TIBCO Flogo® Connector for Data Conversion

User's Guide

Software Release 1.0
August 2020

Important Information

Contents

TIBCO Documentation and Support Services

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit https://docs.tibco.com.

Product-Specific Documentation

The following documentation for this product is available on the TIBCO Flogo® Connector for Data Conversion Product Documentation page.

- *TIBCO Flogo® Connector for Data Conversion Release Notes*
- *TIBCO Flogo® Connector for Data Conversion Installation*
- *TIBCO Flogo® Connector for Data Conversion User's Guide*

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit http://www.tibco.com/services/support.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at https://support.tibco.com.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to https://support.tibco.com. If you do not have a user name, you can request one by clicking Register on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to https://community.tibco.com.

Overview

COBOL copybooks describe the data structure for the corresponding binary data. You can interpret or create COBOL data according to the specified copybook using TIBCO Flogo® Connector for Data Conversion.

With this connector, you can interact with applications that produce or consume COBOL data. To process COBOL data in a flow, the data must be parsed into JSON format. If a flow needs to supply COBOL data for an external application, JSON data must be mapped from the output of activities and rendered into COBOL form.

The Data Conversion connector consists of the following components:

- COBOL Copybook Connection: This connection contains a COBOL copybook and generates a JSON schema used in input and output mappings of activities.
- Parse Copybook Data Activity: This activity converts incoming COBOL data to JSON. It parses incoming data according to the selected Copybook connection. After parsing, the output data conforms to the schema generated by the Copybook connection, and other activities can access the data.
- Render Copybook Data Activity: This activity converts existing JSON data into COBOL output. It accepts data from other activities and produces data according to the COBOL copybook held in the selected Copybook connection. The COBOL data can then be passed to an application. Flogo flows can use other activities to carry out the data delivery.

Creating a COBOL Copybook Connection

You need to create a Copybook connection to define a copybook layout. It contains COBOL copybook source code and creates a JSON schema that is used by Parse and Render activities.

Prerequisites

You can edit the COBOL code and validate the correctness using the connection resource.

Procedure

1. Click **Create** on Connection page.
2. In the **Copybook**, paste or type the source code of your COBOL copybook. Ensure no errors are shown; edit the copybook if necessary.
3. Click **Save**.

COBOL Copybook Connection Details

The COBOL Copybook dialog box contains the following fields:

Field	Description
Copybook Name	The unique name for the copybook. This is displayed in the Copybook drop-down list for the Parse and Render activities.
Description	A short description of copybook. This is an optional field.
Fixed Format	The field that specifies whether the content is in fixed or free format: • If this field is selected, the data is to be in COBOL fixed format; that is, columns 0-6 are reserved for line numbers. The content is expected to end at or before column 72. • Disable this option if the input data is in COBOL free format; that is, the COBOL code can be within columns 0-6 and can span beyond column 72.
Floating Point	Floating point format for `COMP-1` and `COMP-2`. For more information, see Handling of Floating Point Items.
Day First	Determines the order of day and month in certain date formats. For more information, see Date Formats.
Base year for two-digit year	Base year to apply to two-digit year DATE FORMAT items. For more information, see Two-Digit Year in DATE FORMAT Clause
TRUNC(BIN)	When this option is selected, all COMP(BINARY) items are treated as `COMP-5`. Valid values are not limited by the `PICTURE` clause but rather by the full range of the underlying storage size.
Copybook	The source text of a COBOL copybook. This field is editable and is validated.
Errors/Warnings	This read-only field shows errors and warnings that may be present in the copybook text. When none are detected this field is hidden.

Field	Description
Copybook Schema	This read-only field shows the JSON schema that represents the copybook. If errors are detected, the **Copybook Schema** field is not displayed as a schema cannot be generated.

Language Features

The Data Conversion connector supports most, though not all, COBOL features.

The COBOL capability of the connector is tested against IBM Enterprise COBOL for z/OS 3.4.1 (5655-G53). Other compilers might produce incompatible data layouts.

Supported Usages

Usage	Notes
BINARY COMP, COMPUTATIONAL, COMP-4, COMPUTATIONAL-4, COMP-5, COMPUTATIONAL-5	For more information, see Handling of Computation Items.
COMP-3, COMPUTATIONAL-3 PACKED-DECIMAL, COMP-6, COMPUTATIONAL-6	For more information, see Handling of Computation Items.
DISPLAY	Depending on the picture type and the Copybook Connection settings, the connector converts the data items with DISPLAY usage to **string**, **integer**, or **number** type in the schema.
COMP-1, COMPUTATIONAL-1	The Connector converts items with this usage to the **number** type in the schema.
COMP-2, COMPUTATIONAL-2	The Connector converts items with this usage to the **number** type in the schema.
FUNCTION-POINTER	The Connector converts items with FUNCTION-POINTER usage to the **integer** type in the schema.
INDEX	The Connector converts items with INDEX usage to the **integer** type in the schema.
POINTER	The Connector converts items with POINTER usage to the **integer** type in the schema.
PROCEDURE-POINTER	The Connector converts items with PROCEDURE-POINTER usage to the **integer** type in the schema.

Supported Picture Types

Picture Type	Example	Notes
Alphabetic	PIC AAA PIC A(12)	The Connector converts the data item with this picture type to the **string** type in the schema.

Picture Type	Example	Notes
Alphanumeric	PIC XX PIC X(10)	The Connector converts the data item with this picture type to the **string** type in the schema.
Alphanumeric-edited	PIC XX/XX/XX PIC XXXBX(3)BX(4)	The Connector converts the data item with this picture type to the **string** type in the schema.
Numeric-edited	PIC 99/99/99 PIC -9(7).99 PIC +,++ +,999.00	The Connector converts the data item with this picture type to the **string** type in the schema. However, you can configure the connector to represent it as type **number** by using `@parseNumeric` annotation.
External Floating Point	PIC +99.9999E+99 PIC -9.99999E-99	The Connector converts the data item with this picture type to the **string** type in the schema. However, you can instruct the connector to represent it as type **number** by using `@parseNumeric` annotation.
Numeric	PIC 999 PIC 9(5) PIC S9(8) PIC S9(5)V99	If a picture of the data item contains a decimal point, the Connector uses the **number** type in the schema. Otherwise **integer** type is used.

Supported COBOL Clauses and Features

Clause	Notes
DATE FORMAT	This clause specifies that an item is a date field. For more information, see DATE FORMAT Clause.
OCCURS	This clause specifies that the item repeats a fixed number of times. In the schema, such items are represented as type **array**.
OCCURS … DEPENDING ON ...	This clause specifies that the item repeats a number of times defined by the value of another data item. For more information , see OCCURS min To max TIMES DEPENDING ON object Clause. In the schema, such items are represented as type **array**.
REDEFINES	This clause specifies that two or more items occupy the same storage. The connector aggregates these items as properties of an object named **redefineGroupN**. JSON data of this object can only contain one property.
SIGN	This clause specifies the position and mode of representation of the operational sign for the signed numeric item.
SYNCHRONIZED	This clause specifies an elementary item’s alignment on a natural boundary in storage.

Clause	Notes
VALUE	This clause specifies a data item's initial content. The Render Copybook Data activity generates that value for items whose values are not specified in the input data.

DATE FORMAT Clause

The COBOL **`DATE FORMAT`** clause specifies that a data item is a date field. Some formats do not specify whether day or month comes first. In such cases this is determined by the **Day First** field of the Copybook connection.

In JSON date fields are represented by strings in `YYYY-MM-DD` format.

Not all COBOL formats contain full year, month, and day. For example, `YY` only contains a two-digit year. In such cases, when the date is parsed, the missing date parts are defaulted to the lowest values: day 1, year 0001, and month of January.

When one of the incomplete formats is rendered, the date parts in JSON that are absent in COBOL DATE FORMAT are ignored.

Two-Digit Year

Two-digit year format is processed in accordance with the **Base year for two-digit year** setting in the Copybook connection. This value acts as the base-year in **`YEARWINDOW(base-year)`** option of the COBOL compiler.

You must specify it with one of the following values:

- **An integer number between 1900 and 1999:** This specifies the starting year of a fixed window. For example, `YEARWINDOW(1930)` indicates a century window of 1930-2029. Years preceding 30 are considered as years of the 21st century and years starting at 30 belongs to the twentieth century.
- **A negative integer from -1 through -99:** This indicates a sliding window. The first year of the window is calculated by adding the negative integer to the current year. For example, `YEARWINDOW(-80)` indicates that the first year of the century window is 80 years before the year in which the program is run.

DATE FORMAT on a Group Level

The connector does not support `DATE FORMAT` at group level. It is ignored in such cases.

Date Formats

Format	Description	
DATE FORMAT	**Day First** is Selected in Copybook	**Day First** is not selected in Copybook
YY	Two-digit year only	
YYXX	Two-digit year followed by a two-digit month	
YYXXX	Two-digit year followed by a three-digit number of a day in a year	

Format	Description	
YYXXXX	Two-digit year, two-digit day in month, two-digit month	Two-digit year, two-digit month, two-digit day in month
XX	Two-digit day in month	Two-digit month
XXX	Three-digit number of a day in a year	
XXXX	Two-digit day in a month followed by a two-digit month	Two-digit month followed by a two-digit day in a month
XXYY	Two-digit month followed by a two- digit year	
XXXYY	Three-digit number of a day in a year followed by a two digit year	
XXXXYY	Two-digit day in month, two-digit month, two-digit year	Two-digit month, two-digit day in month, two-digit year
YYYY	Four-digit year	
YYYYXX	Four-digit year followed by two-digit month	
XXYYYY	Two-digit month followed by four-digit year	
YYYYXXXX	Four-digit year, two-digit day in month, two-digit month	Four-digit year, two-digit month, two-digit day in month
YYYYXXX	Four-digit year, followed by three-digit number of a day in a year	
XXXXYYYY	Two-digit day in month, two-digit month, four-digit year	Two-digit month, two-digit day in month, four-digit year
XXXYYYY	Three-digit number of a day in a year followed by a four-digit year	

OCCURS Clause

The COBOL OCCURS clause defines a repeating data item. Its number of occurrences is determined by a numeric item. The OCCURS clause can have fixed number of occurrences or it can be variable if a DEPENDING ON clause is present.

The connector represents items with an OCCURS clause as a JSON array. All items need not be supplied through mapping. Items that are not supplied are rendered using VALUE clause or initialized according to the settings of Render activity. If an input array is larger than the permissible limit in the OCCURS clause, an error is raised.

OCCURS min TO max TIMES DEPENDING ON object

The number of occurrences of an item with this clause depends on the value of the `DEPENDING ON` clause.

Parsing

During parsing, the number of occurrences is determined by the value of the object in the binary data. If the value of the object is not in the **min:max** range an error occurs.

Rendering

The number of occurrences rendered is determined by the value of the object in the input JSON. The number of supplied occurrences need not match the value of the object. Note that the value of the object must comply with **min** and **max** limits or an error occurs.

Handling of Computation Items

Numeric values in COBOL data are commonly stored in display or character format, (that is, as base-10 numbers, with each digit represented by the corresponding character). For example, a field defined as PIC 999 that contains the value 123 is stored in three bytes, each byte containing one digit of the value.

When performing computation with numbers, machines can perform the computations significantly faster on binary (base-2) numbers that base-10 numbers. Therefore, if a number is stored in a COBOL data in binary format, it can be used in computations directly. You can use versions of `COMP` (`COMP-3` `COMP-4`, and so on) to change the storage format from text to binary form.

COMP Versus COMP-5

For integer fields in COBOL, `COMP` specifies the storage of half word, full word, or double word (2, 4, and 8 bytes, respectively). However, an additional limitation is applicable based on the number of decimal digits in the PICTURE clause. For instance, even though `PIC S9(4) COMP` has a signed half-word storage, it limits the values to the range of -9999 through +9999 due to the four decimal places in the `PICTURE` clause.

You can use `COMP-5` to eliminate the limitation. This usage, though the same as `COMP`, allows the whole range of values representable by a particular number of storage bytes. For example, `PIC S9(4) COMP-5` has a half-word storage and allows the range of values of -32768 through +32767.

COMP-6

`COMP-6` is a feature of Micro Focus COBOL. The connector considers it synonymous to `PACKED-DECIMAL`.

`COMP-6` is only supported as `PACKED-DECIMAL`. Other interpretations are not supported.

Handling of Names

`FILLER` has special significance in COBOL. These fields cannot be referenced from a COBOL program and are allowed to occur more than once, (because these names do not have to be unique in a copybook).

The Connector applies the following processing:

Copybook items are not required to have unique names. However, if non-unique names are encountered at the same group level or at the top the level, connector generated unique JSON names by adding numeric suffixes -n. For example, for this copybook:

```
01 ROOT.
```

```
02 ITEM PIC X.
```

```
02 ITEM PIC X.
```

The JSON name of the first elementary item is `ITEM`, but the second item is named `ITEM-0`.

Copybook Annotations

You can customize how the connector processes individual copybook items by inserting annotation comments in copybook text.

Annotations take the following form:

```
* @binary
```

```
01 T1 PIC X
```

Annotations are inserted before the item they apply to. In the above example, `@binary` applies to `T1`.

Annotations do not affect validity of the copybook from the COBOL standpoint. If you enter an annotation incorrectly, the connector displays a warning but ignores the issue if no action is taken.

Annotations offer these capabilities:

- Designating a field as base64 binary
- Turning on or off numeric interpretation
- Setting a default redefine item, adding control values

Copybook Annotations

Annotation	Applicability	Effect
`@binary`	Elementary items	Makes the item a `base64Binary` string. Connector does not apply any validation or conversion to items with this annotation.
`@defaultRedefine`	One of the items in a REDEFINE group	Designates an item as default in a `REDEFINE` group.
`@controlField:{item name}`	A redefined item	Assigns a control field to an item in a `REDEFINE` group.
`@controlValues`	One of the items in a REDEFINE group.	Assigns a set of control values to an item in a `REDEFINE` group.
`@parseNumeric`	A numeric edited item	Turns on editing/de-editing of a numeric-edited item.

Handling of Numeric-Edited and External Floating-Point Items

Numeric-edited and external floating-point items in COBOL are numeric items that are represented as formatted text, which is formed according to the `PICTURE` clause editing rules. You can select the processing type to apply using `@parseNumeric` annotation.

- When `@parseNumeric` is used, the connector represents the item as a JSON number and converts to and from the underlying edited representation. Your flow uses algebraic values that are independent of specific editing applied by COBOL. During parsing, irregularities of the edited data cause an error.
- If `@parseNumeric` is not specified, the data is represented as JSON string. During parsing, the connector does not perform any editing-related validation.

Numeric Items with Scaling Position P

Numeric items can contain the character P in a `PICTURE` clause. These characters can be present as a continuous sequence at the beginning or end of `PICTURE`.

When located at the beginning of the clause, the P character indicates a decimal point followed by zeroes. For example, `PIC PPP9` describes values 0.0001, 0.0002, and so on. When at the end of the clause, the P characters indicate the number of zeros at the end of a value. A decimal point is presumed at the end of a value. For example, `PIC 9PPP` describes values 1000 , 2000, and so on.

The storage size is unaffected by P, and the zeroes are not stored but rather are implicitly added to the actual algebraic value during mathematical operations, `MOVE` operations, and so on.

The Connector adds the zeroes as well, so that the flow sees the implied algebraic value.

- Parse activity adds zeros to the output values and returns an algebraic value.
- Render activity accepts an algebraic value and removes zeroes at P positions. Input values are validated to comply with use P in the `PICTURE` clause.

REDEFINES Clause

An item with REDEFINES clause uses the same storage as the item it redefines. For example:

```
01 ROOT.
     02 A PIC X.
     02 B PIC S9 REDEFINES A.
```

Item A and B use the same storage area. Depending on the application logic, either text item A or zoned item B may be present.

The Connector supports the `REDEFINES` clause. In the JSON schema, both A and B are present, but only one of the properties can be present at a time.

The behavior of Parse is affected by the `@controlField`, `@controlValues`, and `@defaultRedefine` annotations.

The behavior of Render is driven by the input mapping and `@defaultRedefine` when no mappings are established. Only one of the redefined items can be mapped at a time. In the given example either A or B can be mapped. If both are mapped, Render raise an error.

If A mapped, Render generate a PIC X value and if B is mapped, Render generates a zoned PIC S9 value.

@controlField:{item name} and @controlValues

This annotation is used by Parse to generate one of the redefined items based on a value of another item called control field. You can specify a set of values that causes a particular redefined item to be read and returned. For example:

```
01 ROOT.
    02 C PIC X.
*    @controlField: C
*    @controlValues: "A"; "1"
    02 A PIC X.
*    @controlValues: "B"; "2"
    02 B PIC S9 REDEFINES A.
```

When C contains values **A** or **1** Parse reads a `PIC X` item A and return property **A** in the output.

When C contains values **B** or **2** Parse reads a `PIC S9` item B and return property **B** in the output.

A control field can be specified as a relative name, for example **C**. Or it can be specified as a qualified name that includes parent items, such as `ROOT.C`. In case of a multilayer hierarchy, some group items may be removed if the specification is not ambiguous. For example:

```
01 ROOT.
    02 ROOT2.
        03 C PIC X.
* @controlField: ROOT.ROOT2.C
    02 A PIC X.
    02 B PIC S9 REDEFINES A.
```

Here the control field is fully qualified.

```
01 ROOT.
    02 ROOT2.
        03 C PIC X.
* @controlField: ROOT.C
    02 A PIC X.
    02 B PIC S9 REDEFINES A.
```

In the above, **ROOT2** is omitted but this is still a valid specification because there is only one **C** in the hierarchy of **ROOT**.

```
01 ROOT.
    02 C PIC X.
    02 ROOT2.
        03 C PIC X.
* @controlField: ROOT.C
    02 A PIC X.
    02 B PIC S9 REDEFINES A.
```

ROOT.C is ambiguous because there are two items named **C** in the ROOT hierarchy. Connector ignores `@controlField` in this case.

Control values can be specified in several formats. For more information, see Control Values.

@defaultRedefine

This annotation designates an item as a default. It affects the following:

- If no control field is specified or a value of a control field is not on the values specified in `@controlValues`, Parse reads and returns this item. Only one of the redefined items can have a `@defaultRedefine` annotation. For example:

  ```
  01 ROOT.
      02 C PIC X.
  *    @controlField:C
  *    @controlValues: "A"; "1"
  *   @defaultRedefine
      02 A PIC X.
  *    @controlValues: "B"; "2"
  *   @defaultRedefine
      02 B PIC S9 REDEFINES A.
  ```

  `@defaultRedefine` before the item **B** is ignored because **A** was already designated as default.

- If neither **A** or **B** are mapped to the input of Render, it initializes the area according to the rules of the format of the default item.

- If `@defaultRedefine` is not specified on any of the items, the first item in the REDEFINE group is considered to be a default.

Control Values

You can specify the control values in a list and separate them with semicolons.

The Data Conversion Connector supports the following types of control values:

- **Text values:** ABC, "ABC"
- **Bitmask values:** "1001" &

- **Hexadecimal values:** "1FB4"X

All these types can be specified in the same list:

ABC; "1001"&; "1FB4"X

Text Values

You can use a simple text expression or enclose it in single or double quotes. When a value is enclosed in quotes, the quotes are not considered a part of the value. To embed a quote into a quoted value, you must repeat the quote twice unless the quote differs from those at the beginning and at the end of the value.

You can use unquoted values when:

- A value does not include a semicolon
- Does not start or end with a space; leading and trailing spaces are not considered a part of an unquoted value

Several examples are shown here:

Value List	Values
"abc"; 'a'	abc a
"abcd'abcd'xyz"	abcd'abcd'xyz
'abcd''abcd''xyz'	abcd'abcd'xyz
a;b;c	a b c
'abc	Invalid value because of no closing single quotes
'abc"	Invalid value because of no closing single quotes

Numeric Values

Text control values for numeric items are required to be valid numbers. Also special floating point values can be specified:

List	Description
Inf	Positive Infinity
-Inf	Negative Infinity
NaN	Not a Number

Unsupported Clauses, Features, Levels, and Phrases

COBOL clauses, levels, and phrases that are not supported by the Data Conversion Connector might be ignored or result in an unsupported copybook. The following table gives the details about such unsupported features:

Features	Description
Level 66	The connector ignores the data elements defined with level 66 (`RENAMES` clause).
Level 88	The connector ignores the data elements defined with level 88 (`VALUE` or `VALUES` clause).
BLANK WHEN ZERO	The connector ignores this clause.
EXTERNAL	
GLOBAL	
JUSTIFIED	
Nested copybooks	The connector does not support copybooks that contain the `COPY` directive for other copybooks.
`OCCURS` clause • ASCENDING • DESCENDING • INDEXED • KEY	The connector ignores these phrases for the `OCCURS` clause.
`SYNCHRONIZED` clause • LEFT • RIGHT	The connector ignores these phrases for the `SYNCHRONIZED` clause and presumes `LEFT` clause.
`USAGE` clause • NATIVE • OBJECT REFERENCE	The connector ignores these phrases for the `USAGE` clause.

Handling of Zoned Decimal Items

The Connector presents certain COBOL items as zoned decimal numbers in the binary data. Such items are of `USAGE DISPLAY`, a numeric picture clause, and no `SIGN SEPARATE` clause.

The zoned decimal format represents all decimal digits in text — except for the last byte or possibly the first byte, depending on whether `LEADING` or `TRAILING` is in the `SIGN` clause. That byte contains both the decimal digit and the sign for the whole value. The sign is specified in the first nibble (half byte) and the decimal digit in the second nibble.

The zoned decimal format is one of three variations: EBCDIC, Strict ASCII, and Modified ASCII. The Connector expects a particular format, which depends on the **Character Set** setting or **forceCharacterSet** input field for the Render Copybook Data or Parse Copybook Data activity. For details on the related configurations, see the following sections.

EBCDIC Character Sets (Such As IBM037)

The Parse activity expects EBCDIC-zoned decimal format and fails if it encounters any of the ASCII format data. The EBCDIC-zoned decimal format uses EBCDIC codepoints for digits and defines the following sign nibbles for the sign byte:

- xC and xA: Sign positive
- xB and xD: Sign negative

The Parse accepts positive and negative nibbles for signed items (with S in the PICTURE clause for COBOL) and xF for both signed and unsigned items. Note that signed nibbles are not valid in an unsigned item.

The Render activity generates the signed nibbles : xC for positive and xD for negative signed data. For unsigned data, the activity generates xF.

The **ASCII Zoned Format** drop-down list and the input field asciiZonedFormat settings are disregarded.

ASCII Character Sets

The Parse activity accepts both Strict ASCII and Modified ASCII. Both formats use ASCII codepoints for decimal digits but represent the sign byte differently. In Strict ASCII, a nibble value of x3 denotes a positive value; x7, a negative one.

Strict ASCII and Modified ASCII represent positive signed and unsigned values in the same way.

Modified ASCII does not have predefined nibbles. See the table on Modified ASCII Sign Bytes.

Render activity can generate both strict and modified ASCII formats. You can specify the desired format in two ways:

- Use **ASCII Zoned Format** dropdown list in Settings for Render activity to specify a format.
- Pass **strict** or **modified** to the input field **asciiZonedFormat**. This field takes precedence over the **ASCII Zoned Format** dropdown list.

Sign	Sign of the Entire Value	Decimal Digit in Sign Byte
0x7B	Positive	0
0x41	Positive	1
0x42	Positive	2
0x43	Positive	3
0x44	Positive	4
0x45	Positive	5
0x46	Positive	6

Sign	Sign of the Entire Value	Decimal Digit in Sign Byte
0x47	Positive	7
0x48	Positive	8
0x49	Positive	9
0x7D	Negative	0
0x4A	Negative	1
0x4B	Negative	2
0x4C	Negative	3
0x4D	Negative	4
0x4E	Negative	5
0x4F	Negative	6
0x50	Negative	7
0x51	Negative	8
0x52	Negative	9

Sign Nibbles in Zoned and Packed Decimal Items

Items that are zoned, such as PIC S999 or packed decimal, such as USAGE PACKED-DECIMAL or COMP-3, contain a sign nibble which is half a byte.

There is a small difference in processing these items in zOS systems and IBM i systems. On a zOS system, positive values in signed items such as PIC S999 contain nibble xC, while on IBM i system, such items contain nibble xF.

During parsing, the Data Conversion Connector recognizes both variations. However, during rendering, you have to specify what flavor of behavior is needed. You can use the **IBM i Sign Nibbles** setting in the Render activity. Select it to generate the IBM i flavor. Unselect to generate the zOS flavor.

Handling of Floating Point Items

You can configure floating point items to use either the IBM Hexadecimal format or the IEEE 754 format. The format is selected in the Copybook connection.

IEEE-754 and IBM Hexadecimal are two different floating point formats. IBM hexadecimal is the format most commonly used on IBM mainframe computers. IEEE-754 is used by many other hardware platforms and software systems.

There are inherent differences in the degree of precision and the magnitude of the exponent offered by these two formats. The differences are as follows:

- IBM float and double formats use a 7 bit exponent value that represents an exponent of base 16.
- IEEE formats use 8 bits for float and 11 bits for double that represents an exponent of base 2.

- The remaining bits in both formats are used for precision so there are commensurate differences in the degree of precision available.

The Connector is capable of processing IBM hexadecimal format. This functionality is enabled using the **Floating Point** field in the Copybook resource.

Parse Copybook Data Activity

The Parse Copybook Data activity parses binary COBOL data and returns JSON data. This activity parses incoming data according to the selected Copybook connection. After parsing, the output data conforms to the schema generated by the Copybook connection. Also, other activities can access the data.

Configuration

On the **Settings** tab, you can select the Copybook connection and configure the parsing process.

Field	Description
Copybook	The Copybook connection that contains a copybook.
Big Endian	The byte order of the incoming data. • Select true to specify the byte order as Big Endian, for example, for IBM mainframes and certain UNIX platforms. • Select false if the byte order is Little Endian. Examples: For Microsoft Windows and certain UNIX platforms
Character Set	The character-set encoding for the string values in the incoming data except for items with USAGE DISPLAY-1. This also affects the parsing of zoned decimal values. For more information, see Handling of Zoned Decimal Items. Some of the available character sets are multi-byte, for example UTF-8. If you select a multi-byte character set, it is used to convert binary content to strings. However the binary length of item is based on its PICTURE assuming one byte per character. **Character Set** selection does not alter the binary offsets and lengths of items in the copybook.
Character Set for DISPLAY-1 Items	The character-set encoding for items with USAGE DISPLAY-1. Binary length of such items is based on their PICTURE clause (assuming two byte per character). However, you can choose a single-byte or a variable character set, which is then used to convert text. **Character Set** selection does not alter the binary offsets and lengths of items in the copybook.
Multiple Records	This option determines whether to repeat the copybook parsing until the end of the input data is reached and whether to generate multiple output elements. If **Multiple Records** is selected, the schema of the Output property data becomes an array. Otherwise the schema is an object that contains a single instance of the copybook data.
Record Delimiter Length	The field that specifies the length of a delimiter between records in the input data in case of a true selected in **Multiple Records** field.

Field	Description
Trim Whitespaces from text items	Removes leading and trailing whitespace characters as well as additional characters from string values. The additional characters are as follows: U+0000 U+FEFF U+FFFD
Return Empty Nodes	Determines whether to exclude empty strings, objects or arrays from the output.
Return Remaining Bytes	Enable returning of the remainder of the input binary data. If selected, the output of the activity displays the **remainingBytes** field. This field contains the tailing portion of the input binary that has not been processed by this activity. For more information, see Connecting Multiple Parse Activities.
Allow Incomplete Records	This field determines whether the input must contain all the input data for the entire copybook layout (record) or whether incomplete layouts are tolerated. For more information, see Parsing Incomplete Records.
Allow Incomplete Text Items	If enabled, the last text item is allowed to have less bytes that the copybook implies. For more information, see Parsing Incomplete Records.
Ignore Invalid Items	This field turns off errors resulting from invalid content in the binary data in an elementary item. For more information, see Validation of Elementary Items.

Input

The **Input** tab has the following fields:

Field	Description
bytes	A byte sequence that contains COBOL data. It is encoded as Base64.
forceCharacterSet	A runtime override for the **Character Set** setting. For details about **Character Set**, see the description of the field **Character Set** in the section Parse Copybook Data-Configuration.
forceCharacterSetDisp1	A runtime override for the **Character Set setting for DISPLAY-1 Items** setting. For details about **Character Set setting for DISPLAY-1 Items**, see the description of the field **Character Set for DISPLAY-1 Items** setting in the section Parse Copybook Data - Configuration.
recordDelimiterLen	Runtime override for **Record Delimiter Length**. For details about **Record Delimiter Length**, see the description of the field **Record Delimiter Length** in the section Parse Copybook Data - Configuration.

Output Settings

The **Output Settings** tab displays the output schema of the activity as a tree structure. The output is read-only. The information in the schema varies based on the fields selected on the **Configuration** tab.

The properties that are displayed in the schema correspond to the output of this activity and can be used as input by subsequent activities in the flow.

Output

The **Output** tab has the following fields:

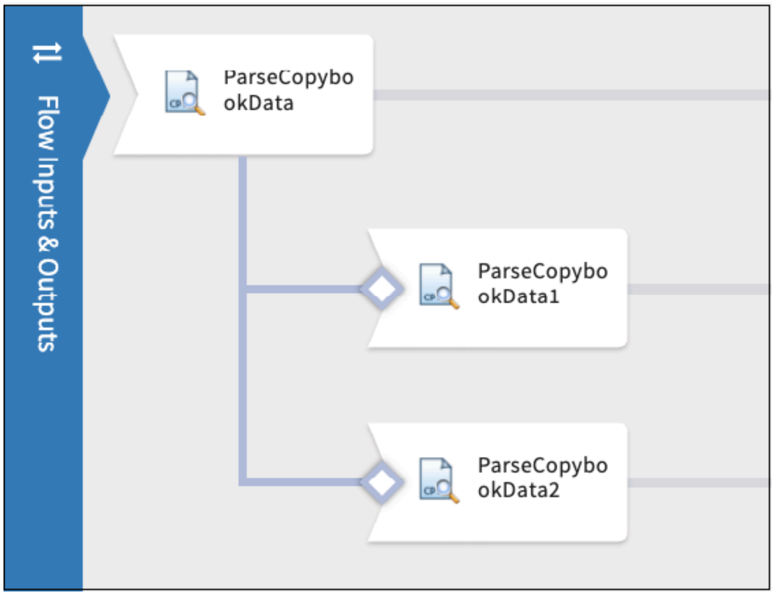
Field	Description
noOfRecords	This is the number of records the activity has read.
remainingBytes	If you select the **Return Remaining Bytes** in Settings, the output of the activity contains the **remainingBytes** field. This field contains the tailing portion of the input base64Binary that has not been processed by this activity. For more information, see Connecting Multiple Parse Activities.
data	JSON data read from the input byte sequence. If **Multiple Records** is selected, it contains an array of records or a single record. The schema for a record is generated by the selected Copybook connection to reflect the copybook.

Connecting Multiple Parse Activities

A binary buffer often consists of data whose layouts could be described in multiple copybooks. Sometimes the decision of which copybook to use next is made based on the data present. Control fields in `REDEFINE` groups offer a form of control but do not cover all patterns. The most typical pattern is as follows:

- Two or more copybooks of different byte size are packed into an envelope message.
- Each copybook is preceded with an indicator field that determines which copybook layout follows.
- Layouts are different sizes, which makes it impossible to pack copybooks into a super-copybook with a `REFEDINES` clause.

To address this scenario, the Parse Copybook Data activity offers an option to return the remainder of the byte data, making it available to the next Parse activity. For example:



In this example, ParseCopybookData parses a small copybook

```
01 INDICATOR PIC X
```

The transitions are conditional on the data returned from the activity. The conditions are as follows:

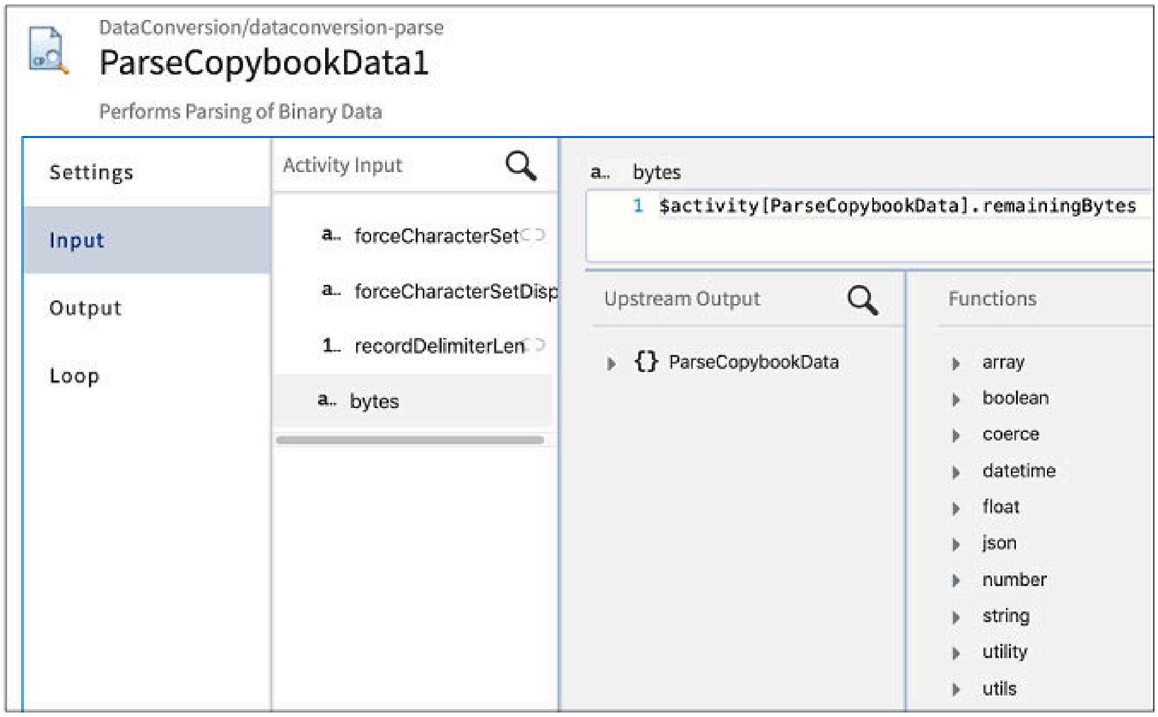
`$activity[ParseCopybookData].data.INDICATOR=='C'`

and

`$activity[ParseCopybookData].data.INDICATOR=='O'`

ParseCopybookData1 is configured to use the copybook connection CUSTOMER and ParseCopybookData2 uses the copybook connection ORDER.

ParseCopybookData has the **Return Remaining Bytes** option enabled, the data is forwarded to the ParseCopybookData1 or ParseCopybookData2 activity.


DataConversion/dataconversion-parse

ParseCopybookData1

Performs Parsing of Binary Data

bytes

```
$activity[ParseCopybookData].remainingBytes
```

The **Return Remaining Bytes** option is not selected by default. It must be enabled using the check box on the **General** tab. Selecting the **Return Remaining Bytes** check box results in the addition of the remainingBytes property to the parse activity's output.

Return Remaining Bytes is not applicable to activities with **Multiple Records** selected. If this option is selected, then Parse reads the copybook multiple times till the end of data is reached.

When the end of data is reached, the returned **remainingBytes** is empty.

Parsing Incomplete Records

A copybook describes a layout where a certain number of bytes is expected for each item. If Parse encounters the end of input data before all items have been fully read, it throws an exception. However, Parse offers flexibility to read incomplete data layouts.

Allow Incomplete Records

When the Allow Incomplete Records field is selected, the following conditions apply:

- If, while reading an item, Parse encounters a premature end of data, then the activity stops and returns the JSON containing all items that were read fully. This is considered a normal end of the processing (as opposed to an error condition when **Allow Incomplete Records** is not selected).

- Similarly, when Parse is reading an item with `OCCURS` clause, the premature end of data is considered normal even though a required number of occurrences was not read. When **Allow Incomplete Records** is not selected, this results in an error.

When an incomplete item is discarded, the Data Conversion Connector is unable to perform any data validity checks, which means that a data corruption is less likely to be detected.

Allow Incomplete Text Items

The **Allow Incomplete Text Items** setting is designed for copybooks that end with a long PIC X(...) or PIC G(...) item that is not fully populated. To preserve resources, your program may choose to only deliver the beginning portion of this field that contains non-blank characters.

If **Allow Incomplete Text Items** is selected, Parse returns the string based on the available data even though it might be shorter than the declared item length. When **Allow Incomplete Text Items** is not selected, this field is discarded or the activity fails if **Allow Incomplete Records** is not selected.

Validation of Elementary Items

The Parse Copybook Data activity requires the binary data of an elementary item to be in full compliance with its definition in a copybook by default. For example, packed decimal data can only have certain values for a sign nibble. Normally, Parse throws an error in case of a violation.

However, this behavior is customizable using the **Ignore Invalid Items** field on a Parse activity. This setting turns off errors resulting from invalid content in the binary data in an elementary item.

When **Ignore Invalid Items** is selected:

- Any malformed data is ignored and no error is thrown.
- The item is not included in the resulting JSON.

For example, this copybook defines an unsigned item:

```
01 ZONED-UNSIGNED PIC 99.
```

F1E2 (in hexadecimal form) is not valid for this copybook because E is not a valid sign nibble. If **Ignore Invalid Items** is not selected, this results in a failure. However, if you select **Ignore Invalid Items**, then the item is quietly ignored and not included in the resulting JSON. However, that parsing does not stop, and any valid items found after an erroneous one are processed and returned.

This setting applies to a wide variety of items, including:

- zoned and packed-decimal numbers
- edited numerics
- items with DATE attribute
- external floating point items

Ignore Invalid Items is not a replacement for **Allow Incomplete Records**. It is applied to items that have enough bytes in the data. To allow premature end of data than select **Allow Incomplete Records**.

Special Cases

There are several circumstances under which Ignore Invalid Items is not in effect:

- When an invalid item is used as an object of `OCCURS DEPENDING ON` clause. For example:

  ```
  01 COUNT PIC 9.
               01 REP PIC X OCCURS 1 TO 10 TIMES DEPENDING ON COUNT
  ```

  If COUNT is invalid, Parse cannot determine the number of occurrences of REP to read. In this case, parsing fails.

Exception: If you selected **Allow Incomplete Records** and there are no bytes following COUNT, then Parse completes normally.

- When an invalid item is used as a control field for a `REDEFINE` group. The connector requires a valid control field value to determine how to proceed.

There are two cases when the failure does not occur. When **Allow Incomplete Records** is selected and the data ends prior to the `REDEFINE` group. In this case Data Conversion Connector does not need the control value to be valid. If during the control field matching (see Control values) Parse encounters only' ...'X and' ...'& control field values. These values operate directly on the binary content and do not require validity of a control field. In this case the control field is not included in resulting JSON but a member of a `REDEFINE` group is.

Render Copybook Data Activity

This activity accepts data from other activities and produces data according to the COBOL copybook held in a selected Copybook connection. The COBOL data can then be passed to an application. Flogo flows can use other activities to carry out the data delivery.

Configuration

The fields on the **Settings** tab of this activity are described as follows:

Field	Description
Copybook	The Copybook connection that contains a copybook.
Big Endian	The byte order of the incoming data. Select true to specify the byte order as Big Endian, for example, for IBM mainframes and certain UNIX platforms. Select false, if the byte order is Little Endian, for example, for Microsoft Windows and certain UNIX platforms.
Character Set	The character-set encoding for the string values in the rendered data except for items with USAGE DISPLAY-1, also affects the rendering of zoned decimal values. For more information, see Handling of Zoned Decimal Items. Some of the available character sets are multi-byte, for example UTF-8. If you select a multi-byte character set, it will be used to convert strings to binary content. However the item's binary length is based on its PICTURE assuming one byte per character. **Character Set** selection does not alter the binary offsets and lengths of items in the copybook.
Character Set for DISPLAY-1 Items	The character-set encoding for items with USAGE DISPLAY-1. Binary length of such item's is based on their PICTURE clause assuming two byte per character. However, you can choose a single-byte or a variable character set and it is used to render text. **Character Set for DISPLAY-1 Items** selection does not alter the binary offsets and lengths of items in the copybook.
Multiple Records	The field that instructs the activity to render multiple copybook records. If **Multiple Records** is selected, the schema of the Input property data becomes an array. Otherwise the schema is an object that contains a single instance of the copybook data.

Field	Description
Record Delimiter	The field that specifies the delimiter between records in the output data in case of a selected **Multiple Records** field. You can specify the following valid delimiter types: • **None:** Specifies that a new record begins directly after the previous record ends. No bytes separate the records. • **New Line:** Specifies that a new-line character (hex value 0A) separates the records in the output. • **Carriage Return:** Specifies that a Carriage Return character separates (hex value 0D) the records in the output. • **Carriage Return/Line Feed:** Specifies that a carriage-return character followed by a New Line character separates the records in the output (byte sequence 0D 0A). You can specify your own delimiter sequence using **recordDelimiter** on the **Input** tab of the activity.
Truncate Multibyte Strings	The field that specifies whether to truncate long strings and is applicable only if you have selected a multibyte character set. For DISPLAY items, the binary length is equal to the number of characters. If you configure the activity to use a multibyte character set, the binary representation of a string might become longer than the allocated length. In that case • If you select this field, the activity truncates the binary representation at the nearest character boundary to ensure that the activity does not throw an error and that the allocated length is never exceeded. Exercise caution when selecting this field. Even though data truncation occurs at a character boundary, the truncation may cause an undesired content change for certain multibyte character sets. • If you do not select this field, the activity throws an error.
Filler Character	The field that specifies the filler character used for text padding or layout gaps. You can choose either NULL (a byte with value zero) or Space as the fill character; NULL is the default. When you select Space, the exact byte value depends on the **Character Set** setting of your activity. For more information, see Item Initialization During Rendering.

Field	Description
Default Numeric and DATE Items to	This field specifies how the binary data is initialized for numeric or `DATE FORMAT` items that are not specified in the input mapping. The options are as follows: • **Type Defaults:** Numeric and Date items are initialized differently. – Numeric items are initialized with a numeric value 0 and are rendered according to the `PICTURE` and `USAGE` clauses, as well as settings of the Copybook connection and Render activity. – DATE items are initialized with a default date 0001-01-01, which is rendered according to the `PICTURE`, `USAGE` clauses, as well as settings of the Copybook connection and Render activity. • **LOW-VALUES:** All numeric and DATE items are initialized with zero bytes, regardless of the copybook or any other settings. For more information, see Item Initialization During Rendering.
ASCII Zoned Format	You can use this field to select a specific ASCII Zoned format to be generated. Use this field when the **Character Set** field or **forceCharacterSet** field is set to a character set that belongs to the ASCII family. • **Strict:** Generates Strict ASCII zoned decimal format. • **Modified:** Generates Modified ASCII zoned decimal format. The input field **asciiZonedFormat** acts as a runtime override of this field. For more information, see Handling of Zoned Decimal Items.
IBM i Sign Nibbles	If it is selected, this field instructs the activity to render the IBM i flavor of zoned and packed decimal items. If it is not selected, zOS flavor is generated. For more information, see Sign Nibbles in Zoned and Packed Decimal Items.

Input

The **Input** tab has the following fields:

Field	Description
forceCharacterSet	A runtime override for the **Character Set** setting. For details, see the description of the field **Character Set** in the section Render Copybook Data - Configuration.
forceCharacterSetDisp1	A runtime override for the **Character Set setting for DISPLAY-1 Items**. For details, see the description of the field **Character Set for DISPLAY-1 Items** in the section Render Copybook Data - Configuration.
recordDelimiter	A delimiter to be placed between copybook records when **Multiple Records** is selected. If specified, it overrides **Record Delimiter** specified on **Settings** tab. This is a Base64 encoded binary value.

Field	Description
asciiZonedFormat	This is a runtime override for the **ASCII Zoned Format** field of the Render activity. It is used to select a specific flavor of ASCII Zoned Decimal format. Use this field when the **Character Set** field or **forceCharacterSet** field is set to a character set that belongs to the ASCII family. You can use these values: • **strict:** Generates Strict ASCII zoned decimal format. • **modified:** Generates Modified ASCII zoned decimal format.
data	The json data to be converted to a COBOL binary sequence. If **Multiple Records** is selected it contains an array of records or a single record otherwise. The schema for a record is generated by the selected Copybook connection to reflect the copybook contents.

Output

The **Output** tab has the following fields:

Field	Description
bytes	A base64 encoded byte sequence generated from JSON input data. This sequence can be consumed by a COBOL program.
noOfRecords	Number of records rendered.

Item Initialization During Rendering

In most cases, you supply values to a Render activity through mapping. You are not required to always supply a value using mapping.

When you skip a value, the following may happen:

- Parse checks for any initial value supplied in a VALUE clause in the copybook and renders it if it is present.
- The binary data is formed according to the policies described in the following table. This table also describes any padding that results in case of insufficient data length or layout gaps.

Item Type	Action
COBOL text items (PIC X or G)	The **Filler** setting of the activity is used to initialize missing items or pad shorter strings.
Padding within a REDEFINE group	If a REDEFINE group consists of items on unequal byte length, the **Filler** settings is used to pad the shorter items.

Item Type	Action
Gaps due to SYNC clause and top-level item placement on the double-word boundary.	**Filler** setting used to fill the gaps.
@binary items	The **Filler** setting is used to initialize missing items or pad shorter values.
Items with DATE FORMAT	The setting **Default Numeric and DATE Items** to, is in effect. When **Type Defaults** is selected, the default date 0001-01-01 is rendered. Otherwise a zero byte is used as a filler.
External floating point items	The setting **Default Numeric and DATE Items** to, is in effect. When **Type Defaults** is selected, the algebraic value 0 is rendered according to the copybook specifics and activity settings. Otherwise a zero byte is used as a filler.
Numeric edited items	The setting **Default Numeric and DATE Items** to, is in effect. When **Type Defaults** is selected, the algebraic value 0 is rendered according to the copybook specifics and activity settings. Otherwise a zero byte is used as a filler.
Zoned and packed decimal items	A zero byte is used as filler.
COMP-1 and COMP-2 items	A zero byte is used as filler.