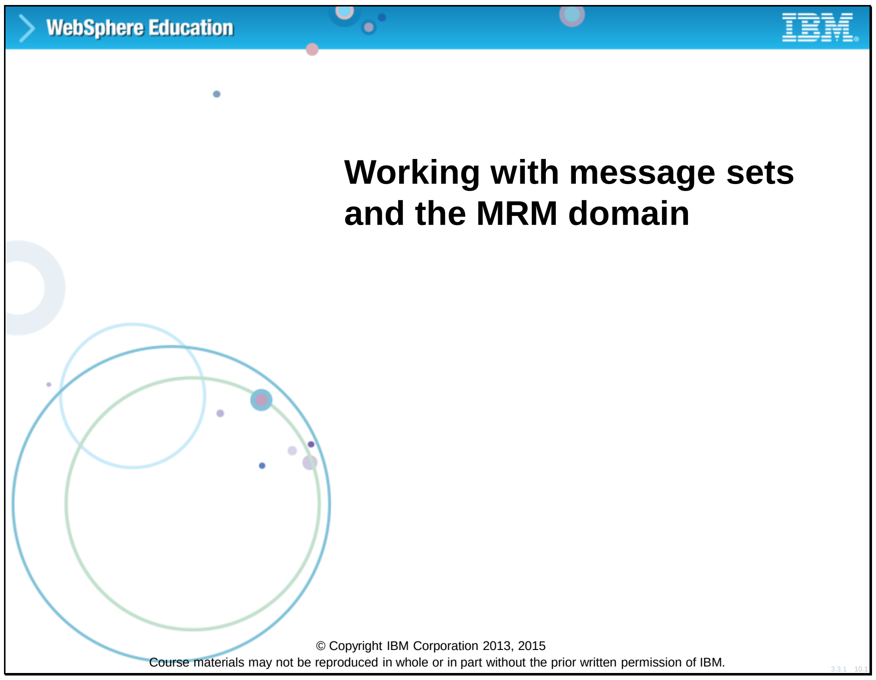
Slide 1

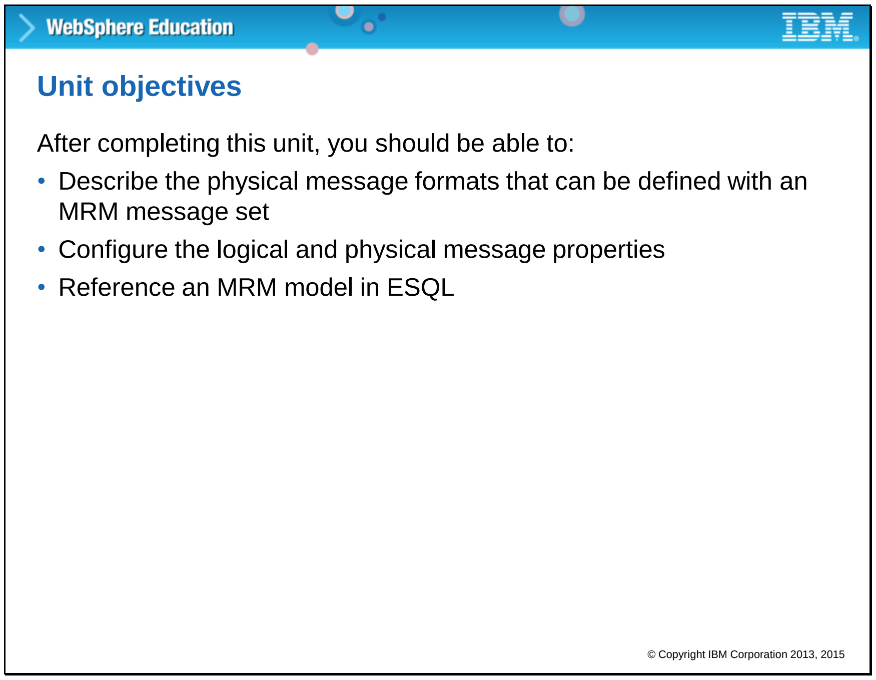


WebSphere Education

IBM

Working with message sets and the MRM domain

Slide 2



WebSphere Education

Unit objectives

After completing this unit, you should be able to:

- Describe the physical message formats that can be defined with an MRM message set
- Configure the logical and physical message properties
- Reference an MRM model in ESQL

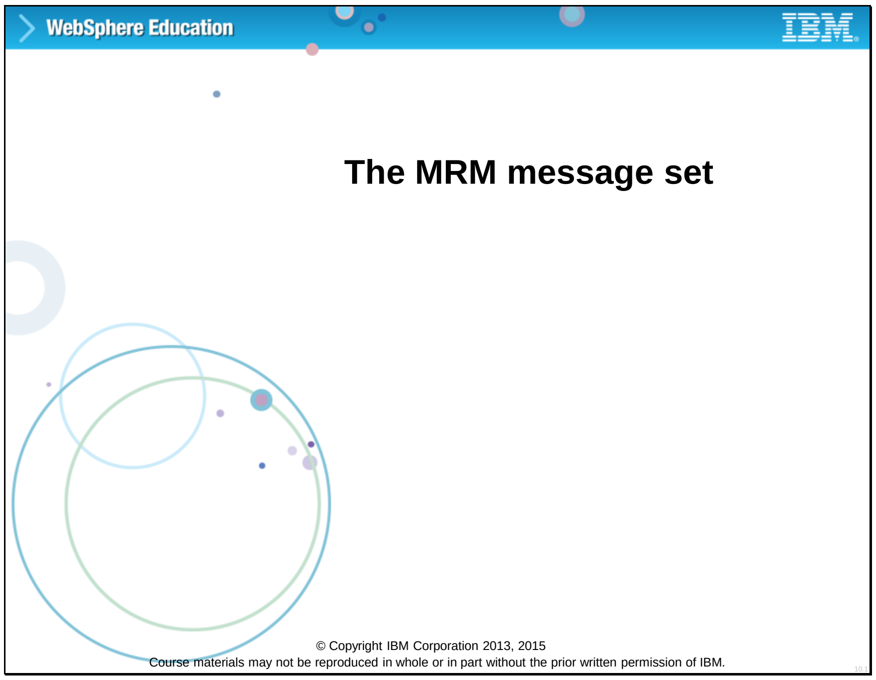
Unit objectives

A message set is the original container for message models that are used by WebSphere Message Broker. In IBM Integration Bus, DFDL schema files that are contained in applications and libraries are the preferred way to model messages for most data formats. Message sets continue to be supported, and if you use the MRM or IDOC domains are required. In this unit, you learn how to create a message set to define a message and use the MRM domain.

After completing this unit, you should be able to:

- Describe the physical message formats that can be defined with an MRM message set.
- Configure the logical and physical message properties.
- Reference an MRM model in ESQL.

Slide 3

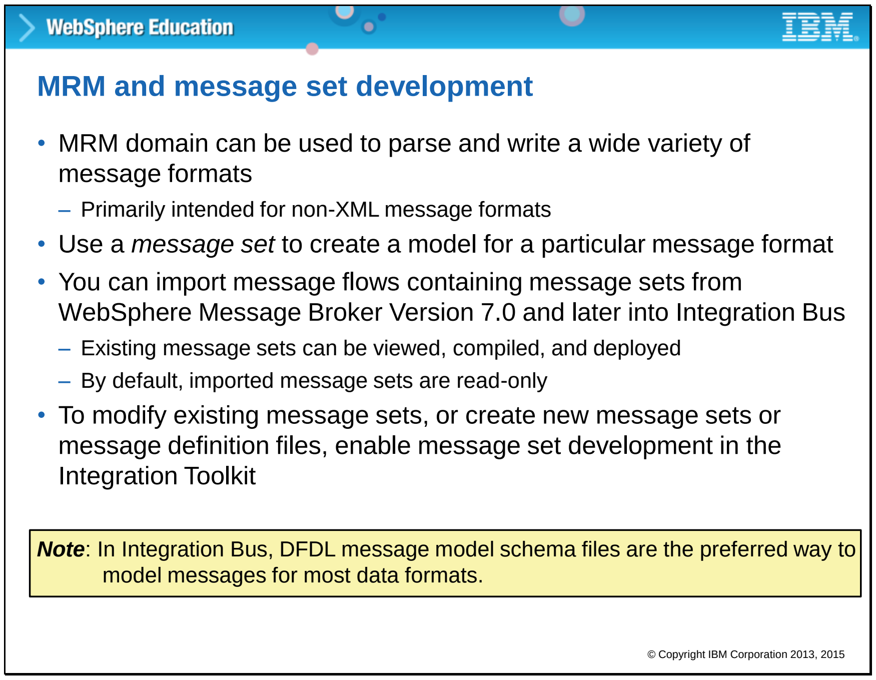

WebSphere Education

The MRM message set

Topic 1: The MRM message set

In this topic, you learn about the MRM message set and the physical message formats that it can define.

Slide 4


WebSphere Education

MRM and message set development

- MRM domain can be used to parse and write a wide variety of message formats
  - Primarily intended for non-XML message formats
- Use a *message set* to create a model for a particular message format
- You can import message flows containing message sets from WebSphere Message Broker Version 7.0 and later into Integration Bus
  - Existing message sets can be viewed, compiled, and deployed
  - By default, imported message sets are read-only
- To modify existing message sets, or create new message sets or message definition files, enable message set development in the Integration Toolkit

Note: In Integration Bus, DFDL message model schema files are the preferred way to model messages for most data formats.

MRM and message set development

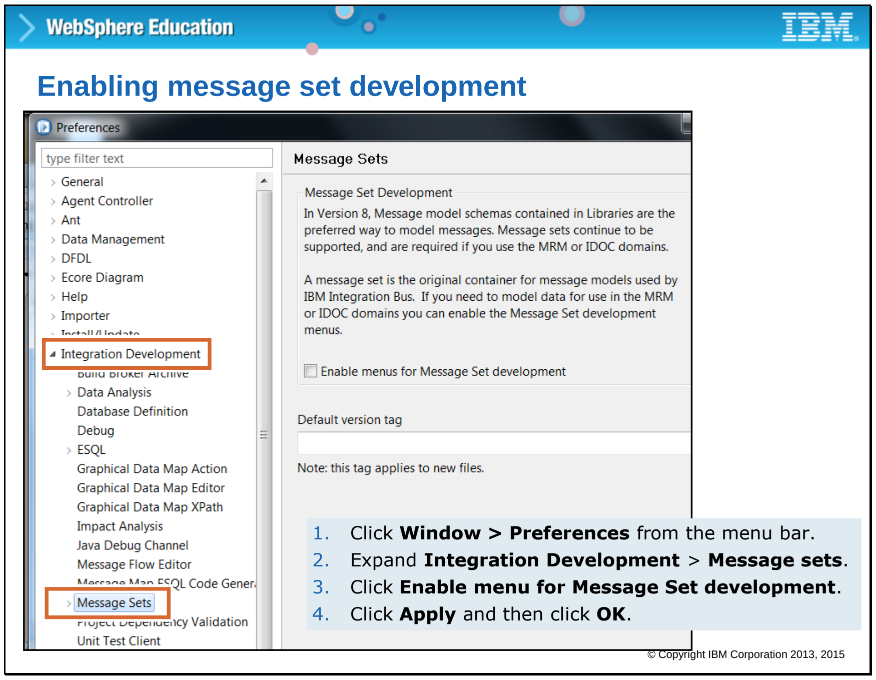
In Integration Bus, the MRM domain can be used to parse and write a wide variety of message formats but is primarily intended for non-XML message formats.

The message model for the MRM domain is contained in a message definition file. Each message definition file within a message set describes both the logical structure of the messages, and the physical formats that describe the precise appearance of your message bit stream during transmission.

In WebSphere Message Broker Version 8.0 and IBM Integration Bus, DFDL message model schema files are the preferred way to model messages for most data formats; however, message sets continue to be supported.

You can import message flows that contain message sets from WebSphere Message Broker Version 7.0 and later into Integration Bus. By default, existing message sets can be viewed, compiled, and deployed. You must enable message set development in the Integration Toolkit to modify existing message sets, or create message sets or message definition files.

Slide 5



WebSphere Education

Enabling message set development

In Version 8, Message model schemas contained in Libraries are the preferred way to model messages. Message sets continue to be supported, and are required if you use the MRM or IDOC domains.

A message set is the original container for message models used by IBM Integration Bus. If you need to model data for use in the MRM or IDOC domains you can enable the Message Set development menus.

☐ Enable menus for Message Set development

Default version tag

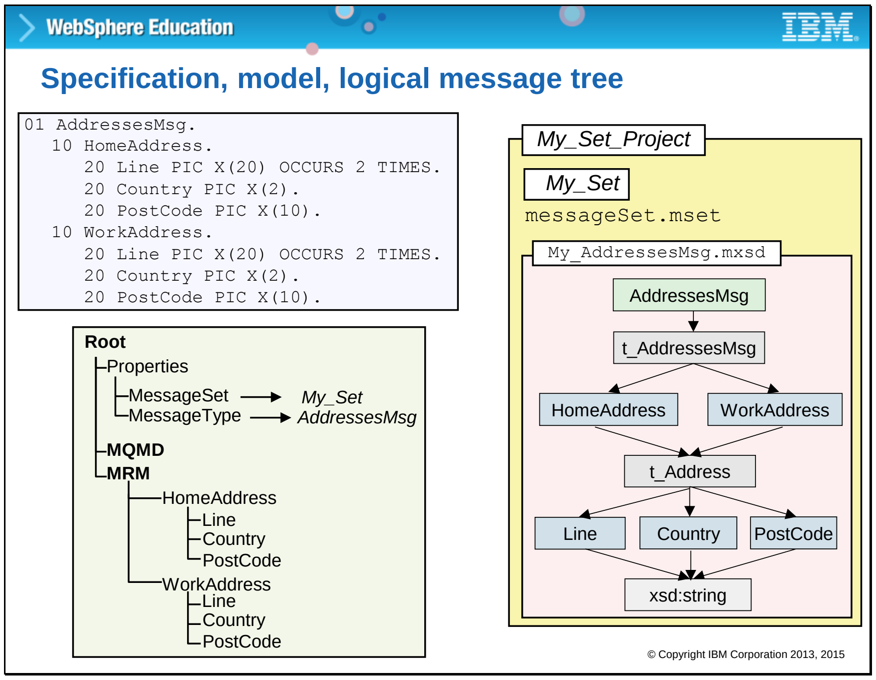
Note: this tag applies to new files.

1. Click **Window > Preferences** from the menu bar.
2. Expand **Integration Development** > **Message sets**.
3. Click **Enable menu for Message Set development**.
4. Click **Apply** and then click **OK**.

Enabling message set development

If you need to modify or create message definition files for the MRM domain, enable message set development in the Integration Toolkit **Message Sets Preferences**, as shown in this slide.

Slide 6



Specification, model, logical message tree

```
01 AddressesMsg.
   10 HomeAddress.
      20 Line PIC X(20) OCCURS 2 TIMES.
      20 Country PIC X(2).
      20 PostCode PIC X(10).
   10 WorkAddress.
      20 Line PIC X(20) OCCURS 2 TIMES.
      20 Country PIC X(2).
      20 PostCode PIC X(10).
```

```
Root
 ├Properties
 │ ├MessageSet  ──▶ My_Set
 │ └MessageType ──▶ AddressesMsg
 ├MQMD
 └MRM
    ├──HomeAddress
    │    ├Line
    │    ├Country
    │    └PostCode
    └──WorkAddress
         ├Line
         ├Country
         └PostCode
```

Specification, model, and logical message tree

The message models that are defined in the MRM are platform-independent and language-independent. Through more physical format layers, a logical message structure can be represented in various physical wire formats.

This slide shows an example of a COBOL copybook and the message definition and logical message tree that describes it.

A message set project contains the objects that describe the message.

A message set is a folder in a message set project that contains a logical grouping of your messages and the objects that comprise them (elements, types, groups).

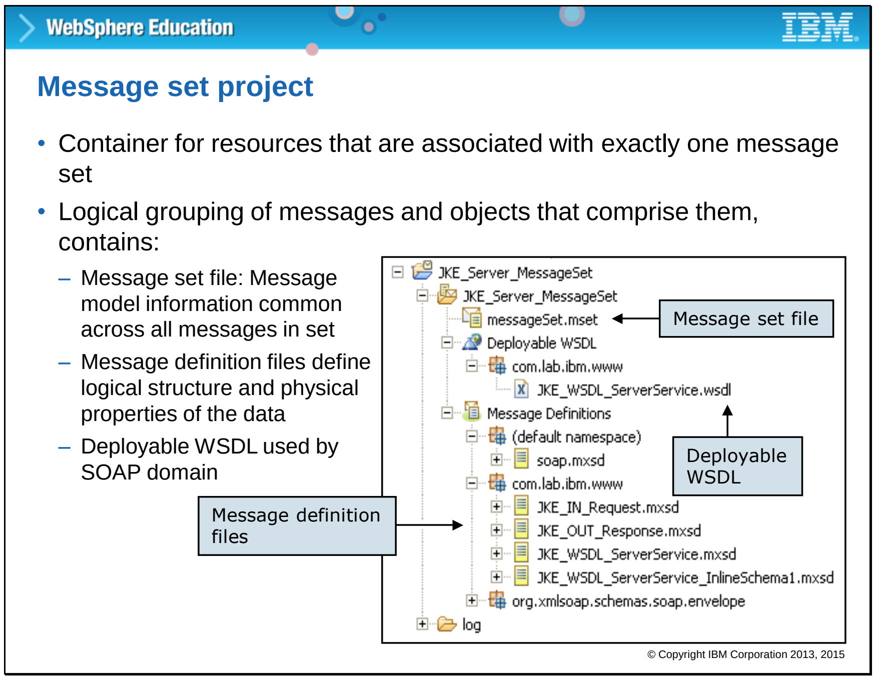
The message definition file contains the messages, elements, types, and groups that make up a message model within a message set. When used in a message flow, the message set, message type, and message definition file names are identified in the Properties folder in the logical message tree.

Similar to a DFDL schema, the top-level element in the message definition file is the message. Under the message element, you can define complex and simple elements. For example, WorkAddress is a complex element that contains Line, Country, and PostCode, which are simple elements.

You can also define global and temporary elements to speed the development of the model. For example, the HomeAddress and WorkAddress records contain the same elements. Instead of creating unique elements for each occurrence of Line, Country, and PostCode, a temporary element that is named t_Address defines an address record. The HomeAddress and WorkAddress in the message definition file references this temporary element.

You learn more about message sets and message definition files next.

Slide 7


WebSphere Education

Message set project

- Container for resources that are associated with exactly one message set
- Logical grouping of messages and objects that comprise them, contains:
  - Message set file: Message model information common across all messages in set
  - Message definition files define logical structure and physical properties of the data
  - Deployable WSDL used by SOAP domain

Message set project

A message set project is a specialized project in which you create and maintain all the resources that are associated with exactly one message set.

As mentioned previously, a message set is a logical grouping of your messages and the objects that comprise them (elements, types, groups). A message set contains the following files:

- Exactly one message set file
- Zero or more message definition files
- Zero or more WSDL files

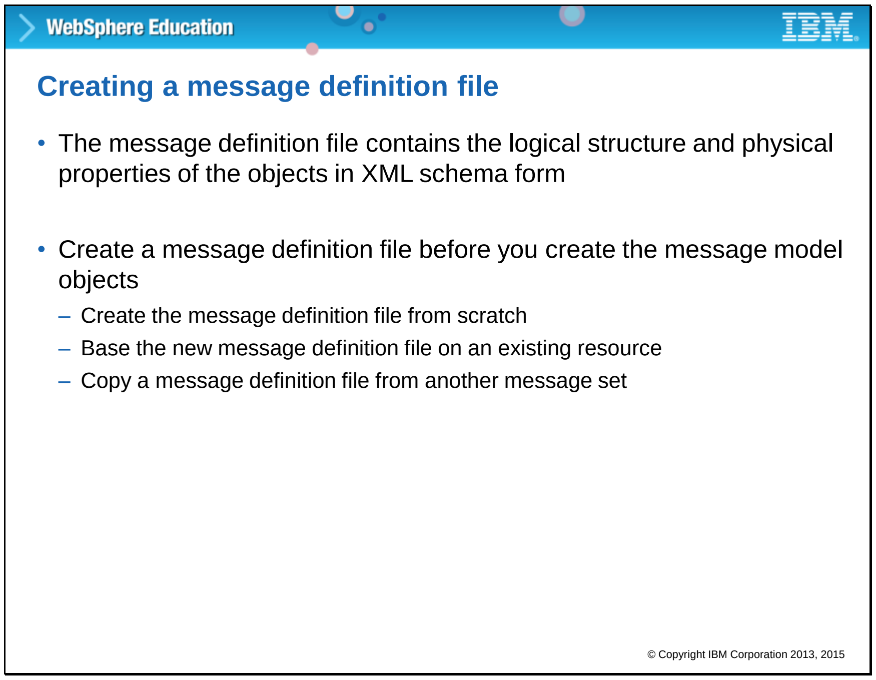
Resources within a message set are created as files, and are shown under the message set folder in the **Application Development** view. A message set is defined as a folder in a message set project that contains a messageSet.mset file. The name of the folder is the name of the message set. You can base your new message set on an existing message set, which copies all the definitions in the existing message set into the new message set.

You can have many message definition files in a message set. Each file contains the logical model, and the associated physical model in XML schema form, for a group of related messages.

You can optionally group messages into message categories (files with extension.category) for convenience. You can add messages to message categories by using the message category editor. In previous versions of the product, message categories were necessary to create WSDL.

Each time that you save a message set file, message definition file, or message category file, the content is validated to ensure that the message model you are creating follows certain rules. Rules for both the logical structure and the physical formats exist.

Slide 8


WebSphere Education

Creating a message definition file

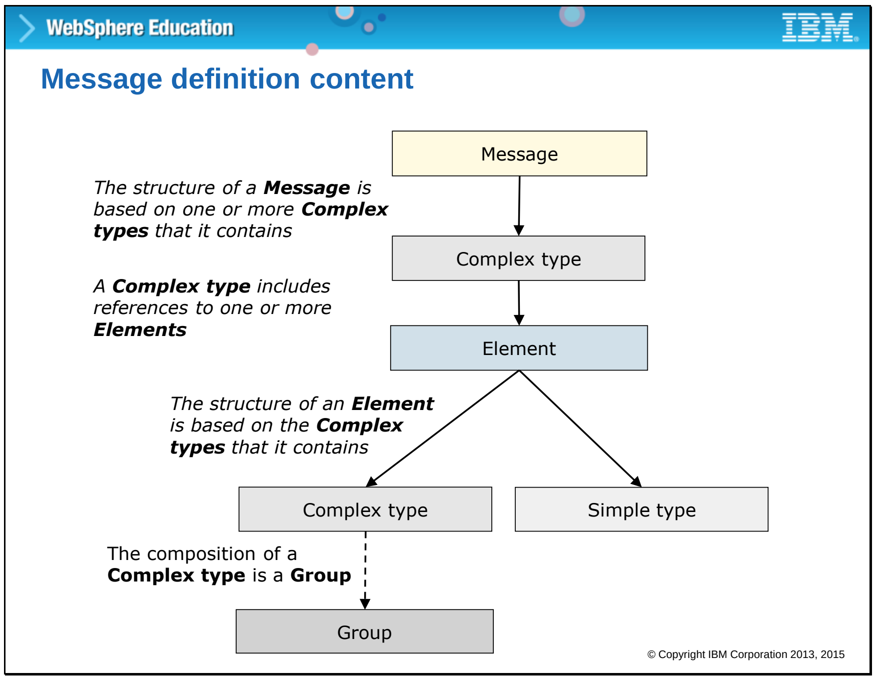
- The message definition file contains the logical structure and physical properties of the objects in XML schema form
- Create a message definition file before you create the message model objects
  - Create the message definition file from scratch
  - Base the new message definition file on an existing resource
  - Copy a message definition file from another message set

Creating a message definition file

After you create a message set, you typically import application message formats to create and populate message definition files. You can then edit the logical structure of your messages in the message definition editor.

In the message definition editor, you can create and edit binary, XML, and tagged-delimited physical formats that describe the precise appearance of your message bit stream during transmission. You can also create an empty message definition file and create your messages by using only the editor.

Slide 9



Message definition content

$$\begin{tikzcd}
 & \text{Message} \arrow[d] & \\
 & \text{Complex type} \arrow[d] & \\
 & \text{Element} \arrow[dl] \arrow[dr] & \\
\text{Complex type} \arrow[d, dashed] & & \text{Simple type} \\
\text{Group} & &
\end{tikzcd}$$

*The structure of a **Message** is based on one or more **Complex types** that it contains*

*A **Complex type** includes references to one or more **Elements***

*The structure of an **Element** is based on the **Complex types** that it contains*

The composition of a **Complex type** is a **Group**

Message definition content

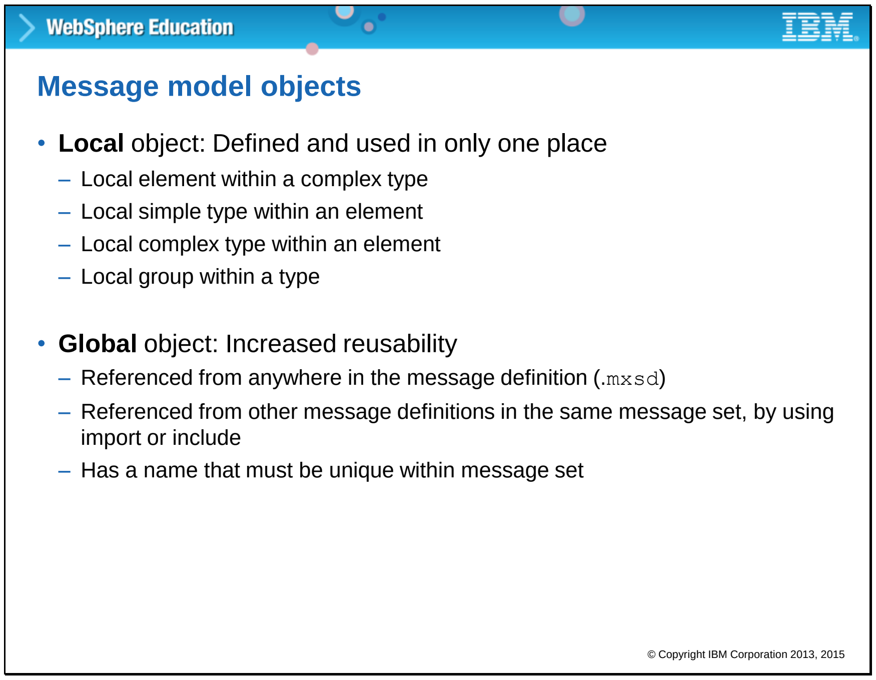
A message definition is a logical description of a message. It is a structured collection of elements. The structure begins with a message, which is a set of data that is passed between applications. Messages must have a structure and format that is agreed upon by the sending and receiving applications.

Similar to DFDL, the MRM message structure is based on a complex type. A complex type describes a structure within a message. It contains elements, attributes, and groups that are organized into a hierarchy.

Also, similar to DFDL, an MRM element can be simple or complex.

- A simple element describes one or more fields in a message. It is based on a simple type. It can repeat, and it can define a default or a fixed value.
- A complex element is a named structure that contains simple elements within the message. Complex elements can contain other complex elements, and they can also contain groups. A complex type defines the content of a complex element. Groups can be ordered (sequence), unordered (all), or selective (choice).

Slide 10



WebSphere Education

Message model objects

- **Local** object: Defined and used in only one place
  - Local element within a complex type
  - Local simple type within an element
  - Local complex type within an element
  - Local group within a type
- **Global** object: Increased reusability
  - Referenced from anywhere in the message definition (`.mxsd`)
  - Referenced from other message definitions in the same message set, by using import or include
  - Has a name that must be unique within message set

Message model objects

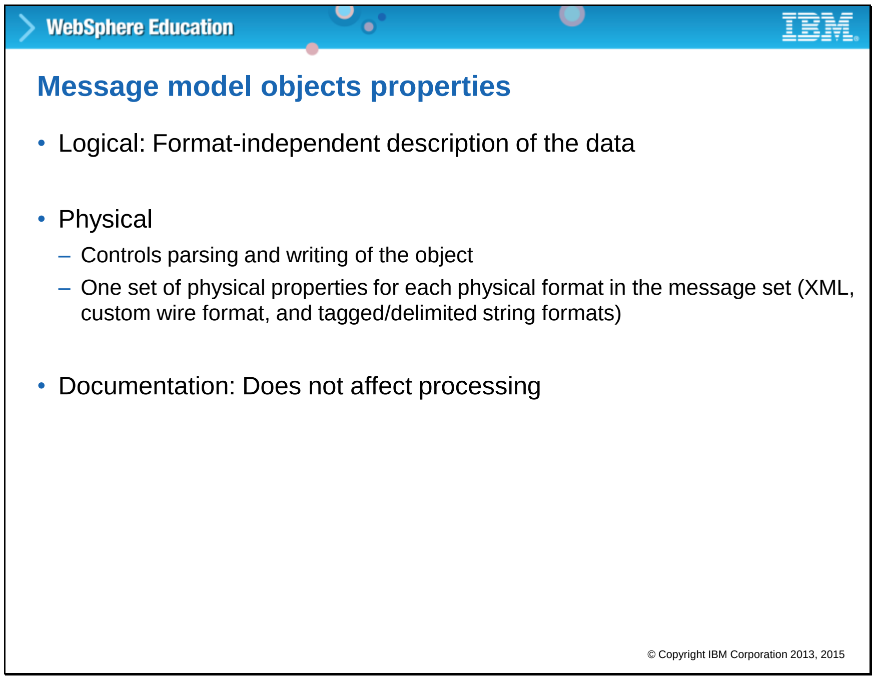
Many of the objects in the message model can be either global or local.

Local objects are defined and used in only one place in the message model.

A global object can be used in more than one place in a message model. A global object must have a unique name within the message set.

Make objects local unless they must be used in more than one place. A local object reduces the probability of name clashes among the global objects in the message model, and you find it easier to work with the message set.

Slide 11


WebSphere Education

Message model objects properties

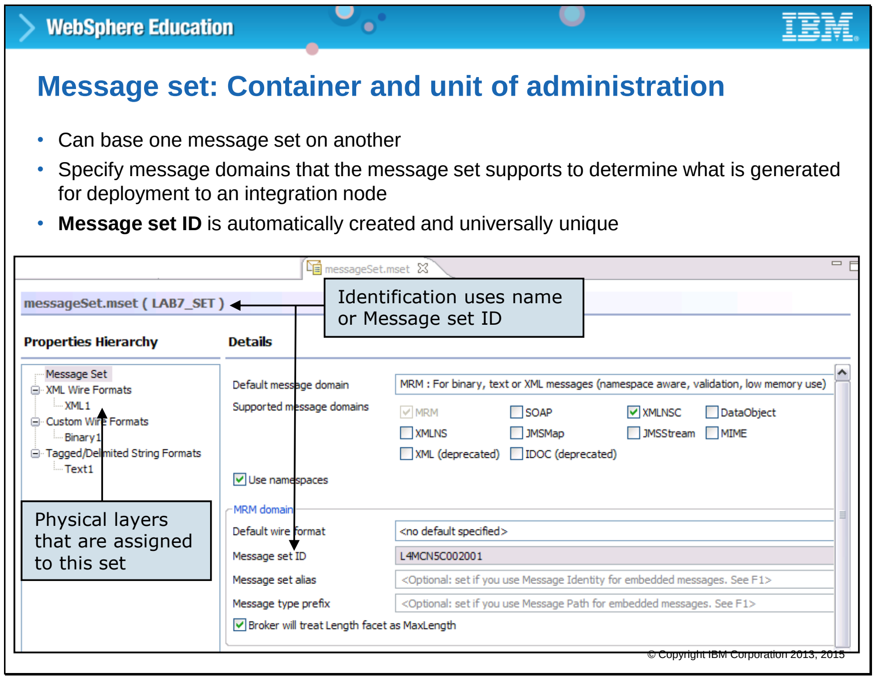
- Logical: Format-independent description of the data
- Physical
  - Controls parsing and writing of the object
  - One set of physical properties for each physical format in the message set (XML, custom wire format, and tagged/delimited string formats)
- Documentation: Does not affect processing

Message model objects properties

Similar to DFDL, the MRM model describes the logical structure of the data and the physical properties. The logical properties of the message describe what data the object contains without saying anything about how it is written down.

The physical properties of an object describe how the object is written down. MRM models can support up to three physical formats in one model. You define one set of physical properties for each physical format in your message set.

Slide 12


WebSphere Education

Message set: Container and unit of administration

- Can base one message set on another
- Specify message domains that the message set supports to determine what is generated for deployment to an integration node
- **Message set ID** is automatically created and universally unique

messageSet.mset

messageSet.mset (LAB7_SET)

Identification uses name or Message set ID

Properties Hierarchy

- Message Set
  - XML Wire Formats
    - XML1
  - Custom Wire Formats
    - Binary1
  - Tagged/Delimited String Formats
    - Text1

Physical layers that are assigned to this set

Details

Default message domain: MRM : For binary, text or XML messages (namespace aware, validation, low memory use)

Supported message domains: [✓ MRM] SOAP [✓ XMLNSC] DataObject XMLNS JMSMap JMSStream MIME XML (deprecated) IDOC (deprecated)

[✓ Use namespaces]

MRM domain

Default wire format: <no default specified>

Message set ID: L4MCN5C002001

Message set alias: <Optional: set if you use Message Identity for embedded messages. See F1>

Message type prefix: <Optional: set if you use Message Path for embedded messages. See F1>

[✓ Broker will treat Length facet as MaxLength]

Message set: Container and unit of administration

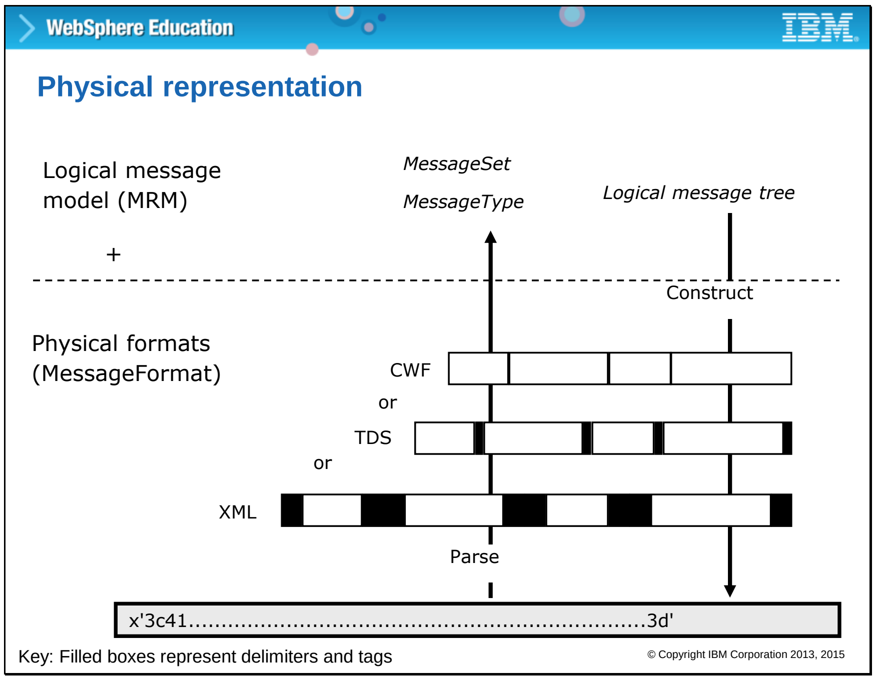
When you create the message set in the Toolkit, you must specify the message domains that the message set supports. The supported domains determine what is generated for deployment to an integration node, and are used when parsing and writing the messages that are defined within the message set.

Multiple message domains can be associated with a single message set. This feature is useful for messages in the MIME domain, where another domain parses the embedded part of the message. It is also useful in the SOAP domain, where the SOAP body will be parsed in the XMLNSC domain after the headers are parsed in the SOAP domain.

A message set has a name, provided by the folder, and a unique 13-character string that identifies the message set. The name or the identifier can be used interchangeably to specify the message set, such as in an MQRFH2 message header. While the name is more readable, the identifier is unique. The identifier is used in places where you do not have control over the names of the message sets that you are using. After you create a message set, the name or the identifier cannot be changed.

Each message set can support multiple physical layers or wire formats: XML, Custom Wire Formats, and Tagged/Delimited string.

Slide 13



Physical representation

WebSphere Education
IBM
Logical message model (MRM)
+
MessageSet
MessageType
Logical message tree
Construct
Physical formats (MessageFormat)
CWF
or
TDS
or
XML
Parse
x'3c41...3d'

Key: Filled boxes represent delimiters and tags

Physical representation

As mentioned previously, the MRM domain uses a language-independent and platform-independent model to reference message structures.

You do have a choice of three physical format layers:

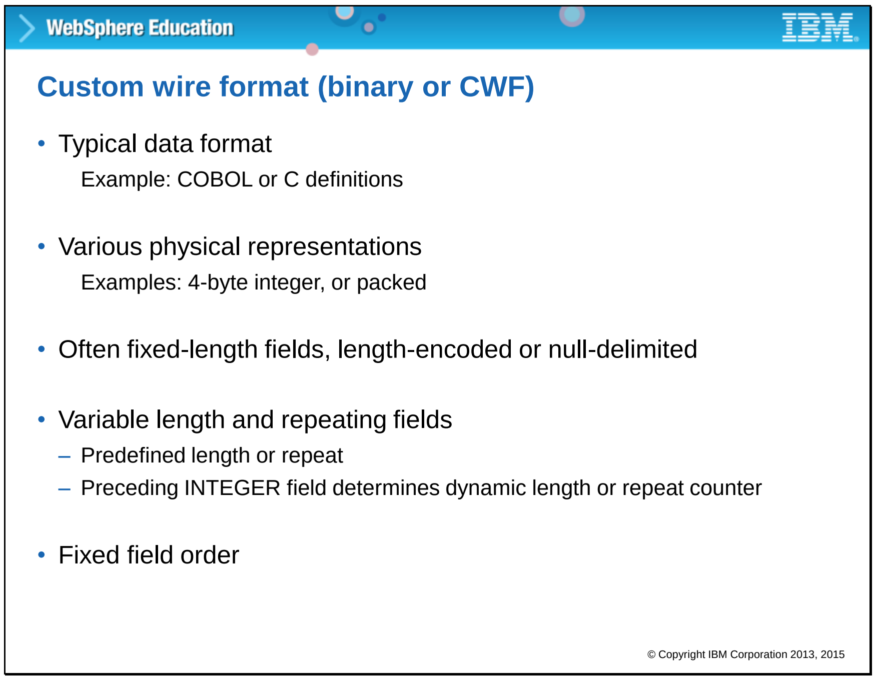
- XML
- Custom wire format (CWF), which is typically for messages that are composed of fixed-length fields.
- Tagged-delimited string (TDS) wire format

The physical layer is added to the message set and has a name, for example, CWF1. The integration node uses the physical layer on input to aid in parser identification, and on output to serialize the message.

At run time, at least one physical layer must be identified in a logical model, but multiple layers are allowed for the same model.

The next slides, take a closer look at the options for the physical format of the message.

Slide 14


WebSphere Education

Custom wire format (binary or CWF)

- Typical data format
  Example: COBOL or C definitions
- Various physical representations
  Examples: 4-byte integer, or packed
- Often fixed-length fields, length-encoded or null-delimited
- Variable length and repeating fields
  - Predefined length or repeat
  - Preceding INTEGER field determines dynamic length or repeat counter
- Fixed field order

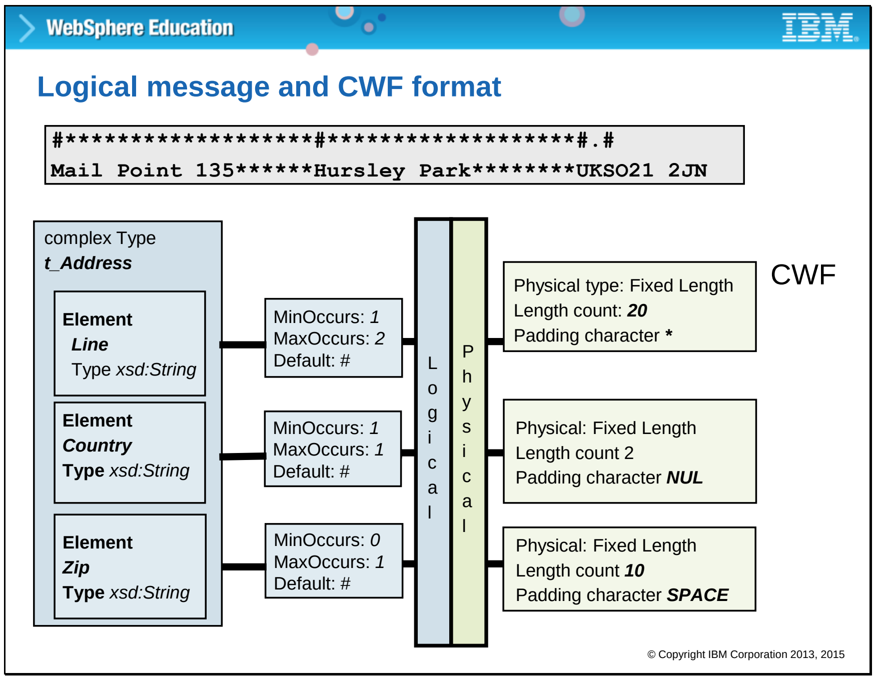
Custom wire format (binary or CWF)

Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

For custom wire format data, it is not possible to distinguish one element from the next without knowledge of the message structure. Examples of custom wire format data include fixed-format text data, such as COBOL, and binary data, such as C definitions.

To correctly determine the values of individual elements in the data stream, the following information must be made available to the message parser:
- The order of the elements defined in the logical properties
- The length of the elements specified in bytes, characters, or character units
- The number of occurrences of each element
- The type of data that is contained in each element.
- Characteristics that are based on the logical type of the data.

While the other physical representations such as XML support self-defining elements, the parsing of a CWF message does not. Any such self-defining elements are discarded during the output of messages in a CWF message.



Logical message and CWF format

```
#******************#*******************#.#
Mail Point 135******Hursley Park********UKSO21 2JN
```

Logical message and CWF format

How does the logical message relate to the CWF physical layer?

Just like DFDL, the logical MRM model defines the structure of the data. As part of the logical model, you define the number of occurrences of an element: If an element is mandatory, optional, or repeating.

If you are using a CWF physical layer, all elements must be mandatory and in fixed order on input. You must then define the physical CWF properties of the element within a complex type.

In this example, the box at the top of the slide shows the sample message. The sample contains three fields:

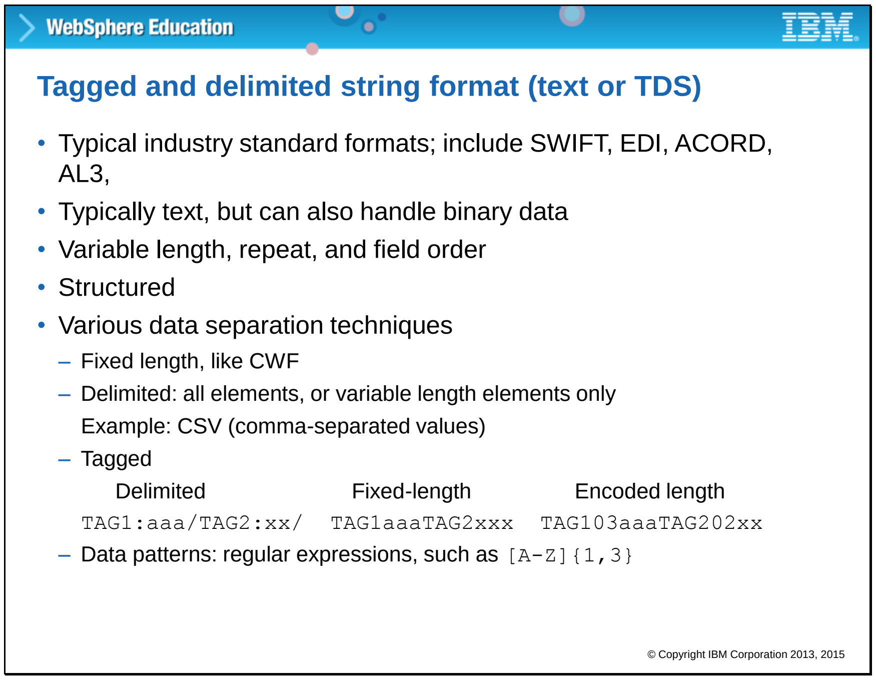
- Address Line, which can occur 1 or 2 times
- Country code
- Postal code

The hashtag character is the default character. The pad character is different for each field.

The physical properties for each element include a physical type and its length, pad characters, and default values.

Be sure to verify the CWF physical properties for each element. The default values are not sufficient and cause warnings or errors in the Task List, thus preventing deployment.

Slide 16



WebSphere Education

Tagged and delimited string format (text or TDS)

- Typical industry standard formats; include SWIFT, EDI, ACORD, AL3,
- Typically text, but can also handle binary data
- Variable length, repeat, and field order
- Structured
- Various data separation techniques
  - Fixed length, like CWF
  - Delimited: all elements, or variable length elements only
    Example: CSV (comma-separated values)
  - Tagged

Delimited	Fixed-length	Encoded length
`TAG1:aaa/TAG2:xx/`	`TAG1aaaTAG2xxx`	`TAG103aaaTAG202xx`

  - Data patterns: regular expressions, such as `[A-Z]{1,3}`

Tagged and delimited string format (text or TDS)

You can use the Tagged Delimited String (TDS) physical format to model formatted text messages, perhaps with field content identified by tags or separated by specific delimiters or both.

With TDS data, the fields in the message can have a tag or a label that precedes the data value. The tag is a string that uniquely identifies the data value. The tagged-delimited string format associates a tag with each element when you define the element.

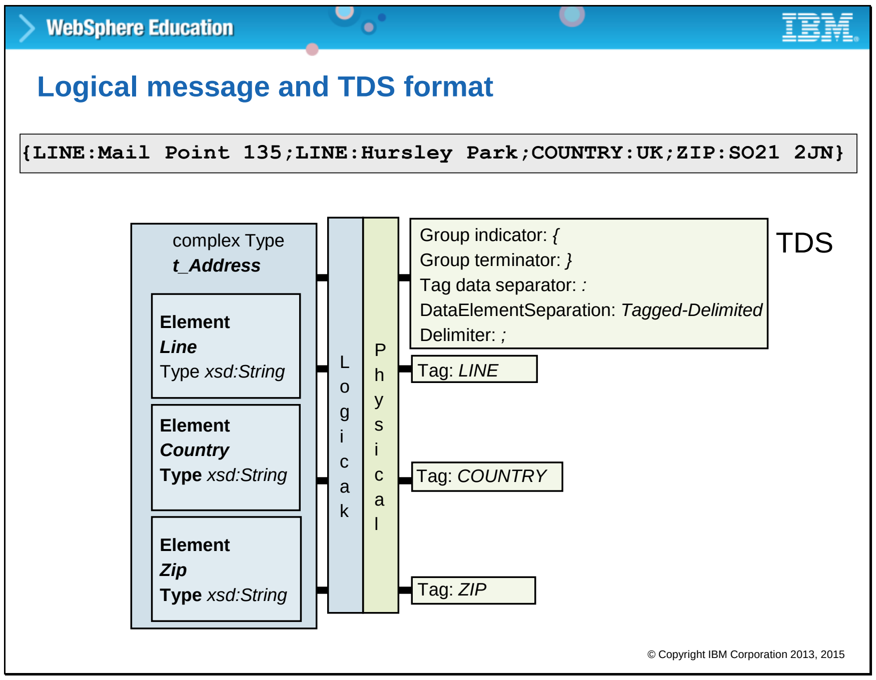
The message can contain various special characters or strings in addition to the tags and text string data values. The tagged-delimited string format supports a number of different types of special characters or strings. Some messages have a special character or string that separates each data value from the next. In the tagged-delimited string format, this character or string is a known as a delimiter.

In formats that have a tag before each data value, the tag can be separated from its data value by a special character or string. In the tagged-delimited string format, this character or string is known as a tag data separator. Some tags can be defined as fixed length, so a tag data separator is not necessary.

The tagged-delimited string property that controls the way elements are separated is **Data Element Separation** property. It has several options that provide you with the option to choose for example, if tags are used, if field lengths are fixed or variable, and what types of fields are allowed.

You can choose the **Use data pattern** option for the Data Element Separation to specify regular expressions to identify parts of the message data.

Slide 17



Logical message and TDS format

```
{LINE:Mail Point 135;LINE:Hursley Park;COUNTRY:UK;ZIP:SO21 2JN}
```

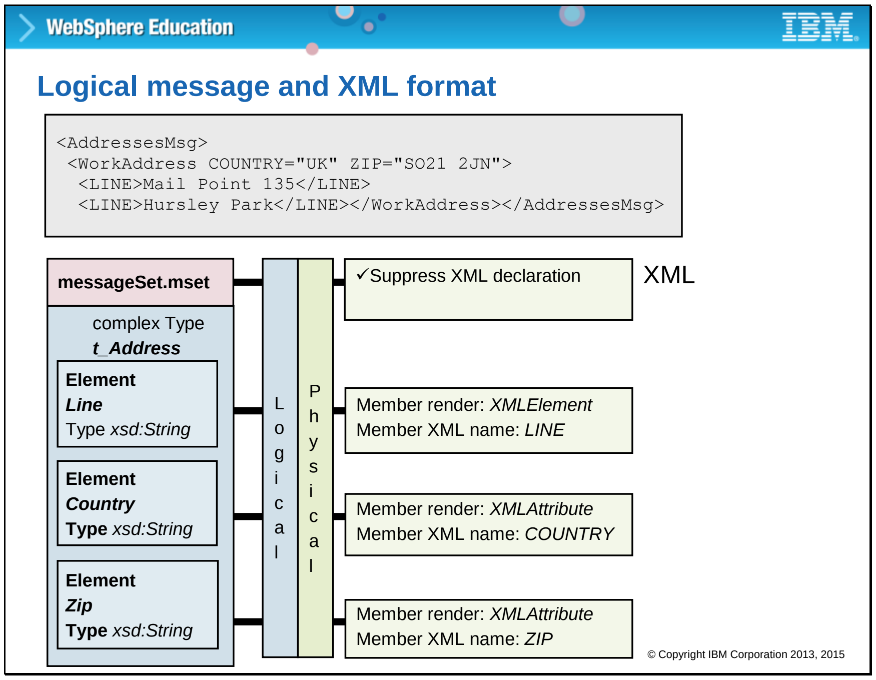
Logical message and TDS format

The slide shows an example of tagged-delimited data and the logical and physical properties that describe it. Notice that the logical message is the same as in the previous example. However, a physical layer exists for tagged-delimited strings.

For each element within the complex type, you must specify how to find it, which means its tag value. Element tags are only needed for a data element separation of tagged. If you had variable delimited elements, for example, the tag property would not be editable. In the example, each element has a tag value. In each element, the colon character separates the tag from the data.

If applicable to your data, you must also identify the delimiter that separates each element. In this example, the delimiter between each element is a semi-colon character.

Slide 18



Logical message and XML format

```
<AddressesMsg>
 <WorkAddress COUNTRY="UK" ZIP="SO21 2JN">
  <LINE>Mail Point 135</LINE>
  <LINE>Hursley Park</LINE></WorkAddress></AddressesMsg>
```

messageSet.mset
complex Type *t_Address*
Element ***Line*** Type *xsd:String*
Element ***Country*** **Type** *xsd:String*
Element ***Zip*** **Type** *xsd:String*
Logical
Physical
✓Suppress XML declaration
XML
Member render: *XMLElement* Member XML name: *LINE*
Member render: *XMLAttribute* Member XML name: *COUNTRY*
Member render: *XMLAttribute* Member XML name: *ZIP*

Logical message and XML format

The final physical layer that you can define in a message definition is XML. You can use the XML physical format to model XML messages, including messages that use XML namespaces. This support includes the ability to create a message model directly from an XML DTD or XML schema file.

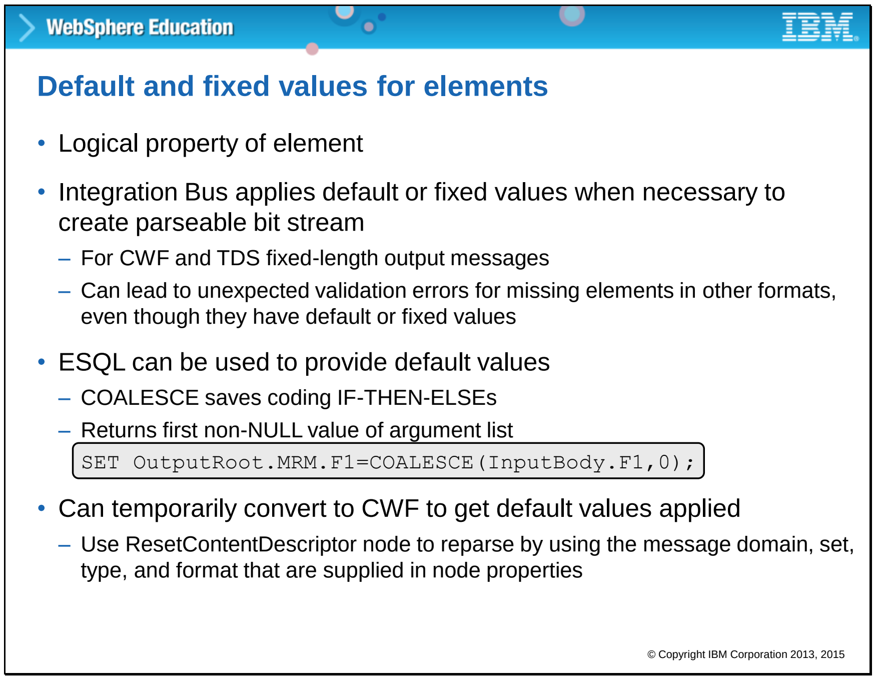
Using MRM XML wire format gives you extra control over the rendering of your data. For example, you might have a data field that is rendered as an XML element in one message, and as an XML attribute in another message. Or you can have a data field that is known by a particular name in one message, and a different name in another message.

XML messages are, by their nature, self-describing: a tag name or an attribute name prefixes each piece of data. So, it is possible for an XML message instance to contain elements that are not in the MRM definition for that message. If such an element exists in the message set, the MRM objects for that element are used in parsing or writing the message. If the element does not exist in the message set, it is treated as a self-defining element and its data type is set to STRING.

This slide shows an example of an XML message, the logical structure, and the XML physical properties.

You should recognize that the advantage of using MRM is the ability to define three physical data types in one model.

Slide 19


WebSphere Education

Default and fixed values for elements

- Logical property of element
- Integration Bus applies default or fixed values when necessary to create parseable bit stream
  - For CWF and TDS fixed-length output messages
  - Can lead to unexpected validation errors for missing elements in other formats, even though they have default or fixed values
- ESQL can be used to provide default values
  - COALESCE saves coding IF-THEN-ELSEs
  - Returns first non-NULL value of argument list

```
SET OutputRoot.MRM.F1=COALESCE(InputBody.F1,0);
```

- Can temporarily convert to CWF to get default values applied
  - Use ResetContentDescriptor node to reparse by using the message domain, set, type, and format that are supplied in node properties

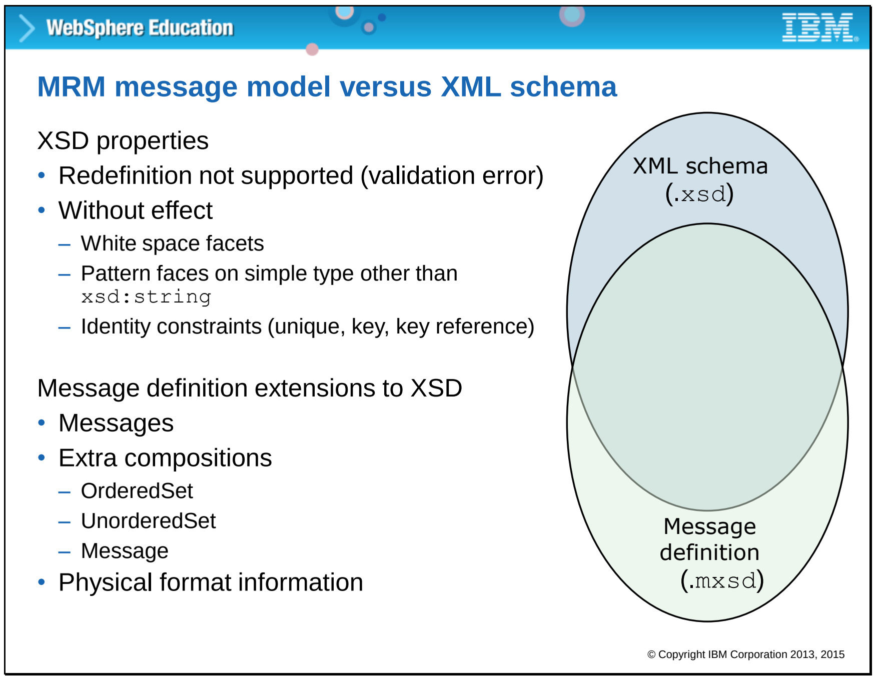
Default and fixed values for elements

You can specify default values when you define the logical and physical models. Those default values are applied only under certain conditions. For example, if you defined a field to have a default value in a message set, but during processing that field is not parsed, the default value is not applied.

You can also provide default values programmatically in a Compute-type node or by using the ResetContentDescriptor node, which forces parsing of the message with a different parser.

The ResetContentDescriptor node can be useful if you want to dynamically reparse a message at run time. For example, some generic error handler routines reparse a message into BLOB format so that it can be saved to a file, database, or queue without regard to the data types in the message body.

Slide 20



WebSphere Education

MRM message model versus XML schema

XSD properties

- Redefinition not supported (validation error)
- Without effect
  - White space facets
  - Pattern faces on simple type other than `xsd:string`
  - Identity constraints (unique, key, key reference)

Message definition extensions to XSD

- Messages
- Extra compositions
  - OrderedSet
  - UnorderedSet
  - Message
- Physical format information

MRM message model versus XML schema

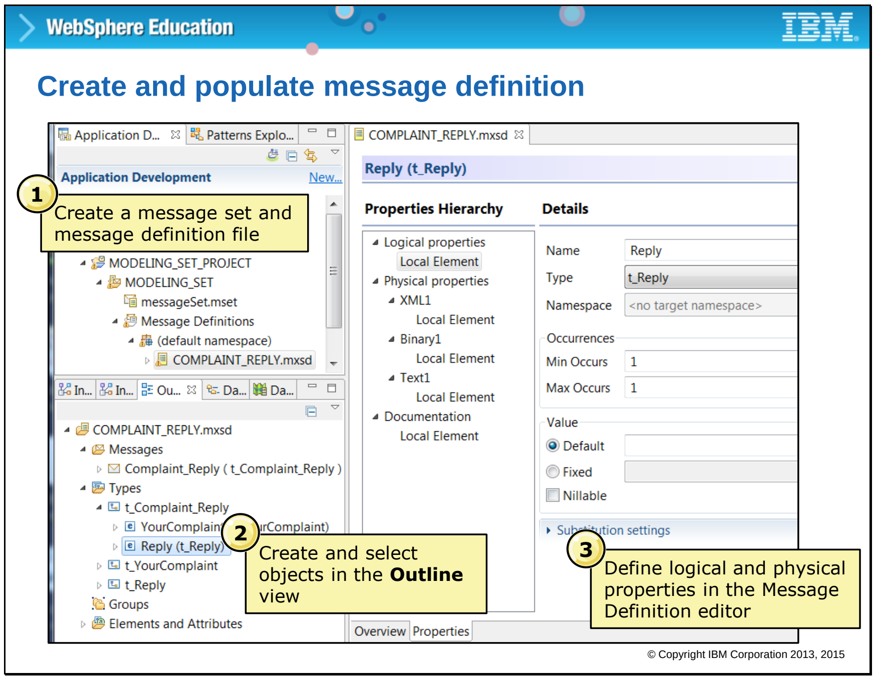
The Integration Bus MRM logical message model is *an* XML schema, with a few extensions and restrictions.

For example, XML schema constructs **key**, **keyref**, and **unique** enforce uniqueness constraints on the value of an element or attribute in an instance document within a certain scope. They use XPath expressions to select a set of elements and attributes, which are the target of uniqueness constraints. They do not participate in building the structural contents of the message.

The MRM message model defines the structure and physical definitions for a message. It includes extra compositions such as Ordered Set and Unordered Set.

The next series of slides show you how to use the Integration Toolkit to create a message set and message definition file.

Slide 21



Create and populate message definition

Name: Reply
Type: t_Reply
Namespace: <no target namespace>
Occurrences
Min Occurs: 1
Max Occurs: 1
Value
[✓ Default]
Fixed
Nillable

1 Create a message set and message definition file

2 Create and select objects in the **Outline** view

3 Define logical and physical properties in the Message Definition editor

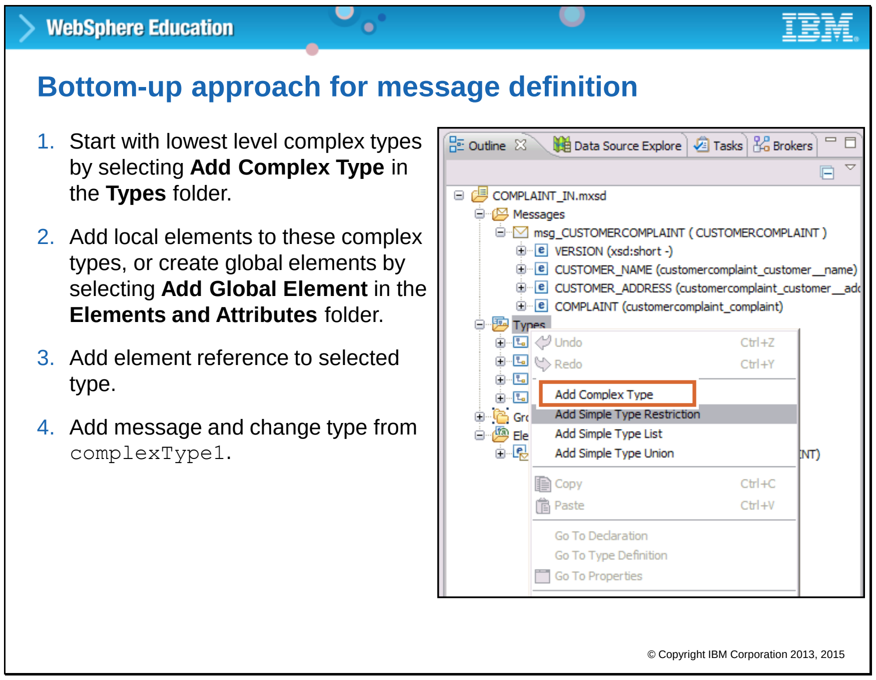
Create and populate message definition

This slide shows the main steps for creating and populating the message definition:

1. Create the message set project, message set, and a message definition file.
2. Create the message definitions by using the Outline view to create and select elements in the message definition.
3. Define the logical properties and physical properties for required physical formats.

This slide assumes that you are creating the message definition by starting with an empty file. You can also populate your message set with message definitions by importing existing application message formats.

Slide 22



WebSphere Education

Bottom-up approach for message definition

1. Start with lowest level complex types by selecting **Add Complex Type** in the **Types** folder.
2. Add local elements to these complex types, or create global elements by selecting **Add Global Element** in the **Elements and Attributes** folder.
3. Add element reference to selected type.
4. Add message and change type from `complexType1`.

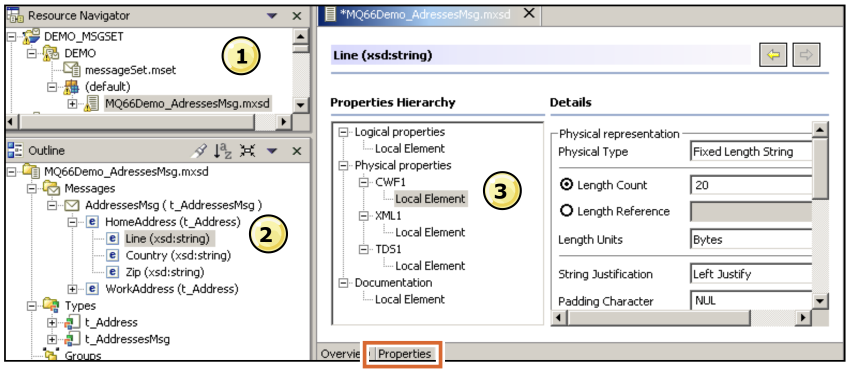
Bottom-up approach for message definition

This slide shows the major steps for creating a message definition by using a bottom-up approach that starts with defining the lowest complex type and working up to the highest complex type.

1. In the **Outline** view of Integration Toolkit, start with lowest level complex type by selecting **Add Complex Type** in the **Types** folder.
2. Add the local simple elements to the complex type, or create global elements for reuse by selecting **Add Global Element** in the **Elements and Attributes** folder.
3. Add an element reference to the selected type.
4. Add a message and change its type from complexType1.

WebSphere Education

Setting physical properties



DEMO_MSGSET
DEMO
messageSet.mset
(default)
MQ66Demo_AdressesMsg.mxsd

MQ66Demo_AdressesMsg.mxsd
Messages
AddressesMsg (t_AddressesMsg)
HomeAddress (t_Address)
Line (xsd:string)
Country (xsd:string)
Zip (xsd:string)
WorkAddress (t_Address)
Types
t_Address
t_AddressesMsg
Groups

*MQ66Demo_AdressesMsg.mxsd

Line (xsd:string)

Properties Hierarchy
- Logical properties
  - Local Element
- Physical properties
  - CWF1
    - Local Element
  - XML1
    - Local Element
  - TDS1
    - Local Element
- Documentation
  - Local Element

Details

Physical representation

Physical Type	Fixed Length String
● Length Count	20
○ Length Reference	
Length Units	Bytes
String Justification	Left Justify
Padding Character	NUL

1. Select `*.mxsd` file in the **Application Development** view.
2. In the **Outline** view, select the object to customize its element properties.
3. In the editor view, define physical properties for one or more physical format.

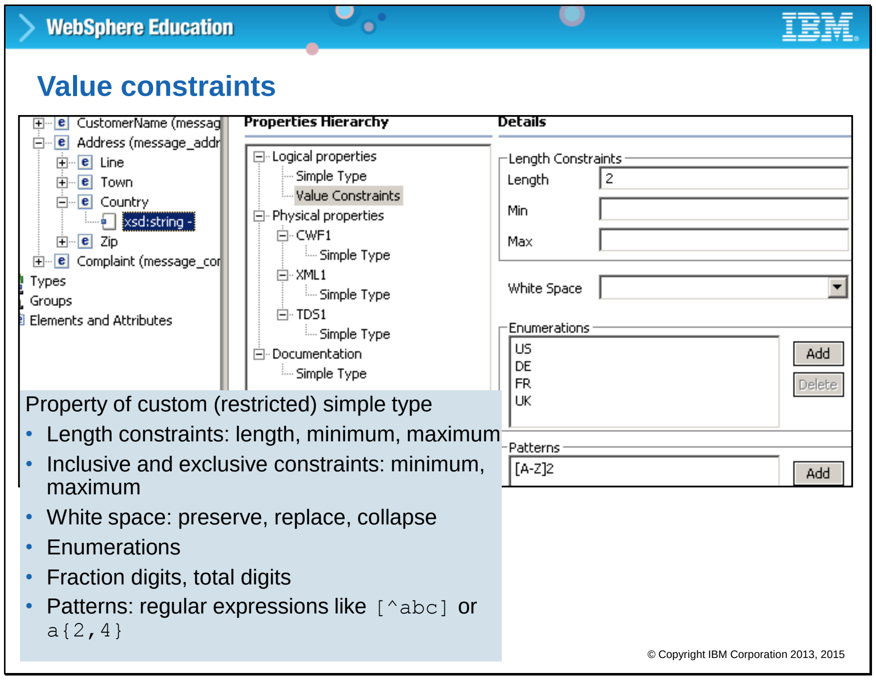
Setting physical properties

After you define the structure of the data, the next step is to define the properties.

This slide shows the steps for setting the physical properties of a CWF format message on the **Properties** tab:

1. Select the message set definition file (.mxsd) in the **Application Development** view.
2. In the **Outline** view, select the object in the message set definition file to customize its properties.
3. In the editor view, click **CWF1 > Local Element** from the **Properties** tab and configure the physical properties.

Slide 24



Value constraints

CustomerName (messag
Address (message_addr
Line
Town
Country
xsd:string -
Zip
Complaint (message_cor
Types
Groups
Elements and Attributes

Length Constraints
Length: 2
Min:
Max:
White Space:
Enumerations: US, DE, FR, UK
Patterns: [A-Z]2

Property of custom (restricted) simple type

- Length constraints: length, minimum, maximum
- Inclusive and exclusive constraints: minimum, maximum
- White space: preserve, replace, collapse
- Enumerations
- Fraction digits, total digits
- Patterns: regular expressions like `[^abc]` or `a{2,4}`

Value constraints

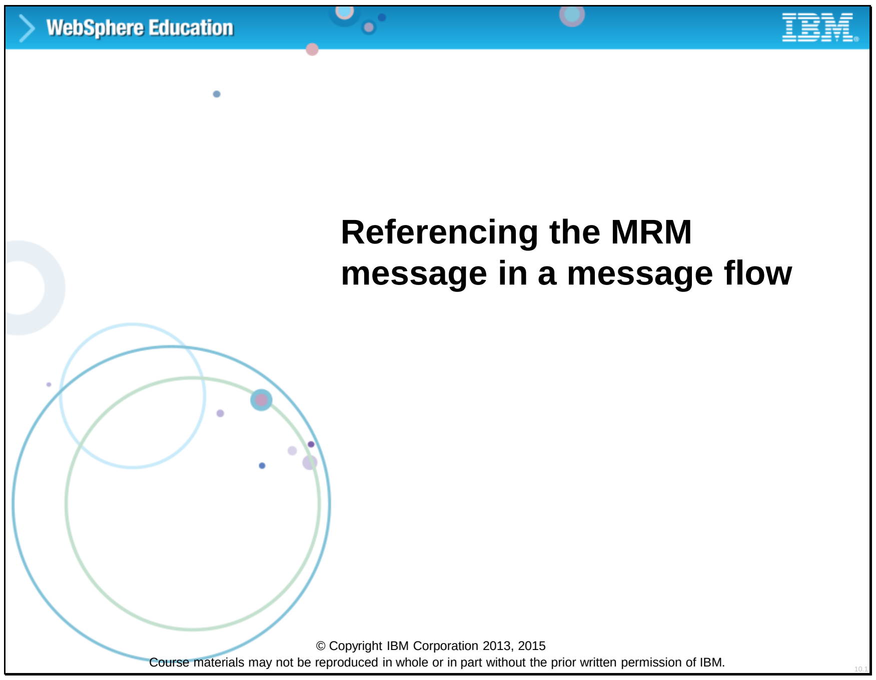
Physical property value constraints are normally associated with a custom simple type; they refine a simple type by defining limits on the values that it can represent. For example, if a payroll record contained a field for regular hours, you might constrain the value to be 1 - 40. Or, if an address record contained country codes, you might constrain the value of that field to only those two-character codes that you provide in a list.

The properties that are shown on the object page and the values that those properties can take can vary according to the type of the object. For example, the properties **Fraction Digits** and **Total Digits** are only available for decimal simple types.

You can use a pattern to define a string that is used as a regular expression that the data in the associated type must match. The regular expression syntax that is supported is a subset of XML schema regular expressions.

To configure value constraints, click the simple type that you are updating in the **Outline** view and click the **Properties** tab. In the **Properties** hierarchy under **Logical Properties**, click **Value Constraints**. The current value constraints settings for the selected simple type are shown in the **Details** pane.

Slide 25

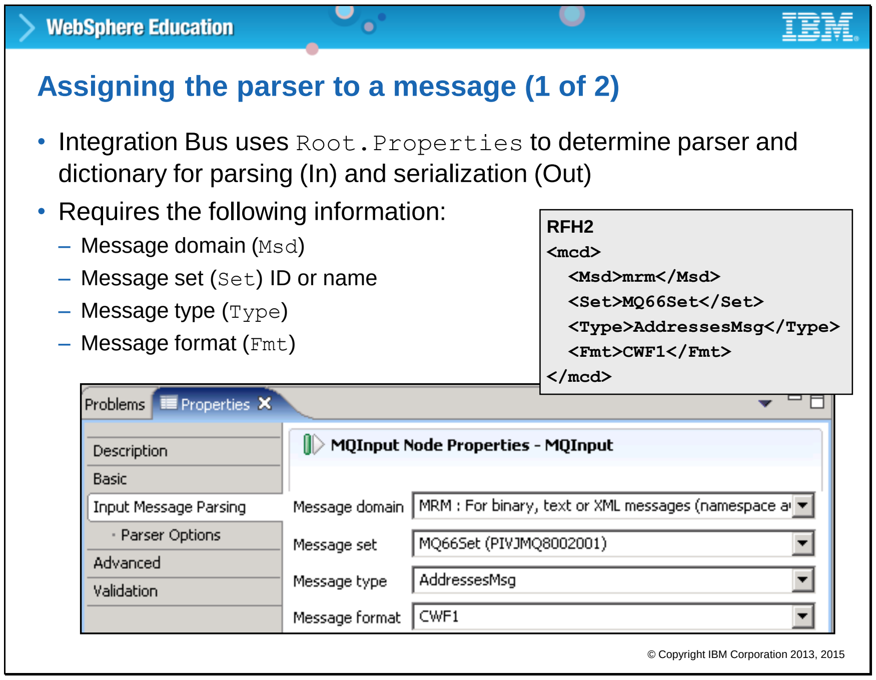


WebSphere Education

Referencing the MRM message in a message flow

Topic 2: Referencing the MRM message in a message flow

In this topic, you learn how to reference an MRM model in a message flow.



WebSphere Education

Assigning the parser to a message (1 of 2)

- Integration Bus uses `Root.Properties` to determine parser and dictionary for parsing (In) and serialization (Out)
- Requires the following information:
  - Message domain (`Msd`)
  - Message set (`Set`) ID or name
  - Message type (`Type`)
  - Message format (`Fmt`)

RFH2

```
<mcd>
  <Msd>mrm</Msd>
  <Set>MQ66Set</Set>
  <Type>AddressesMsg</Type>
  <Fmt>CWF1</Fmt>
</mcd>
```

MQInput Node Properties - MQInput

Message domain: MRM : For binary, text or XML messages (namespace a
Message set: MQ66Set (PIVJMQ8002001)
Message type: AddressesMsg
Message format: CWF1

Assigning the parser to a message (1 of 2)

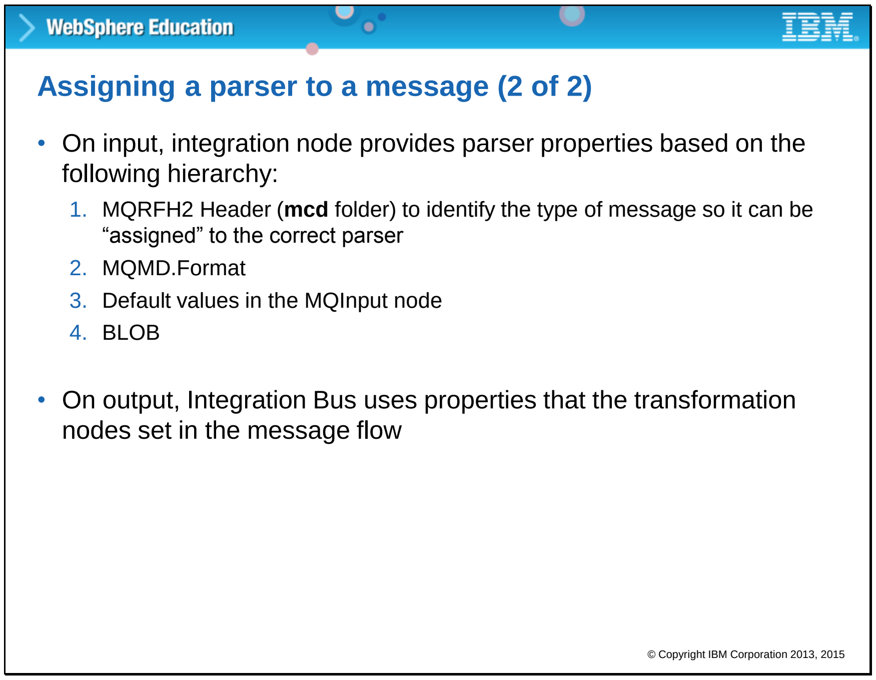
When you save the message set, it is validated to ensure that it is complete. It is not validated against the data, you must validate the data against the data in a message flow.

The process for identifying the model in the message flow is similar to the process that you use when referencing a DFDL model. First, ensure that the message flow application has a reference to the message set project. Then, on the message flow processing node, select the MRM domain and then identify the message set, message type, and message format.

Optionally, the application can provide this same information in the message header, as shown in this example.

To deploy a message set, add the message set to the BAR file with the message flow application. If you attempt to deploy the message set only, the Integration Toolkit suggests that the entire application must be deployed.

Slide 27


WebSphere Education

Assigning a parser to a message (2 of 2)

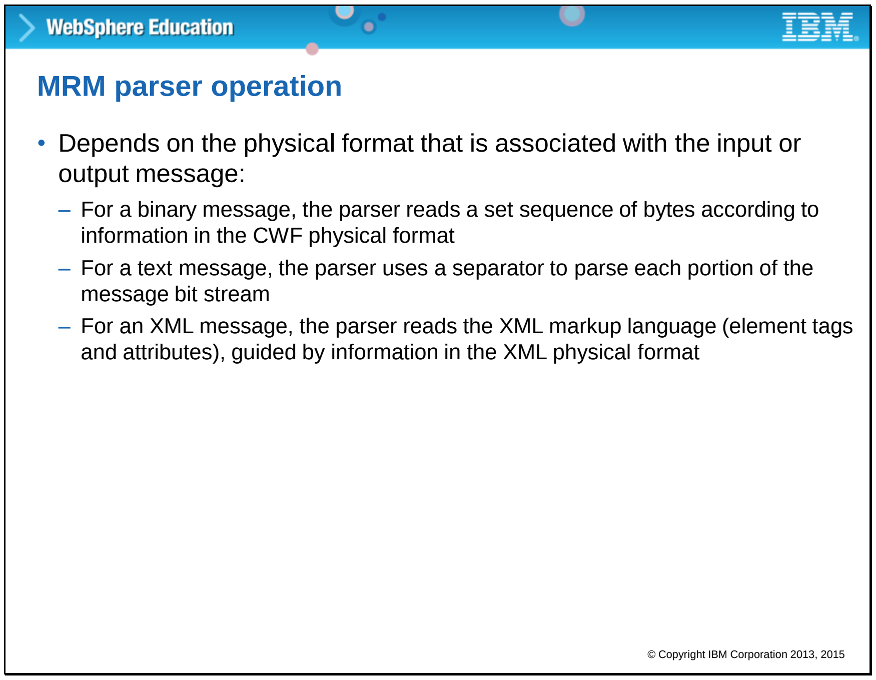
- On input, integration node provides parser properties based on the following hierarchy:
  1. MQRFH2 Header (**mcd** folder) to identify the type of message so it can be “assigned” to the correct parser
  2. MQMD.Format
  3. Default values in the MQInput node
  4. BLOB
- On output, Integration Bus uses properties that the transformation nodes set in the message flow

Assigning a parser to a message (2 of 2)

On input, integration node determines the parser properties based on the following hierarchy:

1. IF the MQRFH2 header exists, the integration node checks the header in the message to identify the type of message so it can be “assigned” to the correct parser.
2. If the MQRFH2 header does not exist or the message properties are not set, the integration node checks the **Format** portion of the MQMD.
3. If the message’s MQMD **Format** fields do not exist or are blank, the integration node then checks the **Message Parsing Properties** specified in the message flow Input node.
4. If no parser and model information can be found, then the default is to treat the message as a bitstream by using the BLOB domain.

On output, Integration Bus determines the domain based on properties that the transformation nodes set in the message flow.



WebSphere Education

MRM parser operation

- Depends on the physical format that is associated with the input or output message:
  - For a binary message, the parser reads a set sequence of bytes according to information in the CWF physical format
  - For a text message, the parser uses a separator to parse each portion of the message bit stream
  - For an XML message, the parser reads the XML markup language (element tags and attributes), guided by information in the XML physical format

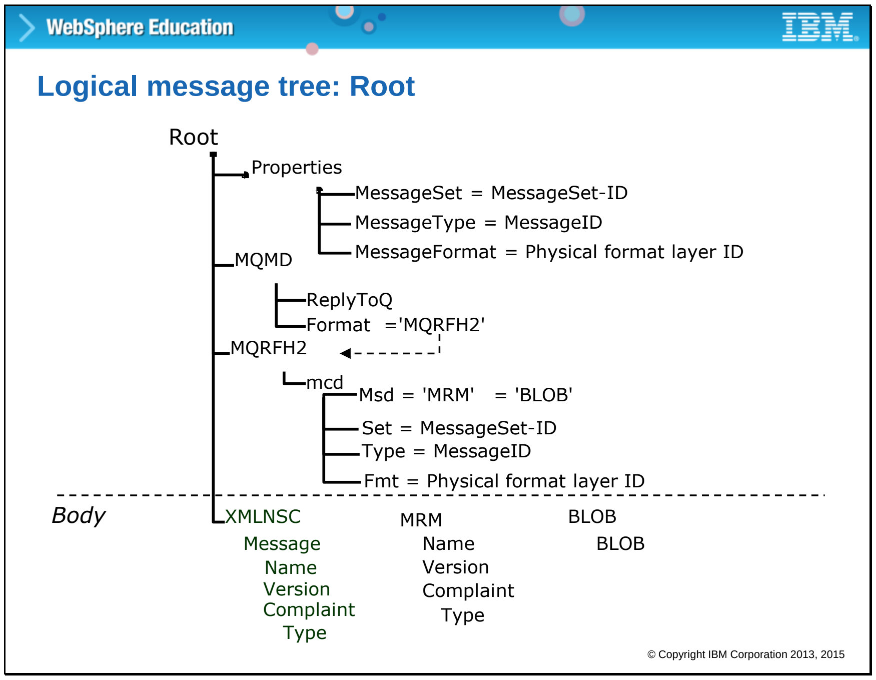
MRM parser operation

The operation of the MRM parser depends on the physical format that is associated with the input or output message.

For a binary message, the parser reads a set sequence of bytes according to information in the CWF physical format, and converts them into the fields and values in the message tree.

For a text message, the parser uses the TDS physical format **Data Element Separation** property to decide how to parse each portion of the message bit stream. This format informs the parser whether the message uses delimiters, tags, fixed-length elements, patterns, and so on. The parser then reads the data according to information in the TDS physical format, and converts it into the fields and values in the message tree.

For an XML message, the parser reads the XML element tags and attributes, which are guided by information in the XML physical format, and converts them into the fields and values in the message tree.



Logical message tree: Root

Logical message tree: Root

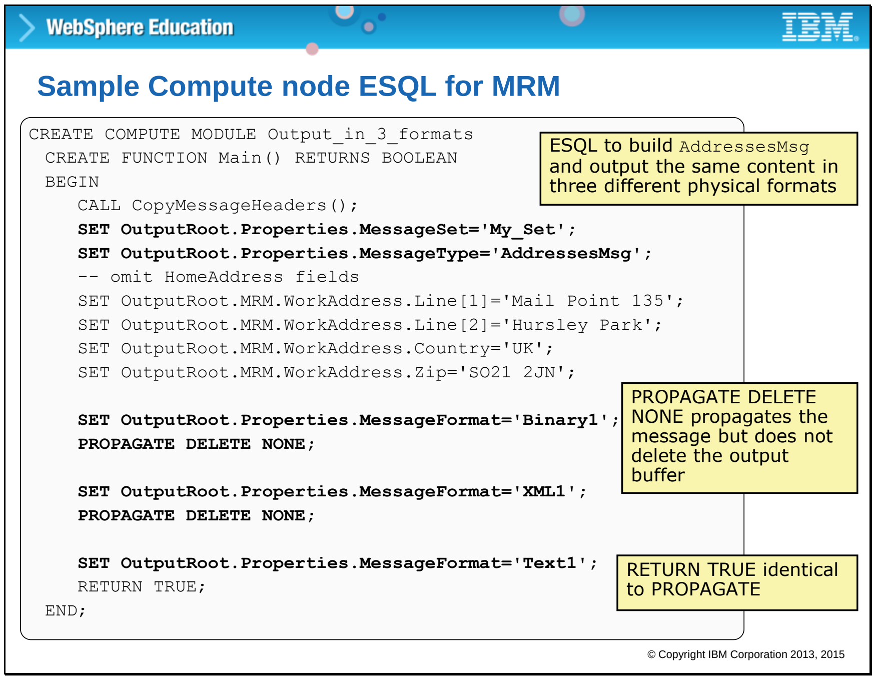
This slide shows an example of a logical message tree.

The first branch, **Properties**, is only present while the message is processed within the integration node. It contains information for predefined messages so that the parser knows where to get the formats. The domain is contained in the last branch of Root; the section where the application data is located.

If the message came from MQ, it contains the MQMD and possibly the MQRFH2. The MQMD can contain the message-format information, or as shown here, point to the message-format information in the MQRFH2.

You can create or modify the logical message tree by using ESQL and Compute node, which is described next.

Slide 30


WebSphere Education

Sample Compute node ESQL for MRM

```
CREATE COMPUTE MODULE Output_in_3_formats
  CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
      CALL CopyMessageHeaders();
      SET OutputRoot.Properties.MessageSet='My_Set';
      SET OutputRoot.Properties.MessageType='AddressesMsg';
      -- omit HomeAddress fields
      SET OutputRoot.MRM.WorkAddress.Line[1]='Mail Point 135';
      SET OutputRoot.MRM.WorkAddress.Line[2]='Hursley Park';
      SET OutputRoot.MRM.WorkAddress.Country='UK';
      SET OutputRoot.MRM.WorkAddress.Zip='SO21 2JN';

      SET OutputRoot.Properties.MessageFormat='Binary1';
      PROPAGATE DELETE NONE;

      SET OutputRoot.Properties.MessageFormat='XML1';
      PROPAGATE DELETE NONE;

      SET OutputRoot.Properties.MessageFormat='Text1';
      RETURN TRUE;
  END;
```

ESQL to build `AddressesMsg` and output the same content in three different physical formats

PROPAGATE DELETE NONE propagates the message but does not delete the output buffer

RETURN TRUE identical to PROPAGATE

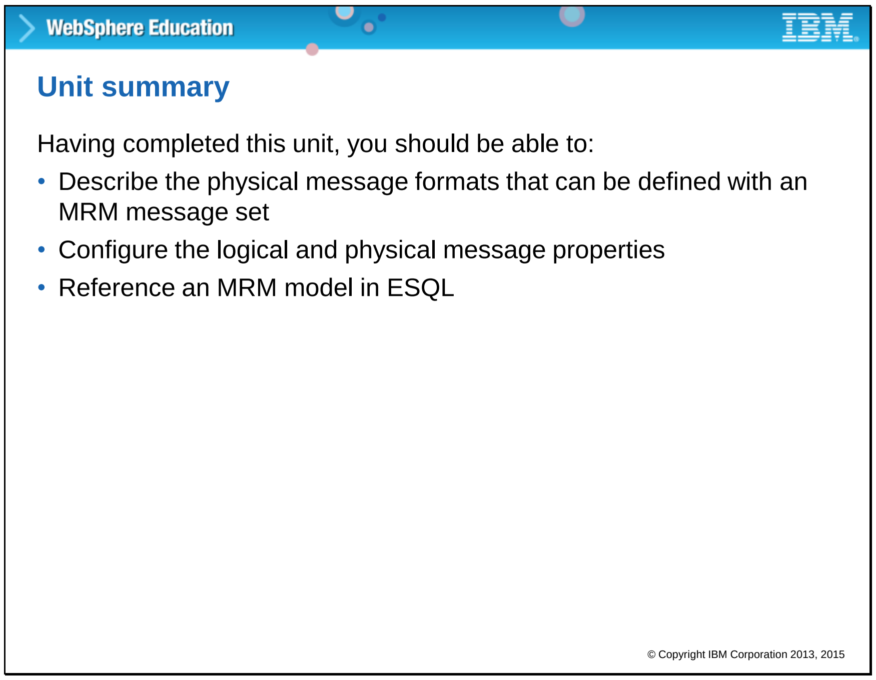
Sample Compute node ESQL for MRM

The slide shows sample ESQL code in a Compute node module that generates the AddressesMsg sample output in three different physical formats.

In addition to setting the Properties in the output logical message tree, this module uses the PROPAGATE statement to create multiple different output messages from a single Compute node.

You can also use JavaCompute node and the Java propagate method to complete these actions.

Slide 31


WebSphere Education

Unit summary

Having completed this unit, you should be able to:
- Describe the physical message formats that can be defined with an MRM message set
- Configure the logical and physical message properties
- Reference an MRM model in ESQL

Unit summary

A message set is the original container for message models that are used by WebSphere Message Broker. In IBM Integration Bus, DFDL schema files that are contained in applications and libraries are the preferred way to model messages for most data formats. Message sets continue to be supported, and if you use the MRM or IDOC domains are required. In this unit, you learned how to create a message set to define a message and use the MRM domain.

Having completed this unit, you should be able to:
- Describe the physical message formats that can be defined with an MRM message set
- Configure the logical and physical message properties
- Reference an MRM model in ESQL