds a complex file Write transformation to the mapping.
- Links ports.

- Creates a physical data object.
  You can configure the physical data object properties. For example, you must specify an HDFS connection for the complex file data object.

Mapping Validation and Run-time Environments

You can use flat file and complex file data objects in a Hadoop run-time environment.

You can configure the mappings to run in native or Hadoop run-time environments. When you run a mapping in the native environment, the Data Integration Service processes the mapping. When you run a mapping in a Hadoop environment, the Data Integration Service can push mappings to a Hadoop cluster. You can run an HBase mapping on the Blaze or Spark engine.

For more information about the native and Hadoop environments, see the *Informatica Big Data Management User Guide*.

Rules and Guidelines for Complex File Targets in a Mapping

Consider the following rules and guidelines for complex file targets in a mapping:

- You cannot use the **Create Target** option to create a complex file target from an existing transformation in the following scenarios:
  - The existing transformation contains a port named FileName.
  - The existing transformation points to a Hive table that contains a complex data type with hierarchical data.
  - The existing transformation points to an Avro complex file that contains field names with special characters.
- If you select the **Overwrite Target** option, the Data Integration Service deletes all the files and folders in the target directory that are prefixed with the target file name.

  To avoid unnecessary deletion of files and folders, you must verify that the target directory does not contain any folder or file with the same name as the target file name.

HDFS Data Extraction Mapping Example

Your organization needs to analyze purchase order details such as customer ID, item codes, and item quantity. The purchase order details are stored in a semi-structured compressed XML file in HDFS. The hierarchical data includes a purchase order parent hierarchy level and a customer contact details child hierarchy level. Create a mapping that reads all the purchase records from the file in HDFS. The mapping must convert the hierarchical data to relational data and write it to a relational target.

You can use the extracted data for business analytics.

The following figure shows the example mapping:

You can use the following objects in the HDFS mapping:

HDFS Input

The input object, Read_Complex_File, is a Read transformation that represents a compressed XML file stored in HDFS.

Data Processor Transformation

The Data Processor transformation, Data_Processor_XML_to_Relational, parses the XML file and provides a relational output.

Relational Output

The output object, Write_Relational_Data_Object, is a Write transformation that represents a table in an Oracle database.

When you run the mapping, the Data Integration Service reads the file in a binary stream and passes it to the Data Processor transformation. The Data Processor transformation parses the specified file and provides a relational output. The output is written to the relational target.

You can configure the mapping to run in a native or Hadoop run-time environment.

Complete the following tasks to configure the mapping:

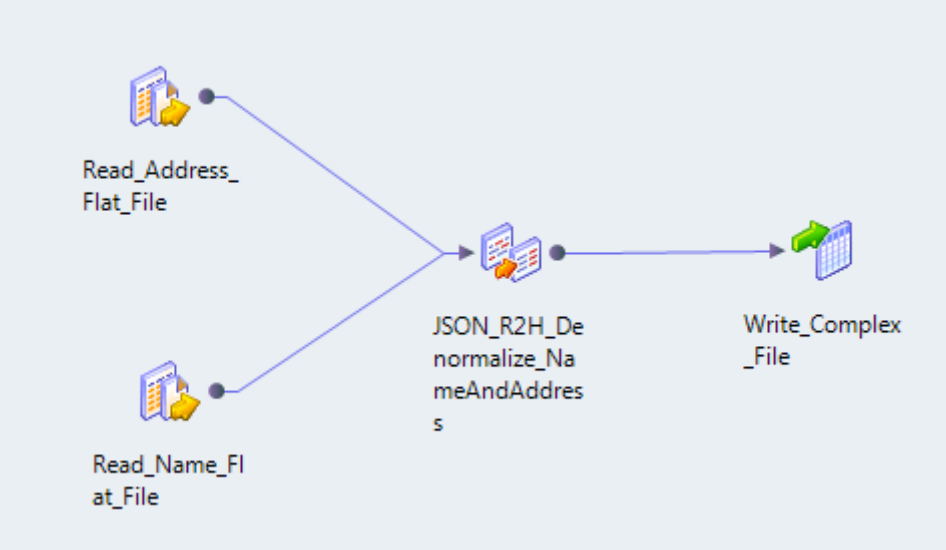
1. Create an HDFS connection to read files from the Hadoop cluster.
2. Create a complex file data object read operation. Specify the following parameters:
   - The file as the resource in the data object.
   - The file compression format.
   - The HDFS file location.
3. Optionally, you can specify the input format that the Mapper uses to read the file.
4. Drag and drop the complex file data object read operation into a mapping.
5. Create a Data Processor transformation. Configure the following properties in the Data Processor transformation:
   - An input port set to buffer input and binary data type.
   - Relational output ports depending on the number of columns you want in the relational output. Specify the port size for the ports. Use an XML schema reference that describes the XML hierarchy. Specify the normalized output that you want. For example, you can specify PurchaseOrderNumber_Key as a generated key that relates the Purchase Orders output group to a Customer Details group.
   - Create a Streamer object and specify Streamer as a startup component.
6. Create a relational connection to an Oracle database.
7. Import a relational data object.
8. Create a write transformation for the relational data object and add it to the mapping.

HDFS Data Load Mapping Example

Your organization needs to denormalize employee ID, name, and address details. The employee ID, name, and address details are stored in flat files in HDFS. Create a mapping that reads all the employee ID, name, and address details from the flat files in HDFS. The mapping must convert the denormalized data to hierarchical data and write it to a complex file target in HDFS.

You can use the target data for business analytics.

The following figure shows the example mapping:



You can use the following objects in the HDFS mapping:

HDFS Inputs

The inputs, Read_Address_Flat_File and Read_Name_Flat_File, are flat files stored in HDFS.

Data Processor Transformation

The Data Processor transformation, JSON_R2H_Denormalize_NameAndAddress, reads the flat files, denormalizes the data, and provides a binary, hierarchical output.

HDFS Output

The output, Write_Complex_File, is a complex file stored in HDFS.

When you run the mapping, the Data Integration Service reads the input flat files and passes the employee ID, name, and address data to the Data Processor transformation. The Data Processor transformation denormalizes the employee ID, name, and address data, and provides a hierarchical output in a binary stream. The binary and hierarchical output is written to the HDFS complex file target.

You can configure the mapping to run in a native or Hadoop run-time environment.

Complete the following tasks to configure the mapping:

1. Create an HDFS connection to read flat files from the Hadoop cluster.
2. Specify the read properties for the flat files.
3. Drag and drop the flat files into a mapping.
4. Create a Data Processor transformation. Set the Data Processor transformation port to binary.
5. Create an HDFS connection to write data to the complex file target.

6. Create a complex file data object write operation. Specify the following parameters:
   - The file as the resource in the data object.
   - The HDFS file location.
7. Drag and drop the complex file data object write operation into the mapping.

HDFS Avro Read Mapping Example

Your organization needs to denormalize customer key, name, address, and other details. The customer details are stored in Avro files in HDFS. Import the Avro file object as a source. Create a mapping that reads all the customer details from the avro files in HDFS, and writes the customers details to an Oracle target.

You can use the target data for business analytics.

You can use the following objects in the HDFS mapping:

HDFS Inputs

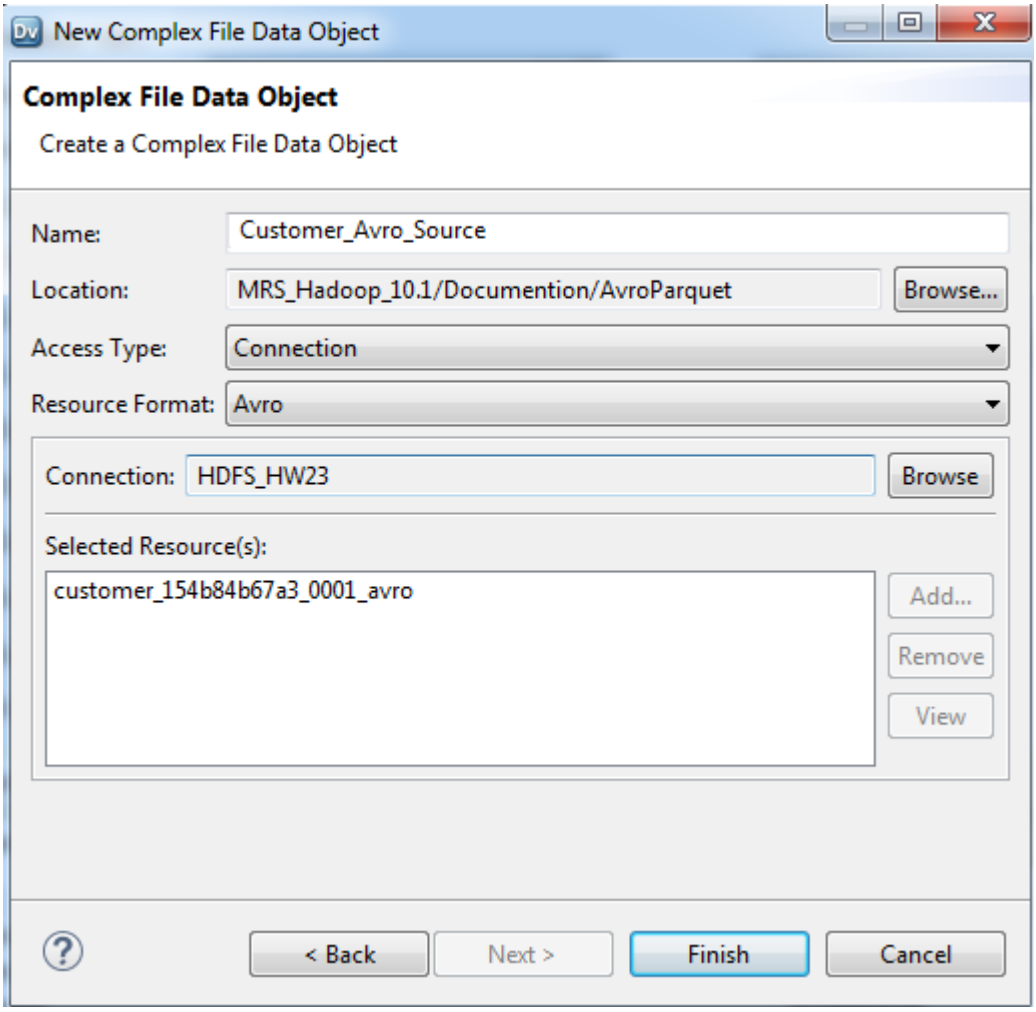
The Customer_Details_Avro file is an Avro files stored in HDFS.

HDFS Output

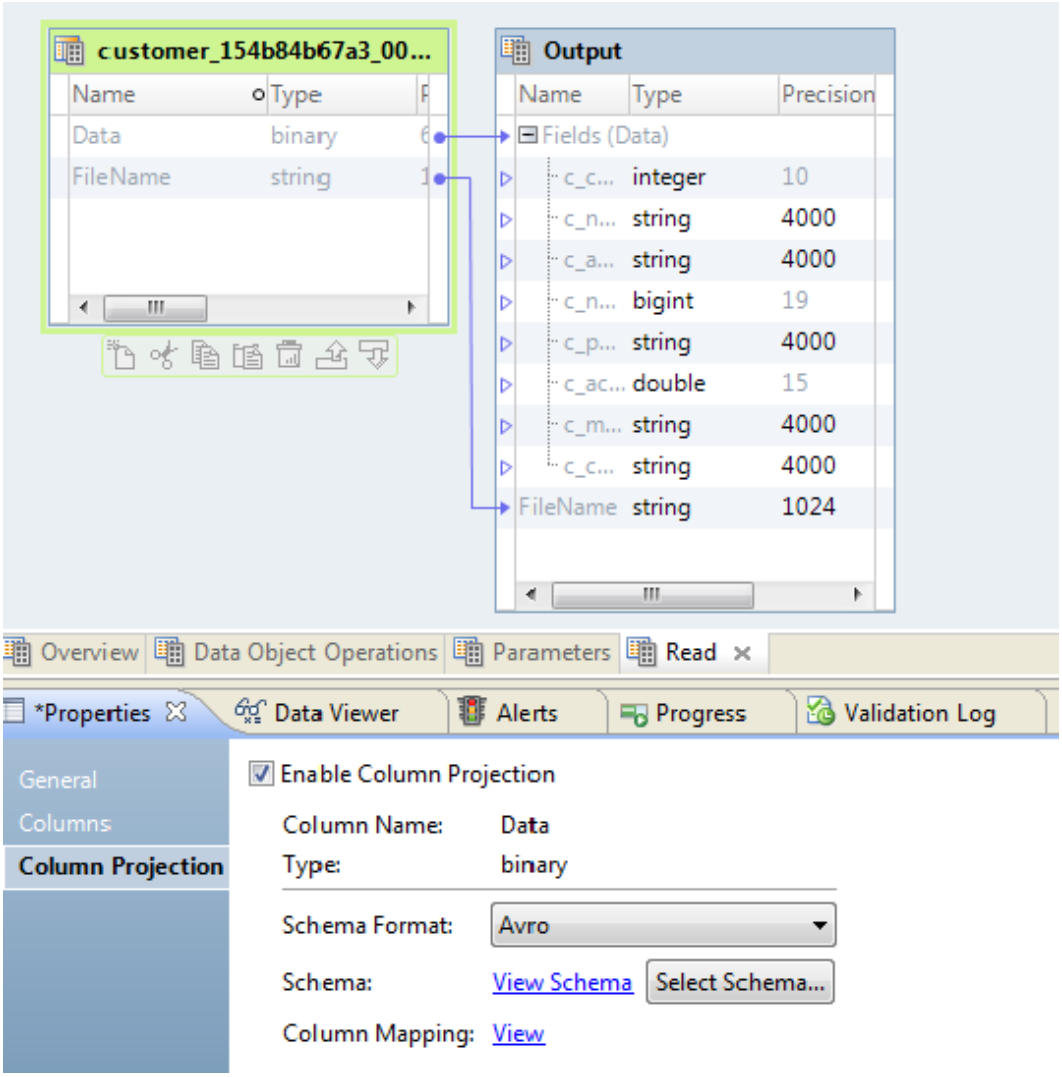
The Customer_Oracle_Target file is an Oracle object.

Create a Complex File Data Object

Create a complex file data object to read data from an Avro file. Verify that you select Avro as Resource Format. The following image shows the sample selection:

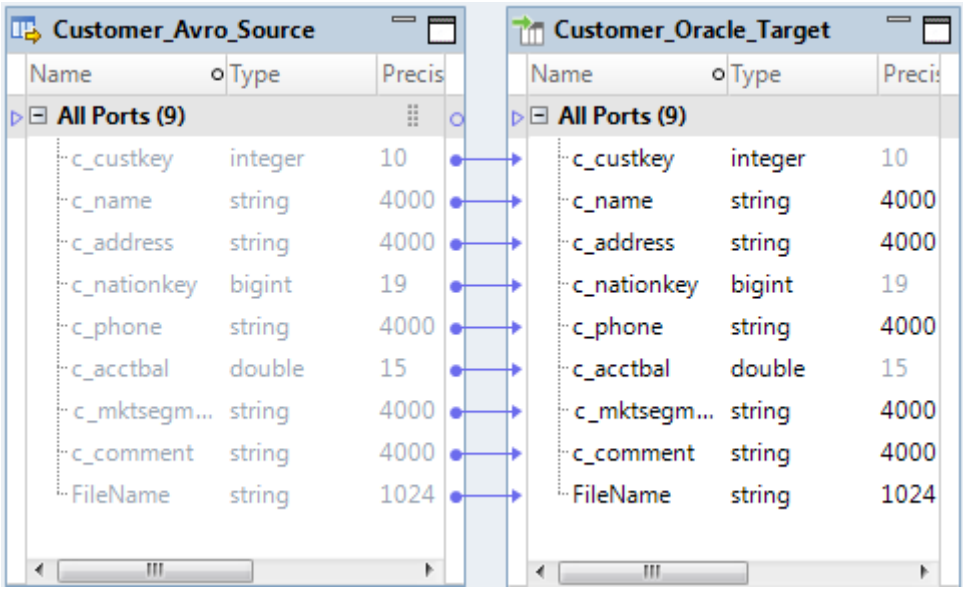


When you create the complex file data object, the read and write operations are created by default. You can view the columns present in the Avro file. The following image shows the sample data object read operation:



The Enable Column Projection is selected by default. You can view or update the associated schema and column mapping.

The following image shows the sample mapping:



When you run the mapping, the Data Integration Service reads the input Avro files and writes the hierarchical output directly to the Oracle target.

You can configure the mapping to run in a native or Hadoop run-time environment.

Perform the following tasks to configure the mapping:

1. Create an HDFS connection to read Avro file from the Hadoop cluster.
2. Create a complex file data object to import the Avro file. You must select Avro as Resource Format. Configure the read operation properties.
3. Create an Oracle database connection to write data to the Oracle target.
4. Create an Oracle data object and configure the write operation properties.
5. Drag the complex file data object read operation and Oracle data object write operation into the mapping.
6. Map ports and run the mapping.

APPENDIX A

Data Type Reference

This appendix includes the following topics:

Data Type Reference Overview

Informatica Developer uses the following data types for HDFS data objects:

- Flat file data types. Flat file data types appear in the physical data object column properties.
- Complex file data types. Complex file data types appear in the physical data object column properties.
- Transformation data types. Set of data types that appear in the transformations. They are internal data types based on ANSI SQL-92 generic data types, which the Data Integration Service uses to move data across platforms. Transformation data types appear in all transformations in a mapping.

When the Data Integration Service reads source data, it converts the native data types to the comparable transformation data types before transforming the data. When the Data Integration Service writes to a target, it converts the transformation data types to the comparable native data types.

Flat File and Transformation Data Types

Flat file data types map to transformation data types that the Data Integration Service uses to move data across platforms.

The following table compares flat file data types to transformation data types:

Flat File Data type	Transformation Data type	Range
Bigint	Bigint	Precision of 19 digits, scale of 0
Datetime	Date/Time	Jan 1, 0001 A.D. to Dec 31, 9999 A.D. (precision to the nanosecond)
Double	Double	Precision of 15 digits
Int	Integer	-2,147,483,648 to 2,147,483,647
Nstring	String	1 to 104,857,600 characters
Number	Decimal	For transformations that support precision up to 38 digits, the precision is 1 to 38 digits, and the scale is 0 to 38. For transformations that support precision up to 28 digits, the precision is 1 to 28 digits, and the scale is 0 to 28. If you specify the precision greater than the maximum number of digits, the Data Integration Service converts decimal values to double in high precision mode.
String	String	1 to 104,857,600 characters

When the Data Integration Service reads non-numeric data in a numeric column from a flat file, it drops the row and writes a message in the log. Also, when the Data Integration Service reads non-datetime data in a datetime column from a flat file, it drops the row and writes a message in the log.

Complex File and Transformation Data Types

Complex file data types map to transformation data types that the Data Integration Service uses to move data across platforms.

The following table lists the complex file data types that the Data Integration Service supports and the corresponding transformation data types:

Complex File Data Type	Transformation Data Type	Range and Description
Binary	Binary	1 to 104,857,600 bytes. You can read and write data of Binary data type in a Hadoop environment. You can use the user-defined functions to transform the binary data.

Avro Data Types and Transformation Data Types

Avro data types map to transformation data types that the Data Integration Service uses to move data across platforms.

The following table compares the Avro data types that the Data Integration Service supports and the corresponding transformation data types:

Avro Data Type	Transformation Data Type	Range
Array	Array	Unlimited number of characters
Boolean	Integer	TRUE (1) or FALSE (0)
Bytes	Binary	Precision 4000
Double	Double	Precision 15
Fixed	Binary	1 to 104,857,600 bytes
Float	Double	Precision 15
Int	Integer	-2,147,483,648 to 2,147,483,647 Precision 10, scale 0
Long	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Precision 19, scale 0
Map	Map	Unlimited number of characters
Record	Struct	Unlimited number of characters
String	String	1 to 104,857,600 characters
Union	Corresponding data type in a union of ["primitive_type|complex_type", "null"] or ["null", "primitive_type| complex_type"].	Dependent on primitive or complex data type.

Avro Union Data Type

A union indicates that a field might have more than one data type. For example, a union might indicate that a field can be a string or a null. A union is represented as a JSON array containing the data types.

The Developer tool only interprets a union of ["primitive_type|complex_type", "null"] or ["null", "primitive_type| complex_type"]. The Avro data type converts to the corresponding transformation data type. The Developer tool ignores the null.

Unsupported Avro Data Types

The Developer tool does not support the following Avro data types:

- enum
- null

Intelligent Structure Model Data Types and Transformation Types

Intelligent structure model data types map to transformation data types that the Data Integration Service uses to move data across platforms.

The following table lists the intelligent structure model data types that the Data Integration Service supports and the corresponding transformation data types:

Intelligent Structure Model Data Type	Transformation Data Type	Range and Description
Array	Array	Unlimited number of characters.
Bigint	Bigint	Precision of 19 digits, scale of 0.
Datetime	Date/Time	Jan 1, 0001 A.D. to Dec 31, 9999 A.D. (precision to the nanosecond)
Double	Double	Precision of 15 digits.
Int	Integer	-2,147,483,648 to 2,147,483,647.
Number	Decimal	Precision 1 to 28, scale 0 to 28
String	String	1 to 104,857,600 characters
Struct	Struct	Unlimited number of characters.

JSON Data Types and Transformation Data Types

JSON data types map to transformation data types that the Data Integration Service uses to move data across platforms.

The following table compares the JSON data types that the Data Integration Service supports and the corresponding transformation data types:

JSON	Transformation	Range
Array	Array	Unlimited number of characters.
Double	Double	Precision of 15 digits
Integer	Integer	-2,147,483,648 to 2,147,483,647 Precision of 10, scale of 0

JSON	Transformation	Range
Object	Struct	Unlimited number of characters.
String	String	1 to 104,857,600 characters

Unsupported JSON Data Types

The Developer tool does not support the following JSON data types:

- date/timestamp
- enum
- union

Parquet Data Types and Transformation Data Types

Parquet data types map to transformation data types that the Data Integration Service uses to move data across platforms.

The following table compares the Parquet data types that the Data Integration Service supports and the corresponding transformation data types:

Parquet	Transformation	Range
Binary	Binary	1 to 104,857,600 bytes
Binary (UTF8)	String	1 to 104,857,600 characters
Boolean	Integer	-2,147,483,648 to 2,147,483,647 Precision of 10, scale of 0
Double	Double	Precision of 15 digits
Fixed Length Byte Array	Decimal	Decimal value with declared precision and scale. Scale must be less than or equal to precision. For transformations that support precision up to 38 digits, the precision is 1 to 38 digits, and the scale is 0 to 38. For transformations that support precision up to 28 digits, the precision is 1 to 28 digits, and the scale is 0 to 28. If you specify the precision greater than the maximum number of digits, the Data Integration Service converts decimal values to double in high precision mode. **Note:** When you run a mapping in the native environment with decimal data type in the source the mapping runs successfully but the data is not written.

Parquet	Transformation	Range
Float	Double	Precision of 15 digits
group (LIST)	Array	Unlimited number of characters.
Int32	Integer	-2,147,483,648 to 2,147,483,647 Precision of 10, scale of 0
Int64	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Precision of 19, scale of 0
Map	Map	Unlimited number of characters.
Struct	Struct	Unlimited number of characters.
Union	Corresponding primitive data type in a union of ["primitive_type", "null"] or ["null", "primitive_type"].	Dependent on primitive data type.

The Parquet schema that you specify to read or write a Parquet file must be in smaller case. Parquet does not support case-sensitive schema.

Parquet Union Data Type

A union indicates that a field might have more than one data type. For example, a union might indicate that a field can be a string or a null. A union is represented as a JSON array containing the data types.The Developer tool only interprets a union of ["primitive_type", "null"] or ["null", "primitive_type"]. The Parquet data type converts to the corresponding transformation data type. The Developer tool ignores the null.

Unsupported Parquet Data Types

The Developer tool does not support the following Parquet data types:

- int96 (TIMESTAMP_MILLIS)
- date
- time
- timestamp

INDEX

R

W