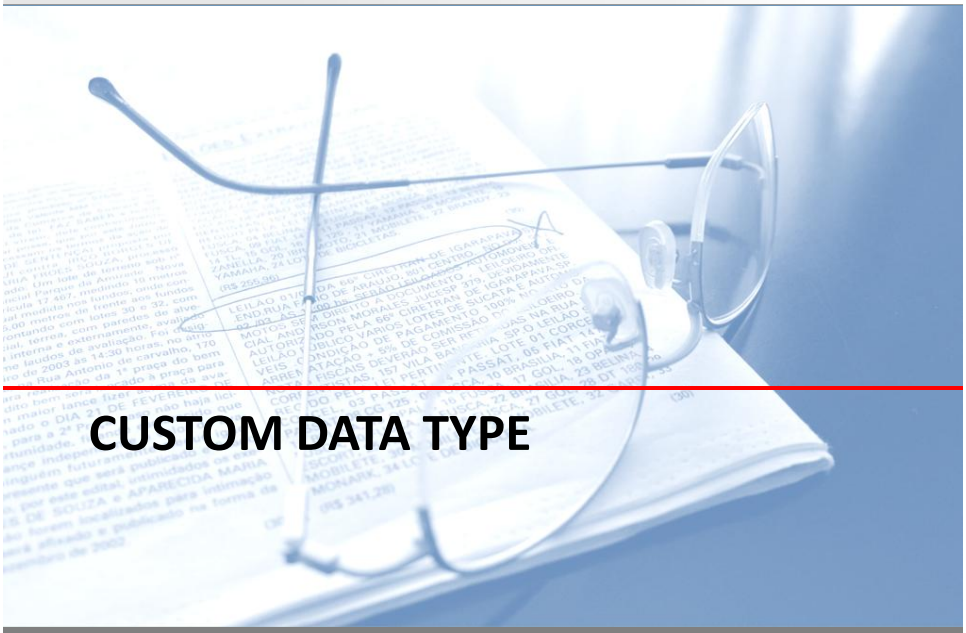
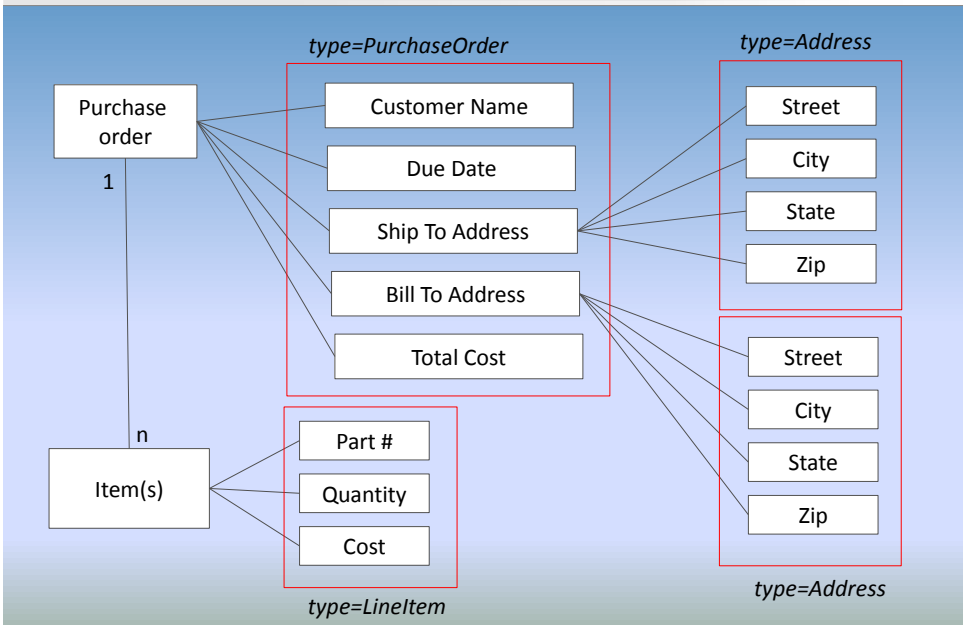


CUSTOM DATA TYPE

A typical purchase order



Typical Java Implementation

FUJITSU

Data types:

```
class Address {
   public String street;
   public String city;
   public String state;
   public String zip;
}

class LineItem {
   public String partNo;
   public int    quantity;
   public float  price;
}

class PurchaseOrder {
   public String  customerName;
   public Date    dueDate;
   public Address shipTo;
   public Address billTo;
   public float   totalCost;
   List<LineItem> items;
}
```

instances or uses:

```
public void simplyConstruct(String[] args)
{
   String myStreet = “315 Elm Street”;

   PurchaseOrder myPurchase = new PurchaseOrder();
   Address myAddress = new Address();
   myAddress.street = myStreet;
   myAddress.city = "Oakhurst";
   myAddress.state = "AZ";
   myAddress.zip = "87654";

   myPurchase.customerName = "Helmut Baer";
   myPurchase.due = getDate(2010, 12, 12);
   myPurchase.shipTo = myAddress;
   myPurchase.billTo = myAddress;

   items = new List<LineItems>();
   items.add(new LineItem("192D334", 15, 100.00);
   items.add(new LineItem("292B334",  1, 800.00);
   items.add(new LineItem("392C334",  5,  50.00);

   myPurchase.total = 2550.00;
}
```


Using Java Types

FUJITSU

instances or uses:

```
public void copyFromOld(String[] args)
{
   PurchaseOrder oldPurchase = findOldOne();
   PurchaseOrder newPurchase = new PurchaseOrder();

   newPurchase.customerName = oldPurchase.customerName;
   newPurchase.due = getDate(2010, 12, 12);
   Address mainAddress = oldPurchase.shipTo;
   newPurchase.shipTo = mainAddress;
   newPurchase.billTo = mainAddress;

   newPurchase.shipTo.street = “315B Elm Street.”;

   items = new List<LineItems>();
   items.add(oldPurchase.items.get(1));
   items.add(oldPurchase.items.get(2));
   items.add(oldPurchase.items.get(3));

   myPurchase.total = 2550.00;
}
```


Custom Data: XML Schema

FUJITSU

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.example.com/PO1" xmlns:po="http://www.example.com/PO1">

 <xsd:complexType name="PurchaseOrder">
  <xsd:sequence>
   <xsd:element name="customerName" type="xsd:string"/>
   <xsd:element name="dueDate"     type="xsd:date"/>
   <xsd:element name="shipTo"      type="po:Address"/>
   <xsd:element name="billTo"      type="po:Address"/>
   <xsd:element name="totalCost"   type="xsd:decimal"/>
   <xsd:element name="items"  type="po:LineItem" minOccurs="0“ maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="Address">
  <xsd:sequence>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city"   type="xsd:string"/>
   <xsd:element name="state"  type="xsd:string"/>
   <xsd:element name="zip"    type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="LineItem">
  <xsd:sequence>
   <xsd:element name="productName" type="xsd:string"/>
   <xsd:element name="quantity"    type="xsd:string"/>
   <xsd:element name="price"       type="xsd:decimal"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```


Custom Data: Object Types

FUJITSU

NameSpace	TypeName
http://www.sample.com/PO1	Address
http://www.sample.com/PO1	LineItem
http://www.sample.com/PO1	PurchaseOrder

In order to use these types, import the schema file in studio.
All types in a particular schema are immediately available for use.

declare a UDA with type:

Address@http://www.sample.com/PO1

Custom Data: Declare UDAs

FUJITSU

UDA definition	Type Declaration
myPurchase	Purchase@http://www.example.com/PO1
myAddress	Address@http://www.example.com/PO1
myStreet	String
myCity	String

UDAs are defined exactly in the same way as the variables are declared in the Java program.

Custom Data UDA variables

FUJITSU

UDA myPurchase *contains:*

```
<customerName>Helmut Baer</customerName>
<dueDate>20101212</dueDate>
<shipTo>
 <street>315 Elm Street</street>
 <city>Oakhurst</city>
 <state>AZ</state>
 <zip>87654</zip>
</shipTo>
<billTo>
 <street>315 Elm Street</street>
 <city>Oakhurst</city>
 <state>AZ</state>
 <zip>87654</zip>
</billTo>
<totalCost>1550.00</totalCost>
<item>
 <partNo>192D334</partNo>
 <quantity>15</quantity>
 <price>100.00</price>
</item>
<item>
 <partNo>292B334</partNo>
 <quantity>1</quantity>
 <price>800.00</price>
</item>
<item>
 <partNo>392C334</partNo>
 <quantity>5</quantity>
 <price>50.00</price>
</item>
```

UDA myStreet *contains:*

315 Elm Street

Values are “copies” of each other, not “references”

UDA myAddress contains:

```
<street>315 Elm Street</street>
<city>Oakhurst</city>
<state>AZ</state>
<zip>87654</zip>
```

There is no root tag!

RefData interface

FUJITSU

- Interface RefData represents:
  - A part of a Complex UDA
  - A Complex UDA
  - A Process Context (the UDAs in the Process Instance)
- Interface ProcessInstance extends RefData

```
public interface RefData
{
   public String getValue(String XPath);
   public void   setValue(String XPath, String value);

   public RefData getRef(String XPath);
   public void    copy  (String XPath, RefData source);

   public void remove(String XPath);
}
```


Java Object Interface

FUJITSU

- An interface that is a reference to a point in the XML tree

  RefData
- One call to get a reference object

  public RefData getRef(String XPath)
- Calls to get and set the value, if it is a leaf

  public String getValue(String XPath)

  public void setValue(String XPath, String value)
- One call to set the value (if it is not a leaf)

  public void copy(String XPath, RefData source)
- One call to remove an element from the XML tree

  public void remove(String XPath)

Manipulating with Java Objects

FUJITSU

```
pi.setValue("myStreet", "315 Elm Street");

pi.copy    ("myAddress/street", pi.getRef("myStreet"));
pi.setValue("myAddress/city" , "Oakhurst");
pi.setValue("myAddress/state", "AZ");
pi.setValue("myAddress/zip" , "87654");

pi.setValue("myPurchase/customerName", "Helmut Baer");
pi.setValue("myPurchase/dueDate"    , "2010-12-12");
pi.copy    ("myPurchase/shipTo"     , pi.getRef("myAddress"));
pi.copy    ("myPurchase/billTo"     , pi.getRef("myAddress"));

pi.setValue("myPurchase/items[1]/partNo" , "192D334");
pi.setValue("myPurchase/items[1]/quantity", "15");
pi.setValue("myPurchase/items[1]/price"  , "100.00");
pi.setValue("myPurchase/items[2]/partNo" , "292B334");
pi.setValue("myPurchase/items[2]/quantity", "1");
pi.setValue("myPurchase/items[2]/price"  , "800.00");
pi.setValue("myPurchase/items[3]/partNo" , "392C334");
pi.setValue("myPurchase/items[3]/quantity", "5");
pi.setValue("myPurchase/items[3]/price"  , "50.00");
```


Manipulating with Java Objects (2)

FUJITSU

```
pi.setValue("myStreet", "315 Elm Street");
RefData streetObj = pi.getRef("myStreet");

RefData address = pi.getRef("myAddress");
address.copy    ("street", streetObj);
address.setValue("city" , "Oakhurst");
address.setValue("state", "AZ");
address.setValue("zip"  , "87654");

RefData purchase = pi.getRef("myPurchase");
purchase.setValue("customerName", "Helmut Baer");
purchase.setValue("dueDate",     "2010-12-12");
purchase.copy    ("shipTo",      address);
purchase.copy    ("billTo",      address);

RefData item = purchase.getRef("items[1]");
item.setValue("partNo" , "192D334");
item.setValue("quantity", "15");
item.setValue("price"  , "100.00");
item = purchase.getRef("items[2]");
item.setValue("partNo" , "292B334");
item.setValue("quantity", "1");
item.setValue("price"  , "800.00");
item = purchase.getRef("items[3]");
item.setValue("partNo" , "392C334");
item.setValue("quantity", "5");
item.setValue("price"  , "50.00");
```


XML and Java reference

FUJITSU

- Path notation, or separate calls. The following two forms are the same:

copies the part number value from the first item from the purchase record in the myPurchase UDA into the returned XData object

```
getRef("myPurchase/items[1]/partNo")
```

```
getRef("myPurchase").getRef("items[1]").getRef("partNo")
```

gets a RefData object with a reference to the myPurchase UDA which is returned and used anonymously

gets a RefData object with a reference to the first item record in the UDA which is used anonymously

gets a RefData object with a reference to the partNo value

Manipulating Collections

FUJITSU

- An empty tag is equivalent to a null value in Java.
- An empty tag is a tag without any children tags (e.g. <items/>).
- Empty tags can be added as needed to make the array bigger.
- getValue from an empty tag should return null
- getRefData from an empty tag should return a regular RefData object
- setValue of a tag to null or “”, should result in making that tag empty.

sets the first item tag within the purchase, replacing whatever was there	purchase.setValue("items[1]", item);
Sets the tenth item tag within the purchase, replacing whatever was there. It may have to create up to 9 items tags in front of this.	purchase.setValue("items[10]", item);
Removes all children of the fourth items tag and makes it an empty tag.	purchase.setValue("items[4]", null);
Removes the fourth items tag.	purchase.remove("items[4]");

More on null & missing values

FUJITSU

- Calling remove for a tag will remove that tag
- Setting a path that does not exist, adds elements as necessary to allow the specified element to exist.

```
<address>
 <street>44 Deerfallen Way</street>
 <city>Albuquerque</city>
 <state>NM</state>
</address>
```

```
RefData address = getReferenceToTheValueAbove();
String country = address.getValue("address/country");
address.remove("address/state");
address.setValue("address/post/xx:office/code", "87123");
```

After the above statements:
country will have the value of null (because it does not exist) without throwing any exception

the address variable will hold the following value:
(new tags are generally added at the end)

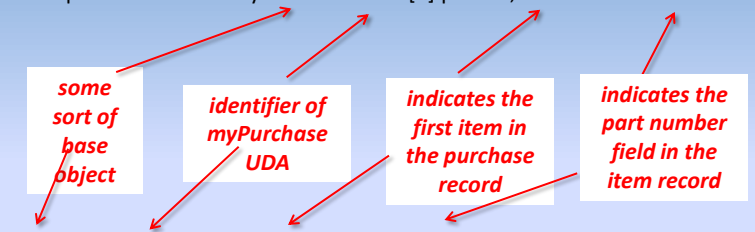
```
<address>
 <street>44 Deerfallen Way</street>
 <city>Albuquerque</city>
 <post>
  <xx:office>
   <code>87123</code>
  </xx:office>
 </post>
</address>
```


JavaScript Dot Notation (1)

FUJITSU

- Java script offers a clever way to reduce the complexity of the expressions. :

```
var expectedPart = ctx.myPurchase.items[1].partNo;
```




```
ctx.myPurchase.items[1].partNo = “89950”;
```

Upper expression takes the value of the part number, and puts it into the JS variable.

The lower expression puts the value 89950 and places it directly into the part number of the first item in the purchase record in the myPurchase UDA.

JavaScript Dot Notation (2)

FUJITSU

- values can be moved from one part of a complex uda value to another part of that (or another) uda:

ctx.myPurchase.items[3].partNo = ctx.myPurchase.items[1].partNo;

Expression takes the value of the part number of the first item in the myPurchase UDA value, and puts it directly into the part number of the third item in the purchase record in the myPurchase UDA.

var addrObj = ctx.myAddress;
ctx.myPurchase.shipTo.city = addrObj.city;

- Copies the value of myAddress to a local variable, and then copies the city field of the shipTo field of the purchase record in the myPurchase UDA.

FUJITSU

shaping tomorrow with you