


Slide 1

> WebSphere Education

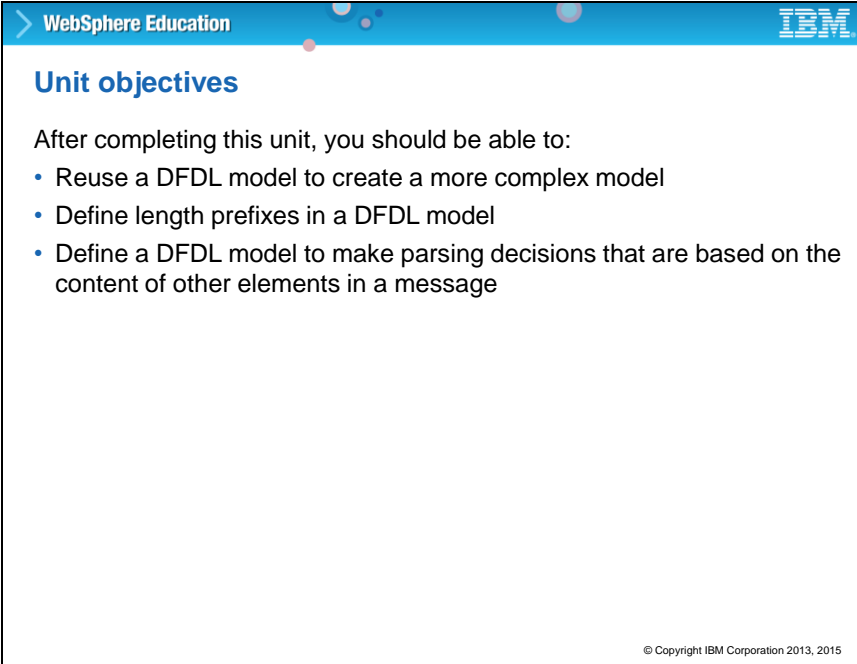
IBM

## Modeling complex data with DFDL



© Copyright IBM Corporation 2013, 2015  
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

IBM



The slide features a blue header bar with the text 'WebSphere Education' on the left and the IBM logo on the right. Below the header, the title 'Unit objectives' is displayed in blue. The main content area contains a list of objectives. At the bottom right, there is a small copyright notice.

> WebSphere Education IBM

### Unit objectives

After completing this unit, you should be able to:

- Reuse a DFDL model to create a more complex model
- Define length prefixes in a DFDL model
- Define a DFDL model to make parsing decisions that are based on the content of other elements in a message

© Copyright IBM Corporation 2013, 2015

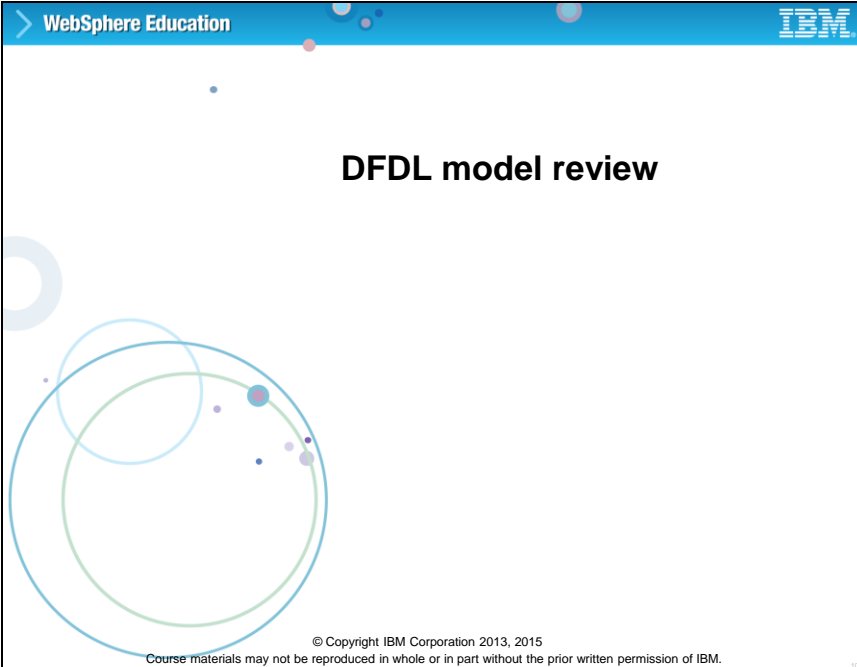
## Unit objectives

You can model a wide variety of message formats by using DFDL schema files. In this unit, you learn how to use DFDL to model complex data that includes multiple records types, length prefixes, choice groups, optional elements, and variable array elements.

After completing this unit, you should be able to:

- Reuse a DFDL model to create a more complex model.
- Define length prefixes in a DFDL model.
- Define a DFDL model to make parsing decisions that are based on the content of other elements in a message.


## Slide 3




The slide features a blue header bar with the text "WebSphere Education" on the left and the "IBM" logo on the right. The main title "DFDL model review" is centered in a large, bold, black font. Below the title, there is a decorative graphic consisting of several overlapping circles in light blue and green, with small colored dots scattered around them. At the bottom of the slide, there is a small copyright notice: "© Copyright IBM Corporation 2013, 2015. Course materials may not be reproduced in whole or in part without the prior written permission of IBM."

### Topic 1: DFDL model review

Data Format Description Language (DFDL) is an XML-based language that is used to define the structure of formatted data in a way that is independent from the data format itself. This topic is a brief review of the DFDL concepts that are described in detail in the prerequisite course, *ZM666 IBM Integration Bus V10 Application Development I*.



WebSphere Education 

### DFDL data support (1 of 2)

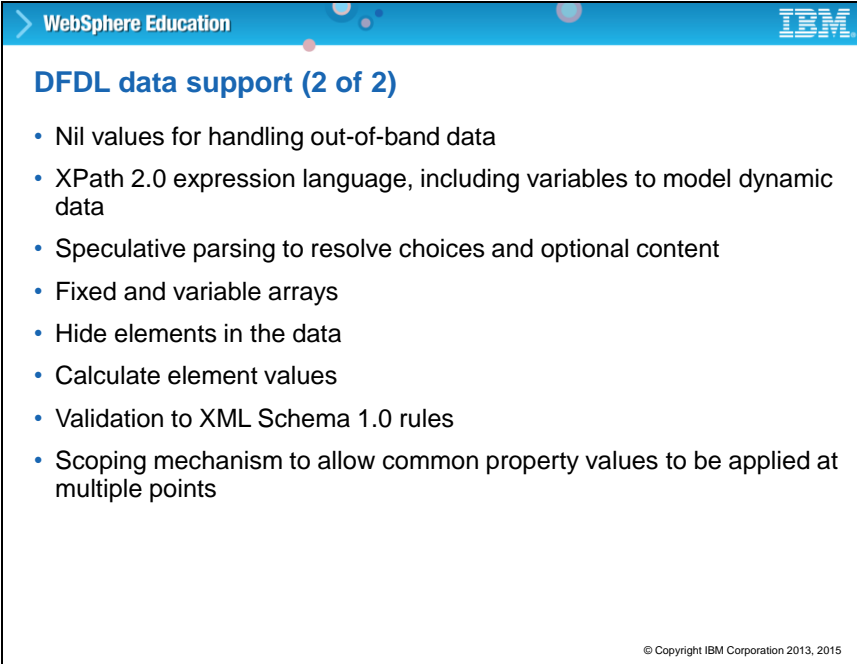
- Language structures such as COBOL, C, and PL/I
- Industry standards such as SWIFT, HL7, FIX, HIPAA, X12, EDIFACT, ISO8583
- Fixed data and text of binary markup delimited data
- Text data types such as strings, numbers, zoned decimals, calendars, Booleans
- Binary data types such as integers, floats, BCD, packed decimal, calendars, Booleans
- Bidirectional text
- Bit data of arbitrary length
- Pattern languages for text numbers and calendars
- Ordered, unordered, and floating content
- Default values on parsing and serializing

© Copyright IBM Corporation 2013, 2015

### DFDL data support (1 of 2)

By using DFDL, you can model all the types of data that Integration Bus traditionally supports, plus a number of newer varieties of data elements.

The complete list of data types that the IBM implementation of DFDL supports is shown on this slide and the next.



The slide is titled "DFDL data support (2 of 2)" and is part of a WebSphere Education presentation. It features a blue header with the "WebSphere Education" text and the IBM logo. The main content is a bulleted list of DFDL data support features. The footer contains a copyright notice: "© Copyright IBM Corporation 2013, 2015".

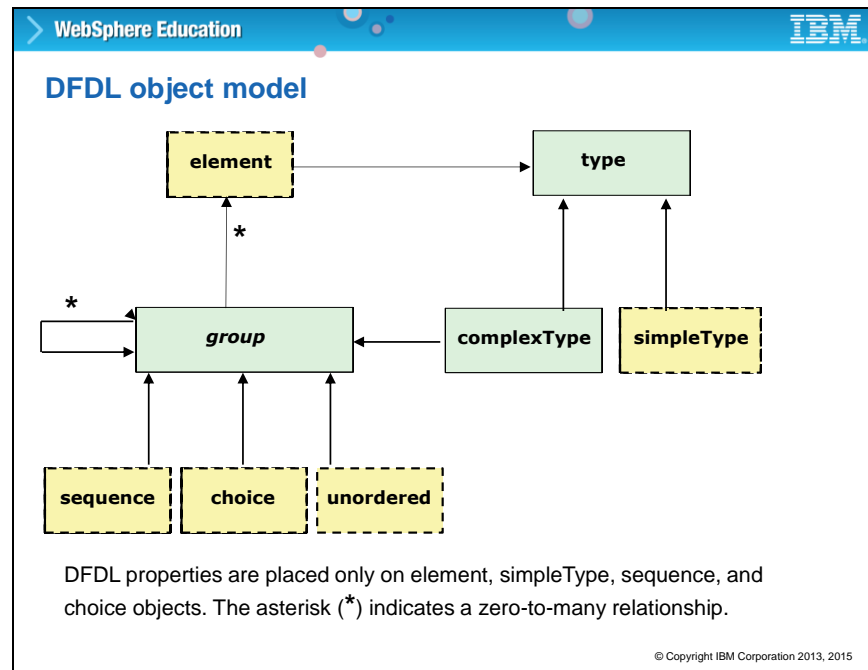
- Nil values for handling out-of-band data
- XPath 2.0 expression language, including variables to model dynamic data
- Speculative parsing to resolve choices and optional content
- Fixed and variable arrays
- Hide elements in the data
- Calculate element values
- Validation to XML Schema 1.0 rules
- Scoping mechanism to allow common property values to be applied at multiple points

© Copyright IBM Corporation 2013, 2015

### **DFDL data support (2 of 2)**

The DFDL standard provides an extensive list of supported data types. Not all of them are reflected in this slide and the previous figure.

For more information about the DFDL data types that are not supported and implementation restrictions, see the IBM Knowledge Center for IBM Integration Bus.




## DFDL object model

When you model with DFDL, objects define the logical format of the data. The diagram displays the objects, and the relationships between them. The asterisk (\*) represents a zero-to-many relationship.

An example of a group object is header record or trailer record. They are made up of simpleType elements such as fields but might also contain other complexType objects.

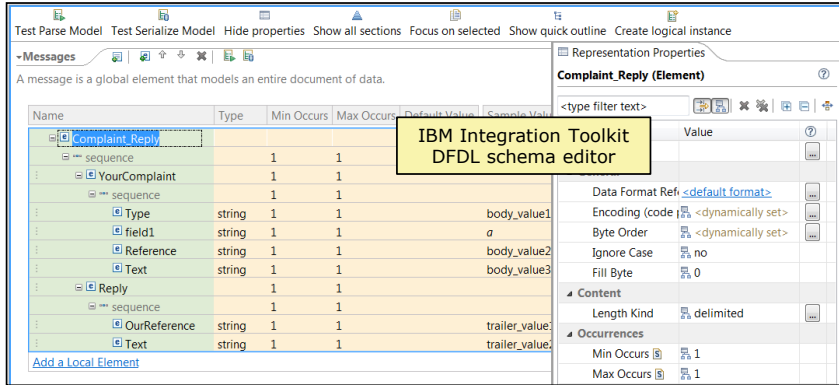
A group can be a sequence group where the elements are always in a specific order. A group can also be a choice group. Mixed data that contains purchase order records and acknowledgment records would be an example of a choice group.

DFDL properties can be placed only on sequence, choice, element, and simpleType.

WebSphere Education 

## DFDL properties

- Describe the physical representation of the objects in a DFDL schema
- Do not have built-in defaults



The screenshot shows the IBM Integration Toolkit DFDL schema editor. The main window displays a tree view of the 'Complaint\_Reply' element, which is a sequence of several sub-elements. The 'Representation Properties' panel on the right shows the properties for the selected 'Complaint\_Reply' element, including Data Format Ref, Encoding, Byte Order, Ignore Case, Fill Byte, Content, Length Kind, and Occurrences.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Complaint_Reply	sequence	1	1		
YourComplaint	sequence	1	1		
Type	string	1	1	body_value1	
field1	string	1	1	a	
Reference	string	1	1	body_value2	
Text	string	1	1	body_value3	
Reply	sequence	1	1		
OurReference	string	1	1	trailer_value1	
Text	string	1	1	trailer_value2	

IBM Integration Toolkit  
DFDL schema editor

© Copyright IBM Corporation 2013, 2015

## DFDL properties

DFDL properties describe the physical representation of the data objects in a DFDL schema. The object determines the available properties.

- For elements and simple types, you can define the data representation and how the length of the element is determined.
- For a sequence group, you can define the sequence type, such as ordered or unordered. You can also define the separator character between the elements of the group.
- For all elements, you can define an initiator character or characters, a terminator character, or characters, encoding, and whether the data is aligned to the left or to the right.

It is important to remember that DFDL properties do not have built-in defaults. You should specify default values when you create the model in the DFDL Schema editor.

WebSphere Education

## DFDL expressions

- Can be used to set some properties dynamically at processing-time
  - When a property value must be set dynamically from the contents of the data
  - In an assert or discriminator annotation
  - When setting the value or default value of a variable
- Expression language is a subset of XPath 2.0, including variables, and with some extra DFDL-specific functions

Integration Toolkit DFDL schema editor contains an XPath Expression Builder to help you build the expression

© Copyright IBM Corporation 2013, 2015

## DFDL expressions

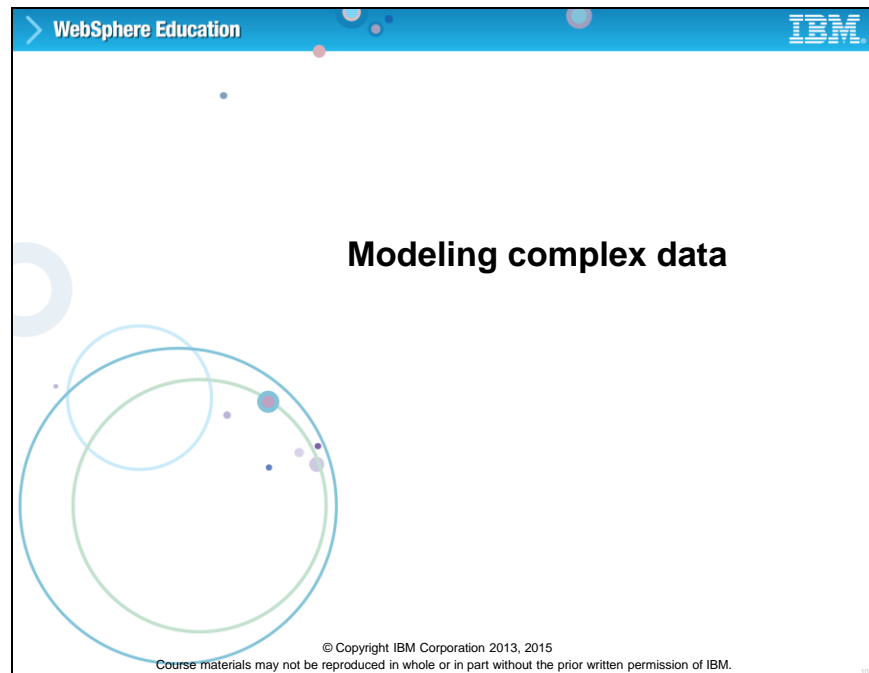
If you are creating or modifying a DFDL schema manually, you can use DFDL expressions to set properties dynamically at processing-time.

The example on this figure shows the use of an expression to provide the count of the number of occurrences for an unbounded array element that is called value. The example shows the use of a relative path to refer to a **count** element earlier in the data.

The DFDL expression language is a subset of XPath 2.0. Expressions follow XPath 2.0 syntax rules, but are always enclosed by using the brace characters.


A discriminator and an assert are examples of DFDL expressions. Discriminators and asserts are described in detail later in this unit. You also define discriminators by using the IBM Integration Toolkit DFDL Schema editor in the exercise that follows this unit.





## Topic 2: Modeling complex data

In this topic, you learn how to reuse a DFDL model to create a more complex model, define length prefixes in a DFDL model, and define a DFDL model to make parsing decisions that are based on the content of other elements in a message.

WebSphere Education 

## Understanding the logical structure of the data

1. Identify complex structures
  - Complex types
  - Complex elements
2. Identify simple items
  - Simple types
  - Simple elements
3. Identify structure order
  - Sequence groups
  - Choice groups
  - Unordered groups
4. Identify cardinality with **Min Occurs** and **Max Occurs**
5. Identify elements that can be nil (null) and default values

```
{ N:Joe Bloggs, A:50, D:19620503, P:Y, S:40000 }
{ N:Fred Smith, A:30, D:19930225, P:Y, S:25000 }
{ N:Jane Plain, A:44, D:19780814, P:N }
```

↑

Name	Type	Min Occurs	Max Occurs	Default Value
employees				
sequence		1	1	
employeeRecord		1	unbounded	
sequence		1	1	
name	string	1	1	
age	int	1	1	
dob	date	1	1	
permanent	boolean	1	1	true
salary	decimal	0	1	

© Copyright IBM Corporation 2013, 2015

## Understanding the logical structure of the data

Before you can model complex data in DFDL, you must understand the logical structure and the physical properties of the data. Understanding the logical structure of your data involves five stages.

1. Identify the complex structures that correspond to the complex types in the model. One complex type exists for the entire data itself. If the data contains substructures, such as records, a complex type exists per substructure.
2. Identify the simple items. Simple items occur within each complex type, and each has a logical data type. For example, each field in a COBOL copybook with a PIC clause, or each comma-separated value in a CSV message, corresponds to an element of a simple type.
3. Identify the structure order rules. This step determines whether the group within a complex type is a sequence or a choice. For example, if the fields in record must always appear in the same order, the group is a sequence group.
4. Identify complex structure and simple item cardinality. This step provides the values for the **minOccurs** and **maxOccurs** logical properties of the elements.
5. Identify nillable items and default values. It might be necessary for some elements to carry a special out-of-band value, in which case they must be nillable. For example, a numeric field

in a COBOL copybook might sometimes be set to SPACES, which is not legal for a DFDL number.

WebSphere Education

## Configuring the DFDL annotations (1 of 2)

- All elements
  - Delimiters: Initiator, Terminator, Encoding
  - Length establishment: Length Kind, Length
  - Number of occurrences: Occurs Count Kind, Occurs
  - Alignment rules: Alignment, Fill Byte
  - Nilable
  - Discriminator needed
- Simple elements
  - Text: Representation, Encoding, Text, Escape Scheme Ref
  - Binary: Representation, Byte Order
  - Type is string: textString
  - Type is number: text Number, binaryNumber
  - Type is Boolean: textBoolean, binaryBoolean
  - Type is Calendar: calendar, textCalendar, binaryCalendar
  - Split properties between Element and SimpleType

employeeRecord (Element)	
Property	Value
Comment	
<b>General</b>	
Encoding (code p	US-ASCII
Byte Order	bigEndian
<b>Content</b>	
Length Kind	implicit
<b>Occurrences</b>	
Min Occurs	1
Max Occurs	unbounded
Occurs Count Kind	implicit
<b>Delimiters</b>	
Initiator	{{
Terminator	}%CR;%LF;

© Copyright IBM Corporation 2013, 2015

## Configuring the DFDL annotations (1 of 2)

After the logical structure of your data is established, the DFDL annotations can be added to describe the physical format of the components.

For all elements, ask yourself these questions.

- Does the element have any delimiters, that is, an initiator or a terminator? If so what is the encoding, and are they present when the element is empty or nil?
- How is the content of the element established?

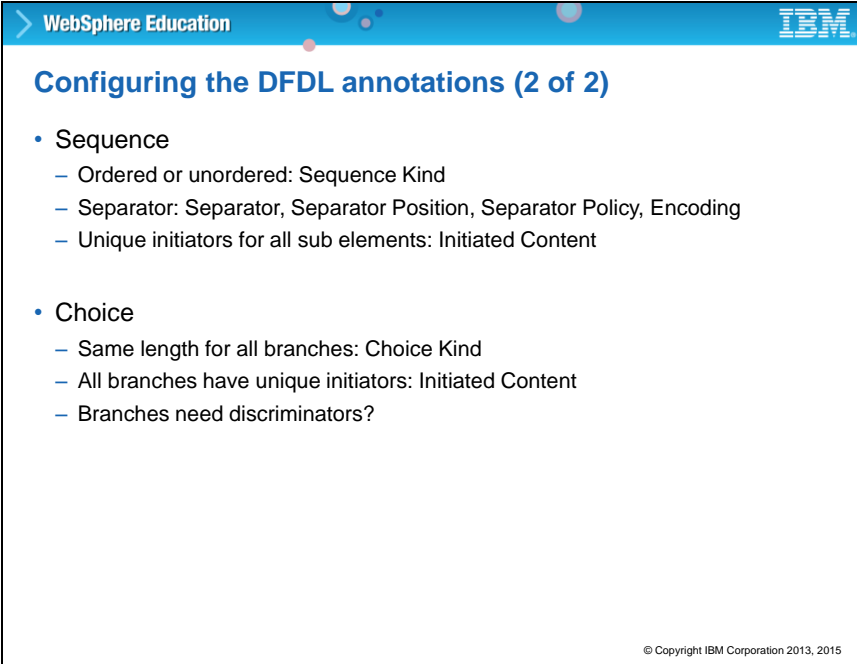
The answers determine the **length Kind** property:

- Select **explicit** for a fixed length element.
- If the data contains a length prefix, select **prefixed**.
- If the data is bounded by delimiter, select **delimited**.
- Select **pattern** to use a regular expression.
- If the data types determine the length, select **implicit**.

If the element is optional or is an array, then how is the number of occurrences established? Are there any alignment rules to apply? How is any nil value described? Is an assert or discriminator needed to establish whether the element exists?

For simple elements, ask yourself these questions about the data. Is the Element text or binary representation? This answer and its simple type determine the other properties that must be set.

You must know the answers to these questions before you attempt to model the data in the DFDL schema editor.



The slide is titled "Configuring the DFDL annotations (2 of 2)" and is part of a WebSphere Education presentation. It contains two main bullet points: "Sequence" and "Choice". The "Sequence" point has three sub-bullets: "Ordered or unordered: Sequence Kind", "Separator: Separator, Separator Position, Separator Policy, Encoding", and "Unique initiators for all sub elements: Initiated Content". The "Choice" point has three sub-bullets: "Same length for all branches: Choice Kind", "All branches have unique initiators: Initiated Content", and "Branches need discriminators?". The IBM logo is in the top right corner, and the copyright notice "© Copyright IBM Corporation 2013, 2015" is in the bottom right corner.

WebSphere Education

### Configuring the DFDL annotations (2 of 2)

- Sequence
  - Ordered or unordered: Sequence Kind
  - Separator: Separator, Separator Position, Separator Policy, Encoding
  - Unique initiators for all sub elements: Initiated Content
- Choice
  - Same length for all branches: Choice Kind
  - All branches have unique initiators: Initiated Content
  - Branches need discriminators?

© Copyright IBM Corporation 2013, 2015


## Configuring the DFDL annotations (2 of 2)

Next, you should consider how the data is structured.

For sequences, ask yourself these questions about the data. Is the sequence ordered or unordered? Does it have a separator that is used to delimit its components? If so, is the separator's position infix, prefix, or postfix, and are there any circumstances when separators are suppressed such as when optional elements are missing? Do all the subelements of the sequence have unique initiators that can identify that they exist? Does the sequence itself have an initiator or a terminator?

For choice groups, ask yourself these questions about the data. Is the choice one where all the branches must occupy the same length or not? Do all the branches of the choice have unique initiators that can identify which one appears? Are discriminators needed on the branches to establish which one appears? Does the choice itself have an initiator or a terminator?

Next, is an example of DFDL annotation configuration.

WebSphere Education 

### Configuring the DFDL annotations example

{ N:Joe Bloggs , A:50 , D:19620503 , P:Y , S:40000 }
{ N:Fred Smith , A:30 , D:19930225 , P:Y , S:25000 }
{ N:Jane Plain , A:44 , D:19780814 , P:N }

- **employees**  
Initiator = <no initiator>, Terminator = <no terminator>,  
Length Kind = implicit
- **employeeRecord**  
Initiator = {, Terminator = }%CR;%LF;, Encoding = ASCII,  
Length Kind = implicit, Occurs Count Kind = implicit
- **employeeRecord sequence**  
Sequence Kind = ordered, Separator = , , Separator Position = infix,  
Separator Policy = suppressedAtEnd
- **salary**  
Initiator = S:, Terminator = <no terminator>, Encoding = ASCII, Length Kind = delimited,  
Representation = text, Text Number Rep = standard, Text Number Pattern = #0.##
- **permanent**  
Initiator = P:, Terminator = <no terminator>, Encoding = ASCII, Length Kind = delimited,  
Representation = text, Text Boolean True Rep = Y, Text Boolean False Rep = N

Name
employees
sequence
employeeRecord
sequence
name
age
dob
permanent
salary

© Copyright IBM Corporation 2013, 2015

## Configuring the DFDL annotations example

This slide provides an example of DFDL annotation configuration. The upper portion of the example includes the sample data and the DFDL model. The bullets identify some of the DFDL elements and their key properties.

The top-level element, named **employees**, defines the entire file. The file does not contain an initiator or terminator, so those properties are not set. The **Length Kind** property is set to **implicit**, which means that the records in the file define the length of the **employees** element.



The **employeeRecord** element, defines a single record in the file. In the example data, each record begins with a '{' character and ends with a '}' character followed by a carriage return and line feed. In the DFDL Schema editor, the **Initiator** and **Terminator** properties are set to the appropriate values. The **Length Kind** property is set to **implicit**, which means that the fields in the **employeeRecord** element define the length of the element.

The **employeeRecord sequence** element defines the **employeeRecord** element as an ordered sequence, which means that fields are always in the same order. The separator is a comma and its position is infix, which means that it appears between each field.

The S: characters identify the **salary** field. The **Text Number Pattern** property identifies the implied format of the salary value.

The P: characters prefix the **permanent** field. The **Text Boolean True Rep** and **Text Boolean False Rep** identify the true and false values.



 WebSphere Education 

### DFDL points of uncertainty

- DFDL parser is a recursive-descent parser with look-ahead for resolving 'points of uncertainty':
  - A choice
  - An optional element
  - A variable array of elements
- DFDL parser must speculatively attempt to parse data until an object is either '*known to exist*' or '*known not to exist*'
  - Until that applies, the occurrence of a processing error causes the parser to suppress the error, back track, and then make another attempt
- Discriminator annotation can be used to assert that an object is '*known to exist*', which prevents incorrect back tracking
- Initiators can also assert '*known to exist*'


© Copyright IBM Corporation 2013, 2015

### DFDL points of uncertainty

When the DFDL parser encounters constructs such as a choice, an optional element, or a variable array, it might not be able to determine whether it can parse those constructs.

The parser tries to parse the data until it can determine for certain whether an object exists or does not exist. When the parser cannot determine either of those states, it can cause the parser to backtrack and try parsing the object by a different means. This backtracking increases the expense of the parsing operation.

To reduce the amount of backtracking, you can add annotations or initiators to the schema as a "hint" to the parser. You can also add a discriminator annotation that helps to resolve a point of uncertainty.

WebSphere Education 

### DFDL points of uncertainty example

```

<xs:choice>
  <xs:element name="Update" >
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Type" type="xs:int" dfdl:representation="binary"
...
          <xs:annotation><xs:appinfo source="http://www.ogf.org/dfdl/" >
            <dfdl:discriminator test="{. eq 1}" />
          </xs:appinfo></xs:annotation>
        </xs:element>
        ...
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Create" >
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Type" type="xs:int" dfdl:representation="binary"
...
          <xs:annotation><xs:appinfo source="http://www.ogf.org/dfdl/" >
            <dfdl:discriminator test="{. eq 2}" />
          </xs:appinfo></xs:annotation>
        </xs:element>
        ...
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:choice>

```

Discriminator resolves the choice

© Copyright IBM Corporation 2013, 2015

### DFDL points of uncertainty example

This slide is an example of the source for a DFDL schema. It uses a discriminator to resolve the point of uncertainty, which is a choice.

In this example, the data that this DFDL schema defines can contain **Update** type data when the **Type** field is set to '1', or **Create** type data when the **Type** field is set to '2'. Discriminators are used to resolve the choice.

You can add discriminators to the DFDL schema in the DFDL Schema editor in the Integration Toolkit.

WebSphere Education
IBM

## Adding a discriminator to a DFDL element

1. In the DFDL Schema editor, select the DFDL schema element on which you want to add a discriminator.
2. On the **Asserts and Discriminators** tab of the DFDL **Properties** area, click **Discriminator**.
3. Click **Add discriminator**.
4. Enter the test expression in the **Test Condition** field, and a human-readable failure message in the **Message** field.
  - Content assistance is available for the **Test Condition** field
  - Type Ctrl+Space to open the Path Expression Builder

Representation Properties
Variables
Asserts and Discriminators
sequence

Asserts

Assert defines a test to be used to ensure the data are well formed. Assert is used only when parsing data. Only asserts with test expressions are supported in the current IBM DFDL implementation.

Test Kind	Test Condition	Message
<a href="#">Add assert</a>		

Discriminator

Discriminator defines a test to be used when resolving a point of uncertainty such as choice branches or optional elements. Discriminator is used only when parsing data to resolve the point of uncertainty to one of the alternatives. Only discriminators with test expressions are supported in the current IBM DFDL implementation.

Test Kind	Test Condition	Message
<a href="#">Add discriminator</a>		

© Copyright IBM Corporation 2013, 2015

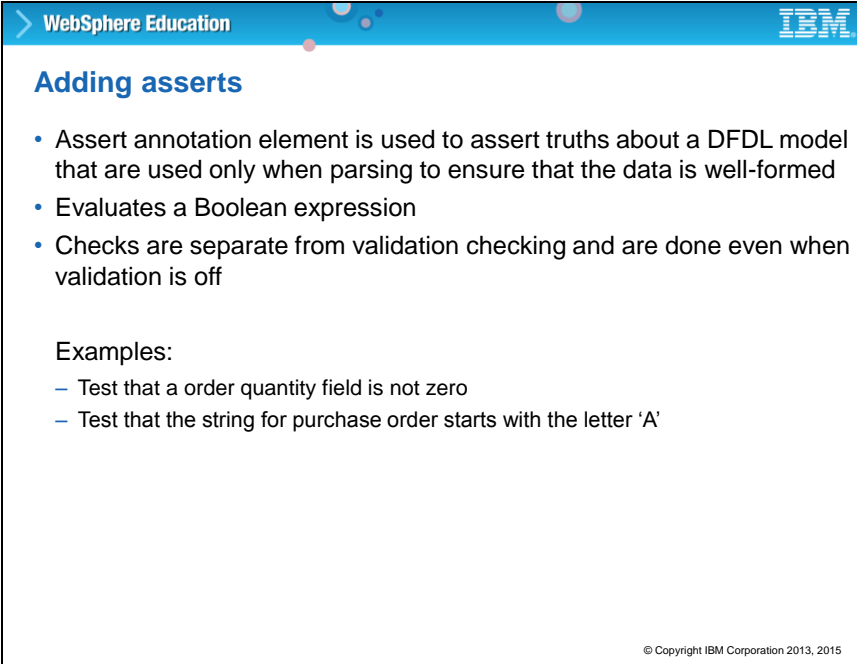
## Adding a discriminator to a DFDL element

You can add a discriminator to a schema object on the **Asserts and Discriminators** tab in the Integration Toolkit DFDL Schema editor.

To add a discriminator to an element:

1. Select the DFDL schema element.
2. On the **Asserts and Discriminators** tab of the DFDL properties area, click **Discriminators**.
3. Click the **Add discriminator** link or the green plus sign.
4. Enter the test expression in the **Test Condition** field, and a human-readable failure message in the **Message** field. Content assistance is available for the **Test Condition** field; press Ctrl+Space to open the XPath Expression Builder.

As an alternative, you can add asserts to an element to resolve points of uncertainty during parsing.



The slide is titled "Adding asserts" and is part of a WebSphere Education presentation. It contains a bulleted list of three points and two examples. The IBM logo is in the top right corner, and the copyright notice "© Copyright IBM Corporation 2013, 2015" is at the bottom right.

WebSphere Education

### Adding asserts

- Assert annotation element is used to assert truths about a DFDL model that are used only when parsing to ensure that the data is well-formed
- Evaluates a Boolean expression
- Checks are separate from validation checking and are done even when validation is off

Examples:

- Test that a order quantity field is not zero
- Test that the string for purchase order starts with the letter 'A'

© Copyright IBM Corporation 2013, 2015

## Adding asserts

You can add asserts to a DFDL schema object to define tests to ensure that the data is well-formed.

In the examples on the figure, the first assert tests the data to ensure that the value is not zero. The second assert tests for a precondition violation. If the expression evaluates to true, parsing continues. If the expression evaluates to false, a processing error is raised.

WebSphere Education
IBM

## Adding an assert to DFDL element

1. In the DFDL Schema editor, select the DFDL schema element on which you want to add an assert.
2. On the **Asserts and Discriminators** tab of the DFDL **Properties** area, click **Asserts**.
3. Click **Add assert**.
4. Enter the test expression in the **Test Condition** field, and a human-readable failure message in the **Message** field.
  - Content assistance is available for the **Test Condition** field
  - Type Ctrl+Space to open the Path Expression Builder

Representation Properties
Variables
Asserts and Discriminators
sequence

Asserts

Assert defines a test to be used to ensure the data are well formed. Assert is used only when parsing data. Only asserts with test expressions are supported in the current IBM DFDL implementation.

Test Kind	Test Condition	Message
<a href="#">Add assert</a>		

Discriminator

Discriminator defines a test to be used when resolving a point of uncertainty such as choice branches or optional elements. Discriminator is used only when parsing data to resolve the point of uncertainty to one of the alternatives. Only discriminators with test expressions are supported in the current IBM DFDL implementation.


Test Kind	Test Condition	Message
<a href="#">Add discriminator</a>		

© Copyright IBM Corporation 2013, 2015

## Adding an assert to DFDL element

This slide lists the steps for adding an assert to a DFDL element. An XPath Expression builder is available to help you create the assert statement.

You cannot add both discriminators and asserts to an element.

WebSphere Education 

### Asserts with recoverable errors

- If the **Test Condition** expression evaluates to “false”, the **Failure Type** property determines the error type:
  - Processing error** generates an exception and stops processing the data
  - Recoverable error** logs the error and continues processing the data
- In a message flow, DFDL recoverable errors are handled like runtime validation errors but are generated regardless of whether runtime validation is enabled

Name	Type	Min Occurs	Max Occurs
Unordered			
sequence		1	1
Type1		1	unbounded
sequence		1	1
A1	string	1	1
B1	string	1	1
C1	string	1	1
Type2		1	unbounded

**Asserts**

Assert defines a test to be used to ensure the data are well formed. A when parsing data. Only asserts with test expressions are supported in DFDL implementation.

Test Kind	Test Condition	Message	Failure Type
expression	{fn:contains (,'a')}	No 'a'	recoverableError

[Add assert](#)

© Copyright IBM Corporation 2013, 2015


### Asserts with recoverable errors

If the assertion identifies invalid data, you can specify the action when you add an assert,.

If you specify “processing error”, an error generates an exception and stops processing the data.

If you specify “recoverable error”, the error is logged but processing continues. In a message flow, DFDL recoverable errors are handled like runtime validation errors but are generated regardless of whether runtime validation is enabled.

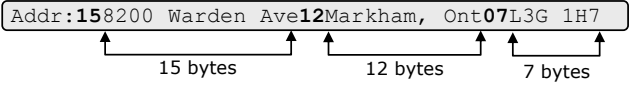
Selecting the “recoverable error” option is useful if you need to check the physical data instead of the logical structure of the data. For example, you might want to check the text length of a number or calendar data type but you do not want to stop the parser if the text length is not within the expected range.

WebSphere Education 

### Length prefixes

- When a prefix to the element contains the length of the element itself

Example: `Addr:158200 Warden Ave12Markham, Ont07L3G 1H7`



- Common variations
  - Value in the length prefix represents the length of the element to which it refers
  - Value in the length prefix includes the length of the prefix and that of the element
  - Length prefix has different characteristics from the element

© Copyright IBM Corporation 2013, 2015


## Length prefixes

Some data might contain prefixes that identify the number of bytes of data that follows the prefix.

In the example, the first 2 characters of the **Addr** element specify the number of bytes of data that follows. The next field begins after the 15 bytes of data. In the example, the next field has a length prefix of 12, which means that the next 12 bytes are data.

The value in the length prefix might represent the length of the element to which it refers, or the value in the length prefix might include the length of the prefix and that of the element. The length prefix might have different characteristics from the element, for example it might be a binary prefix whereas the element is text. It is even possible for a length prefix to have another length prefix provide its own length.

Many common data types have length prefixes. The next slide shows you how to define length prefixes in the schema editor.

WebSphere Education 

### Length prefixes example

Addr:158200 Warden Ave12Markham, Ont07L3G 1H7

- Define simple type element that is named **TwoCharsText** with physical characteristics of the prefix.

Content	short
Representation	text
Length Kind	explicit
Length	2
Length Units	character
Text Content	
Text Number Representation	standard
Number Pattern	00

- Modify the **Content** elements to refer to the new simple type element.

Content	string
Representation	text
Length Kind	prefixed
Length Units	characters
Prefix Length Type	TwoCharsText
Prefix Includes Prefix Length	no

Name of element that is defined in Step 1

© Copyright IBM Corporation 2013, 2015

### Length prefixes example


How do you handle length prefixes when you are modeling your data with DFDL?

First, create a simple type that represents the prefix data. In the example in the figure, the simple type that represents the prefix is named **TwoCharsText**.

Second, for any elements that have a prefix, modify the **Content** elements:

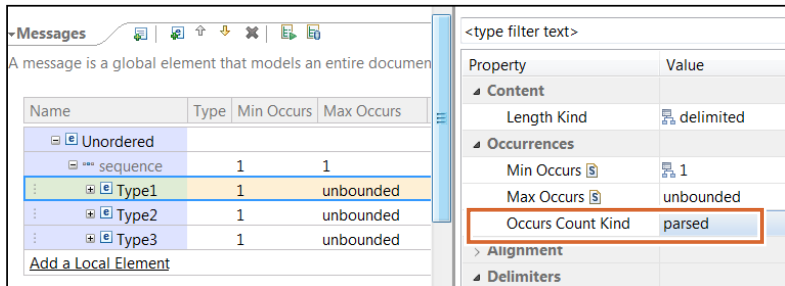
1. Set the **Length Kind** to prefixed.
2. Set the **Length Units** to characters.
3. Set the **Prefix Length Type** to the simple type that represents the prefix data.



WebSphere Education 

## Parsed arrays

- DFDL parser determines the number of occurrences by parsing the data instead of referencing **minOccurs** and **maxOccurs**
  - Specified by setting **Occurs Count Kind** to **parsed**
  - Can specify on optional elements and array elements



The screenshot shows a message structure in the 'Messages' tab. The structure is defined as follows:

Name	Type	Min Occurs	Max Occurs
Unordered			
sequence		1	1
Type1		1	unbounded
Type2		1	unbounded
Type3		1	unbounded

The 'Occurs Count Kind' property is highlighted in the 'Occurrences' section of the properties pane, set to 'parsed'.


© Copyright IBM Corporation 2013, 2015

## Parsed arrays

For most data, the element's **Min Occurs** and **Max Occurs** properties determine the number of occurrences of an element in the data stream.

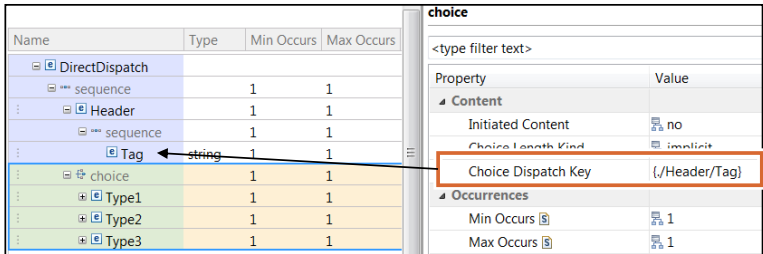
In the IBM implementation of DFDL, you can specify that you want the DFDL parser to determine the number of occurrences of an element. For example, the number of occurrences of an element might be determined by using an initiator or a separator.

If the number of occurrences varies and the parser should determine the number automatically, set the **Occurs Count Kind** property to **parsed**.

WebSphere Education 

## Direct dispatch choice

- Identifies choice branch to take during parsing
  - DFDL parser evaluates the expression and jumps to the indicated branch, without speculative parsing
- Example: An element in the data identifies the choice branch
- Specified by setting:
  - Choice Dispatch Key** on **choice** element to a string
  - Choice Branch Key** on each element in the **choice** group to a string that uniquely identifies that branch



The screenshot displays the IBM WebSphere DFDL editor interface. On the left, a tree view shows the structure of a **DirectDispatch** choice. It includes a **sequence** containing a **Header** (sequence) and a **Tag** (string). Below this is a **choice** element with three branches: **Type1**, **Type2**, and **Type3**. The **Tag** element is highlighted with a blue arrow pointing to the **Choice Dispatch Key** property in the right-hand properties pane. The **Choice Dispatch Key** is set to `./Header/Tag`, which is highlighted with a red box. The **Choice Branch Key** for each branch is also visible in the properties pane.


© Copyright IBM Corporation 2013, 2015

## Direct dispatch choice

In some data that contains a choice group, a field that appears earlier in the data stream indicates the branch to take in the logical data model. For example, your data model might support files that can contain both purchase orders and invoice records. A field in the data stream might identify whether this data stream is for a purchase order or for an invoice.

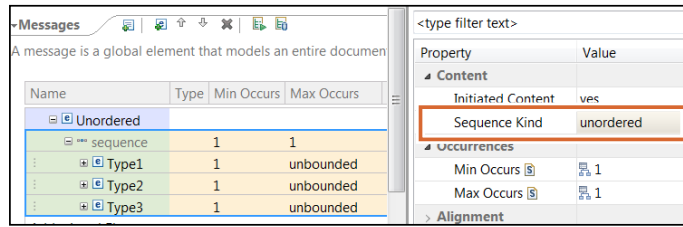
If the data contains some data that identifies the choice branch to take in the DFDL model, the **Choice Dispatch Key** property of the choice can be used to look at the data. It then can match it to the **Choice Branch Key** property of one of the branches. If a match is found, then that choice branch is deemed to be present. If a subsequent component in that choice branch fails to parse, it is a real parsing error in that choice branch and no backtracking takes place.

In the example, the **Choice Dispatch Key** for **choice** element, identifies the **Header Tag** element as the element that identifies whether the data is Type1, Type2, or Type3.

WebSphere Education 

## Unordered sequences

- Elements can occur in the data in any order, and are arranged into schema order in the tree by the DFDL parser
- Specified by setting **Sequence Kind** to **unordered**
  - Only elements are allowed in the sequence
  - Must have unique name/namespace
  - If optional or array element, **Occurs Count Kind** must be set to **parsed**
- Order of the data is not preserved in the message tree
- Data in the message tree must be in schema order before serializing



The screenshot shows the WebSphere Education interface. On the left, a table lists the elements of a message: 'Unordered' (Type: sequence, Min Occurs: 1, Max Occurs: 1), 'Type1' (Type: Type1, Min Occurs: 1, Max Occurs: unbounded), 'Type2' (Type: Type2, Min Occurs: 1, Max Occurs: unbounded), and 'Type3' (Type: Type3, Min Occurs: 1, Max Occurs: unbounded). On the right, the 'Sequence Kind' property is highlighted in orange and set to 'unordered'.

© Copyright IBM Corporation 2013, 2015

## Unordered sequences


In a DFDL unordered sequence, elements can occur in the data in any order. A DFDL unordered sequence is specified by setting the **Sequence Kind** property to **unordered** on the **sequence** element.

If you identify a sequence element as unordered, some restrictions apply to the content of the sequence element:

- Only elements are allowed in the sequence.
- Each element must have a unique name-namespace combination.
- If the element is optional or is an array, **Occurs Count Kind** must be set to **parsed**.

The order in the data is not preserved in the DFDL logical model; the model is static. After parsing, the data is arranged in the order that is declared in the DFDL schema.

On output, the data in the message tree must be in schema order before serializing.

WebSphere Education


### Adding a reference to another schema (1 of 3)

- Create reference to another DFDL schema in the workspace when there are DFDL objects in one schema file that you want to refer to in another schema
- Namespaces of the two schema files determine whether to use the **import** or **include** option

	Target file has a target namespace	Target file has no target namespace
Parent file has target namespace	<code>xsd:import</code>	<code>xsd:include</code>
Parent file has no target namespace	<code>xsd:import</code>	<code>xsd:include</code>

© Copyright IBM Corporation 2013, 2015

### Adding a reference to another schema (1 of 3)

You can create a schema by reusing message objects from another DFDL schema file in the workspace.

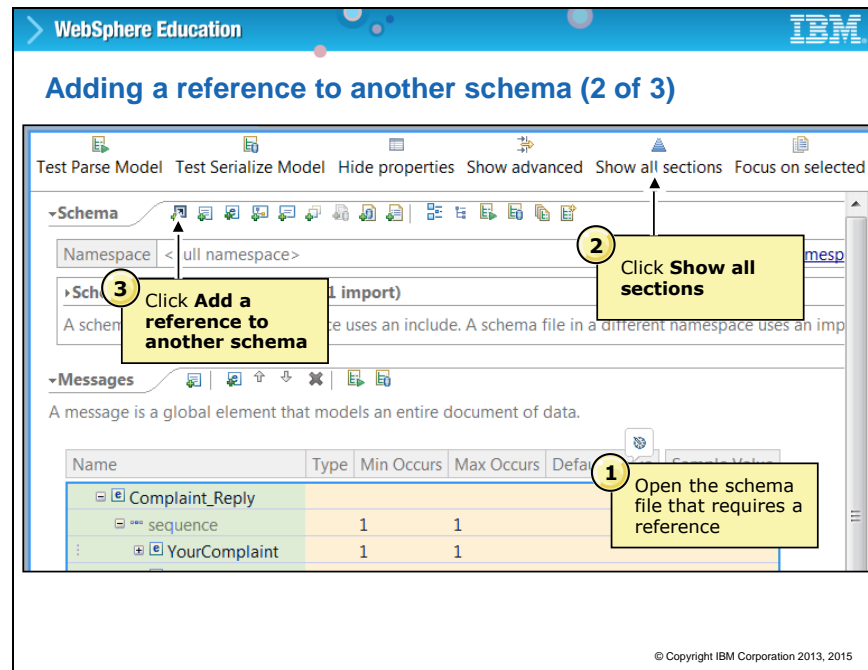
Two mechanisms for reusing message definition files are available. You can include a reference to the message definition or you can import the message definition.

As shown in the table, the namespace of the schemas determines whether the import or include command is used.

When a target namespace file includes a “no target” namespace file, referencing an object in the target file from the parent file causes the object to be present in the namespace of the parent file. When import or include are used, global objects from the target file can be used in the parent file.

The namespace of objects in the target file is preserved in the parent file, with the exception noted in the previous table of a target namespace file that includes a “no target” namespace file.

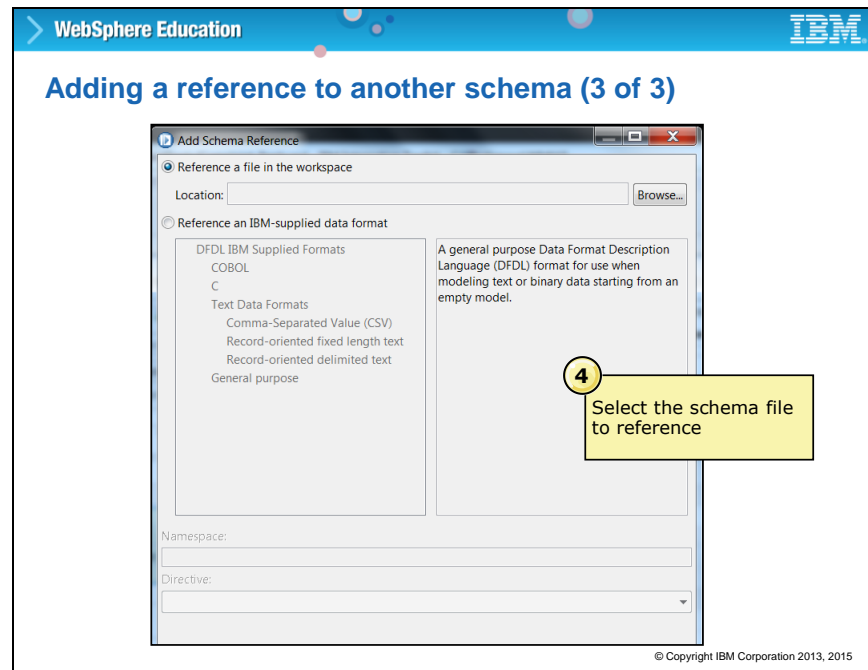
The next series of slides show you how to add a reference to a DFDL schema.



## Adding a reference to another schema (2 of 3)

To add a reference to a schema:

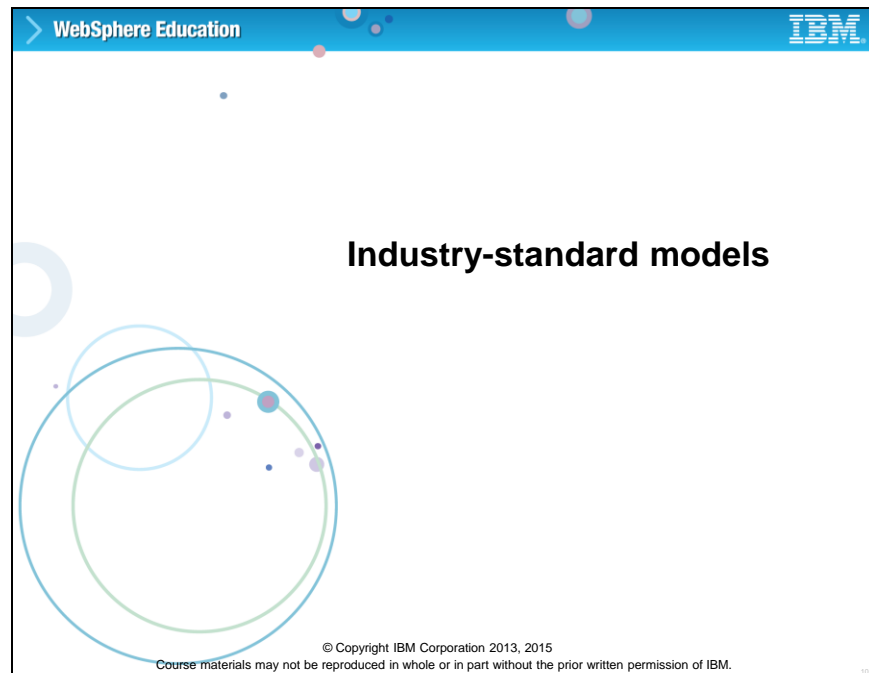
1. Open the DFDL schema file that requires a reference in the DFDL Schema editor.
2. Click **Show all sections** in the toolbar to show all the **Schema** section.
3. In the **Schema** section, click the **Add a reference to another schema** icon.



### Adding a reference to another schema (3 of 3)

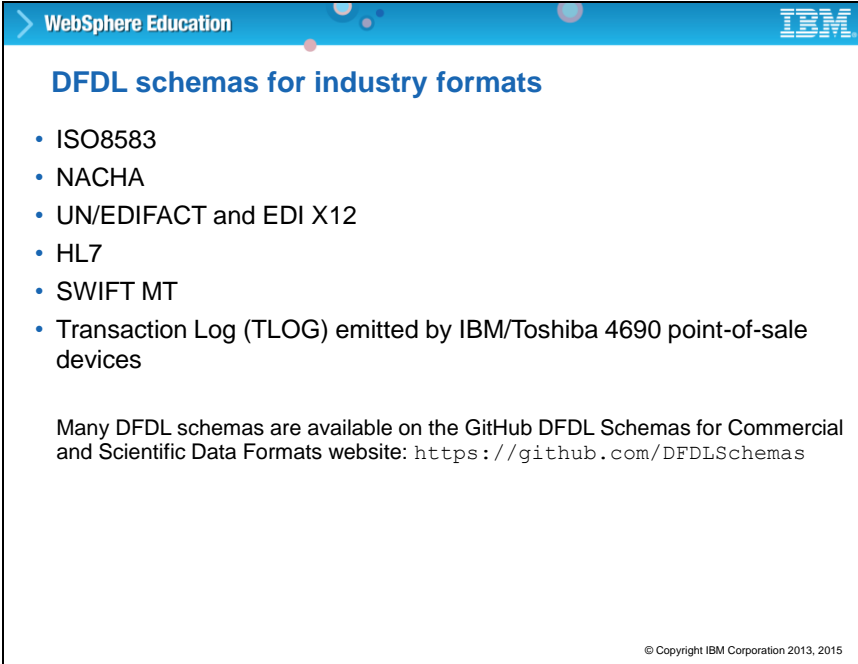
The next step is to select the schema file to reference from the workspace. As an option you can reference an IBM-supplied schema.

In the exercise at the end of this unit, you get hands-on experience with an existing DFDL model in a new model.



### Topic 3: Industry-standard models

If you are working with industry-standard data, a DFDL schema that defines the data might exist. This topic provides an overview of the industry-standard DFDL schemas that are available for use with IBM Integration Bus.



WebSphere Education

### DFDL schemas for industry formats

- ISO8583
- NACHA
- UN/EDIFACT and EDI X12
- HL7
- SWIFT MT
- Transaction Log (TLOG) emitted by IBM/Toshiba 4690 point-of-sale devices

Many DFDL schemas are available on the GitHub DFDL Schemas for Commercial and Scientific Data Formats website: <https://github.com/DFDLSchemas>

© Copyright IBM Corporation 2013, 2015

## DFDL schemas for industry formats


The figure lists the DFDL schemas that are available for industry-standard formats.

- ISO 8583 is the International Organization for Standardization (ISO) standard for systems that exchange cards.
- NACHA manages the development, administration, and governance of the backbone for the electronic movement of money and data in the United States.
- EDIFACT (ISO 9735) is the international standard for electronic data interchange (EDI). The term stands for Electronic Data Interchange For Administration, Commerce, and Transport.
- HL7 is a standard for exchanging information between medical applications. The DFDL schema for HL7 is available in the IBM Integration Bus Healthcare Pack.
- SWIFT supplies secure, standardized messaging services and interface software to financial institutions.
- IBM/Toshiba 4690 SurePOS ACE is point-of-sale application.

Many of the DFDL schemas are available on GitHub. GitHub is a web-based hosting service for software development projects that use the Git revision control system.

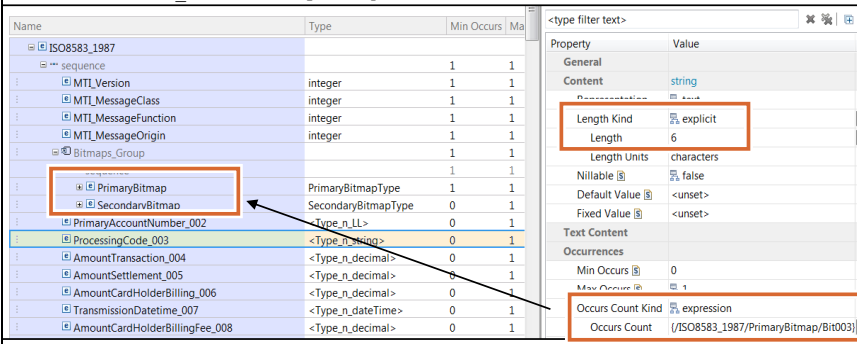
The next series of figures have more information on the schemas that are listed on this slide.



WebSphere Education 

## DFDL for ISO 8583

- Text/binary format for ATM and credit card transactions
  - Message consists of a flat structure of simple data fields
  - Data fields are either fixed length or variable length with a prefix
- Most data fields are optional (Min Occurs '0') but there are no delimiters
- A flag in a special bitmap indicates the presence of a field in the data
  - Occurs Count Kind = expression, Occurs Count =**  
(/ISO8583\_1987/PrimaryBitmap/Bitxxx)



© Copyright IBM Corporation 2013, 2015


## DFDL for ISO 8583

ISO 8583 is a standard for systems that exchange cards. It has three parts:

- Messages, data elements, and code values
- Application and registration procedures for Institution Identification Codes (IIC)
- Maintenance procedures for messages values

In the DFDL schema, most data fields are defined as optional but the data does not contain delimiters. Two simple types that are named **PrimaryBitmap** and **SecondaryBitmap** define the special bitmap that indicates the presence of a field in the data. Any elements that must use the special bitmap have the **Occurs Count Kind** and **Occurs Count** properties that are set to reference the bitmap types.

Configuration for the special bitmap is similar to the configuration for handling length prefixes that you learned about earlier in this unit.

WebSphere Education 

## DFDL for NACHA

- NACHA manages the development, administration, and governance of the backbone for the electronic movement of money and data in the United States
- There are different kinds of record but only one kind appears in a given batch
  - Use a choice with a discriminator on each branch
- All records are 94 characters long and usually terminated with a new line

Name	Type	Min Occurs	Max Occurs
NACHAFile			
sequence		1	1
FileHeaderRecord		1	1
Batch		1	unbounded
sequence		1	1
BatchHeaderRecord		1	1
EntryDetail		1	unbounded
choice		1	1
CCDEntryDetailRecord		1	1
PPDEntryDetailRecord		1	1
TELEntryDetailRecord		1	1
WEBEntryDetailRecord		1	1
CTXEntryDetailRecord		1	1
BatchControlRecord		1	1
FileControlRecord		1	1

Property	Value
General	
Data Format Reference	nachRecordWithT
Encoding (code page)	US-ASCII
Byte Order	bigEndian
Ignore Case	no
Fill Byte	%SP;
Content	
Length Kind	explicit
Length	94
Length Units	bytes
Occurrences	
Min Occurs	1

© Copyright IBM Corporation 2013, 2015

## DFDL for NACHA

NACHA manages the development, administration, and governance of the ACH Network, the backbone for the electronic movement of money and data in the United States.

NACHA records are fixed length.

A file Header record designates the physical file characteristics and identifies the immediate origin and destination of the entries contained within the file. In addition, this record includes date, time, and file identification fields that are used to identify the file uniquely.

A Batch Header record identifies the Originator and briefly describes the reason for the transaction originated by the Originator. This record also identifies bank routing information.

The detail records of a NACHA file contain the information necessary to route the entry to the Receiver. The NACHA file can contain different types of detail records but only one kind appears in a specific batch. For example, a NACHA file can contain Cash Concentration or Disbursement (CCD) records or Prearranged Payments and Deposits (PPD) records. So, the DFDL schema uses a choice group with a discriminator to identify the record type.

WebSphere Education

### DFDL for EDIFACT

- EDIFACT is the international standard for electronic data interchange (EDI)
- Schema models the entire EDIFACT interchange
- For parsing/serializing only, no validation rules
- Recommend 1 GB JVM heap size when deploying and for the Integration Toolkit

```

UNB+UNOA:1+005435656:1+006415160:1+
060515:1434+000000000000778'
UNH+00000000000117+INVOIC:D:97B:UN'
BGM+380+342459+9'
DTM+3:20060515:102'
RFF+ON:521052'
NAD+BY+792820524::16++CUMMINS MID-
RANGE ENGINE PLANT'
NAD+SE+005435656::16++GENERAL
WIDGET COMPANY'
CUX+1:USD'
LIN+1++157870:IN'
IMD+F++::WIDGET'
QTY+47:1020:EA'
ALI+US'
MOA+203:1202.58'
PRI+INV:1.179'
LIN+2++157871:IN'
IMD+F++::DIFFERENT WIDGET'
QTY+47:20:EA'
ALI+JP'
MOA+203:410'
PRI+INV:20.5'
. . .

```

**SAMPLE**

© Copyright IBM Corporation 2013, 2015


## DFDL for EDIFACT

EDIFACT messages are made up of a collection of sequenced segments within defined areas. Some segments can be used in more than one area. The EDIFACT documentation defines the segments that can be used in each area. EDIFACT provides a hierarchical structure for messages.

Messages begin with the Message Header UNH segment and end with the Message Trailer UNT segment. These two segments are the first, and innermost, level of the three levels of “electronic envelopes” within EDIFACT.

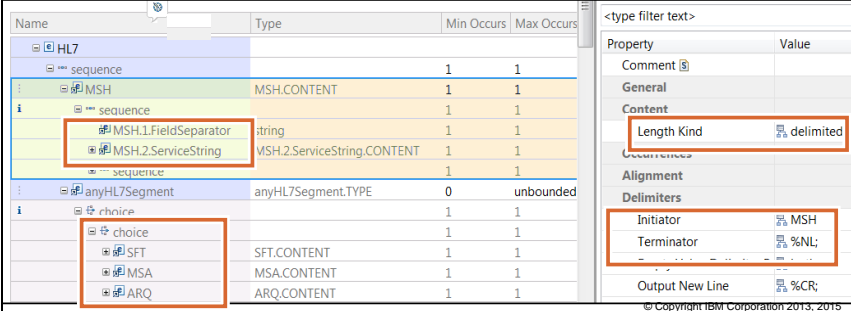
The EDIFACT DFDL schema can be included in a message flow to provide message parsing on input and serialization on output.

EDIFACT files can be large and the DFDL model is complex. It is suggested that you set the JVM heap size to a minimum of 1 GB when deploying message flow applications that process EDIFACT data.

WebSphere Education 

### DFDL for HL7 v2

- Delimited text format that is used in the healthcare industry
  - Message consists of an MSH segment followed by a number of other segments
  - A three-character tag identifies each segment, which is terminated by CR
  - Segments contain variable length fields that are terminated by a delimiter
  - Fields can be simple or complex, each level of nesting has its own delimiter
  - Fields can repeat and occurrences have their own delimiter ('~')
  - Delimiters are dynamically defined in the first (MSH) segment



Name	Type	Min Occurs	Max Occurs
HL7	sequence	1	1
MSH	sequence	1	1
MSH.1.FieldSeparator	string	1	1
MSH.2.ServiceString	sequence	1	1
anyHL7Segment	choice	1	1
SFT	SFT.CONTENT	1	1
MSA	MSA.CONTENT	1	1
ARQ	ARQ.CONTENT	1	1

Property Value

General

Content

Length Kind: delimited

Delimiters

Initiator: MSH

Terminator: %NL;


Output New Line: %CR;

© Copyright IBM Corporation 2013, 2015

### DFDL for HL7 v2

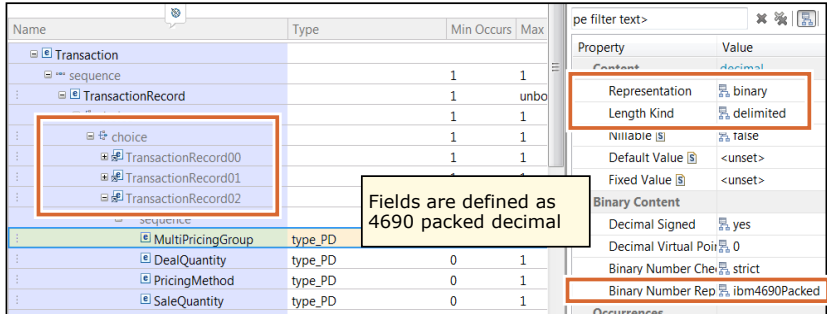
HL7 data is delimited text. Each message consists of an MSH segment and other segments.

The figure shows part of the DFDL schema for HL7. In the schema, a choice group and the **Initiator** property value determine the segment.

WebSphere Education 

## DFDL for TLOG

- A 'transaction log' consists of multiple different transaction records
- Each transaction record has a type (and some records have a subtype)
  - Use a choice with a discriminator on each branch
- Each transaction record is a sequence of delimited binary fields
- Most of the fields are a special packed decimal unique to 4690



Fields are defined as 4690 packed decimal


© Copyright IBM Corporation 2013, 2015

## DFDL for TLOG


The transaction logs that the 4690 emits are in a compact, ASCII delimited binary format, which is known as TLOG. Each TLOG consists of a number of records, called strings. Each string consists of fields that a colon delimits, although some strings use double quotation mark, colon, double quotation mark instead. The fields can be fixed length or variable length but the delimiter is always present. The fields can have a number of physical representations but the most frequent representation is a packed number representation.

The figure shows a portion of the DFDL schema. The schema includes a choice group for handling the different transaction types.

This DFDL schema is available in the IBM Integration Bus Retail Pack.



The slide is titled 'WebSphere Education' in the top left corner and features the IBM logo in the top right corner. The main heading is 'Useful DFDL links'. Below this heading is a bulleted list of seven links. The last link is a multi-line URL. At the bottom right of the slide, there is a small copyright notice.

WebSphere Education 

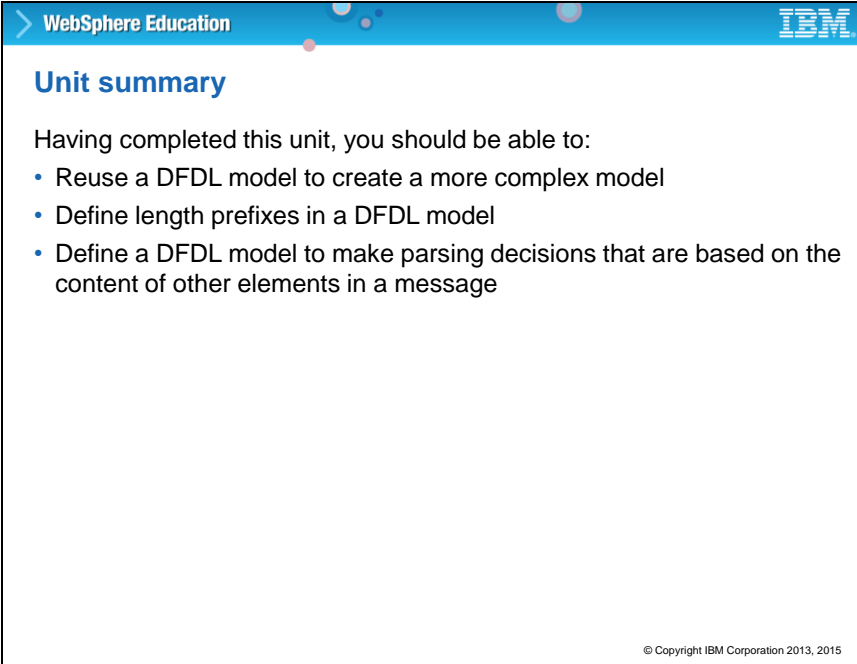
### Useful DFDL links

- OGF DFDL home page: <http://www.ogf.org/dfdl/>
- DFDL 1.0 specification: <http://www.ogf.org/documents/GFD.174.pdf>
- DFDL tutorials: [http://redmine.ogf.org/dmsf/dfdl-wg?folder\\_id=5485](http://redmine.ogf.org/dmsf/dfdl-wg?folder_id=5485)
- DFDL-WG Redmine project: <http://redmine.ogf.org/projects/dfdl-wg>
- DFDL Wikipedia page: <http://en.wikipedia.org/wiki/DFDL>
- DFDL Schemas on GitHub: <https://github.com/DFDLSchemas>
- Daffodil open source project:  
<https://opensource.ncsa.illinois.edu/confluence/display/DFDL/Daffodil%3A+Open+Source+DFDL>


© Copyright IBM Corporation 2013, 2015

### Useful DFDL links

It not possible to teach all there is to know about DFDL in one of two hours. This figure lists some websites that contain useful information about DFDL and DFDL models. The IBM Knowledge Center for IBM Integration Bus also contains information about the IBM implementation of DFDL.



The slide features a blue header bar with the text 'WebSphere Education' on the left and the IBM logo on the right. Below the header, the title 'Unit summary' is displayed in blue. The main content area contains a paragraph followed by a bulleted list. At the bottom right, there is a small copyright notice.

> WebSphere Education 

### Unit summary

Having completed this unit, you should be able to:

- Reuse a DFDL model to create a more complex model
- Define length prefixes in a DFDL model
- Define a DFDL model to make parsing decisions that are based on the content of other elements in a message

© Copyright IBM Corporation 2013, 2015

### Unit summary

You can model a wide variety of message formats by using DFDL schema files. In this unit, you learned how to use DFDL to model complex data that includes multiple records types, length prefixes, choice groups, optional elements, and variable array elements.

Having completed this unit, you should be able to:

- Reuse a DFDL model to create a more complex model.
- Define length prefixes in a DFDL model.
- Define a DFDL model to make parsing decisions that are based on the content of other elements in a message.