Lesson 2: Types and Transformation

Lesson 2 introduces two of XML's most powerful features – the ability to create user-defined types, and the ability to transform one document into another document. We'll cover the steps of defining two document types, and transforming data from one format to the other. The various stages of the project for this lesson are stored as projects quote1- quote4 in the tutorials component archive, Lesson 2 folder.

XML (eXtensible Markup Language) is similar to HTML in that you use markup language (tags) to create documents. Both XML and HTML have evolved from, and are subsets of SGML (Standard Generalized Markup Language.) XML holds out the promise of revolutionizing communications by "enabling the universal exchange of intelligent data."

Rather than treating a document as a monolithic whole, XML lets you separate the data and structure. This means you can perform processing of data without concern for the appearance of the data. And it means that you can readily develop new formats for displaying the content without worrying about the actual content. You can transform the same set of data from one format to another, without having to go back to the server for another copy of the data.

XML doesn't just provide a different set of tags from HTML. Instead, it provides a means for creating extensible, or user defined "tags" as you need them. These tags are specified in DTDs using element type declarations, which specify the element name, content model (the type of information it can contain) and the attributes associated with the element.

This ability to specify custom types has many advantages: among them the ability to publish a "contract" with the users of a component that describes the interface (that is the input and output) to the component.

In this lesson we'll deal with the creation of document types, and transformations between document types. You'll see that Component X Studio relieves you of the chore of hand coding type definitions, and the XSL or Java code to perform transformations between document types. Remember that a key feature of XML is the ability to extend the language infinitely by adding your own custom types. Also, the document type definitions (DTDs) lay out the structure and rules for the XML, and form a "contract" that others can rely on when developing applications to interface with yours.

2.1 The Problem

Consider a company that offers a very small product catalog of two products -- tennis balls, which come in white and yellow, and golf gloves, for which there is no choice of color. The entire catalog is shown in Figure 2-1.

Figure 2-1: Sample Catalog

ACME SPORTING GOODS, LTD **2001 Spring Catalog**	
#1 Tennis Balls Vacuum sealed can of 3. Specify color: White, Yellow	**#2 Golf Glove** Premium cabretta leather. White only. One size fits all.

The catalog does not contain prices, perhaps because prices are dependent on volume. If you want to know the price you need to request a quote. The quote request mechanism, which eventually will be integrated into a web site, must only include the product number, and the quantity of interest.

Thus a quote request (written in XML) for a golf glove is:

```
<quoteRequest>
    <productNo>2</productNo>
    <quantity>1</quantity>
</quoteRequest>
```

A quote request for 2 cans of tennis balls is:

```
<quoteRequest>
    <productNo>1</productNo>
    <quantity>2</quantity>
</itemRequest>
```

The actual quote will contain the information provided in the quote request, as well as unit price and total price (which is quantity * unit price).

So, if the customer requests a quote for a golf glove, we expect the output in XML to be something like this:

```
<quote>
   <qty>1</qty>
   <pno>2</pno>
   <unitPrice>15.99</unitPrice>
   <totalPrice>15.99</totalPrice>
</quote>
```

And if the customer requests a quote for 2 cans of tennis balls, we expect theXML output to be something like:

```
<quote>
   <qty>2</qty>
   <pno>1</pno>
   <unitPrice>4.95</unitPrice>
   <totalPrice>9.9</totalPrice>
</quote>
```

The transformation of the quote request document into the quote document involves the following:

1. Passing through the product number and quantity.
2. Somehow looking up the unit price for the product.
3. Computing the total price (which is quantity * unit price.)
4. Using different names for the elements of the input and output documents.
5. Presenting the output elements in a different order than the input elements.

In this lesson you will create an application that transforms a customer quote request, (consisting of a product number and quantity) into a completed quote (including quantity, product number, unit price and total price.)

This lesson will take you step by step through the transformation, in the process creating document types, which can be used to enforce the business rules that we want to establish.

2.2 Creating a Document Type

(The completed example for this section is provided as project quote1 in lesson 2 folder of the cxTutorial archive.)

In this section you'll create an application containing a document type, add it to a component, and explore how it affects the component.

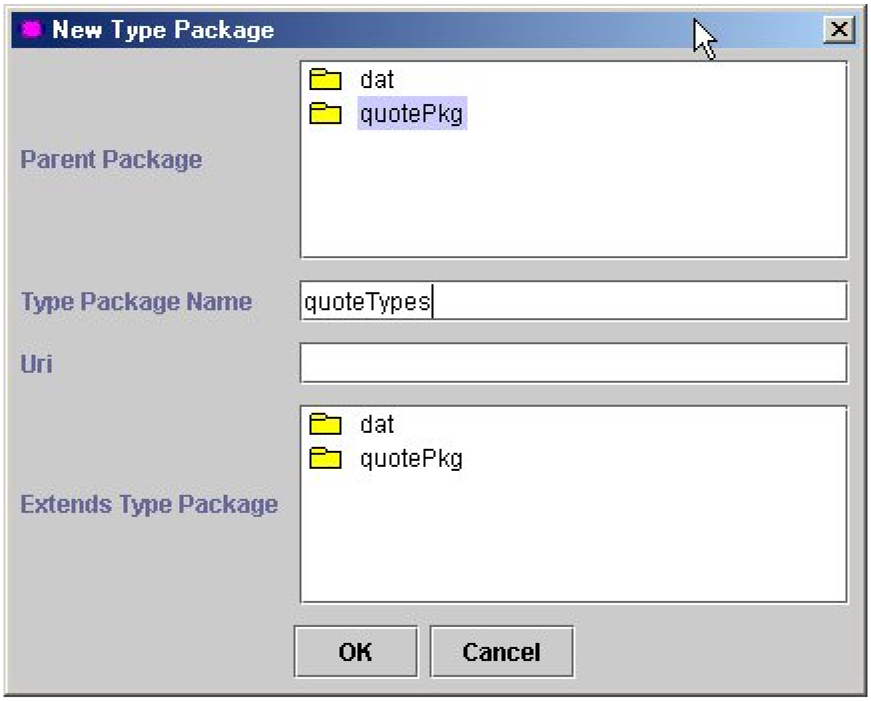
✓ Begin by creating a new project, and saving it in the tutorials archive, lesson 2 folder using the name quote. Remember to save your work periodically during this lesson to guard against inadvertent loss.

2. In the new project, create a convenience package for storing your components. Use the **New | New Package** menu option, or the button. When the dialog appears, specify the name for the package as quotePkg and click OK.

3. Now create a new *type package*, which corresponds to a DTD or schema. It defines a set of related types. You cannot store type components in a regular package.

   To create the type package, use the **New | New Type Package** menu item or the button. When the dialog appears, specify the name for the type packages as quoteTypes. Choose quotePkg as the parent package by clicking on it. Click OK.

Figure 2-2: Create the quoteTypes type package in the quotePkg convenience package.



Parent Package: quotePkg

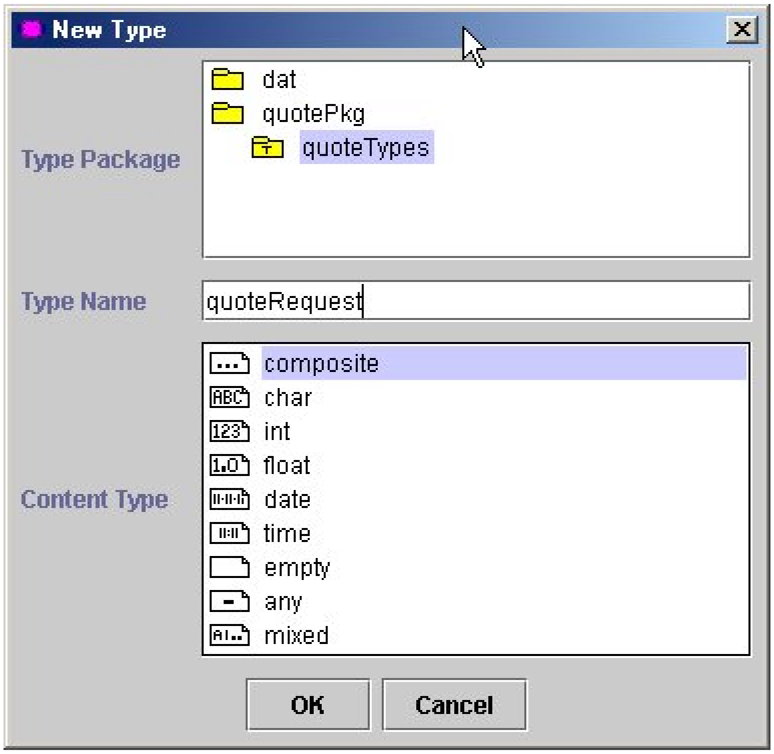
Type Package Name: quoteTypes

Uri:

Extends Type Package

4. Create a type using the **New | New Type** menu option or the (□) button. When the dialog appears, expand the Type Package tree and click on the quoteTypes package to select it. Specify the type name quoteRequest, and the content type composite as shown in Figure 2-3. Composite types can contain other types. Click OK. The quoteRequest type will appear as shown in Figure 2-4.

Figure 2-3: Create the quoteRequest type as a composite type in the quoteTypes type package.

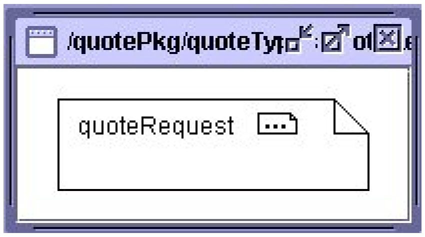


Type Package: quoteTypes

Type Name: quoteRequest

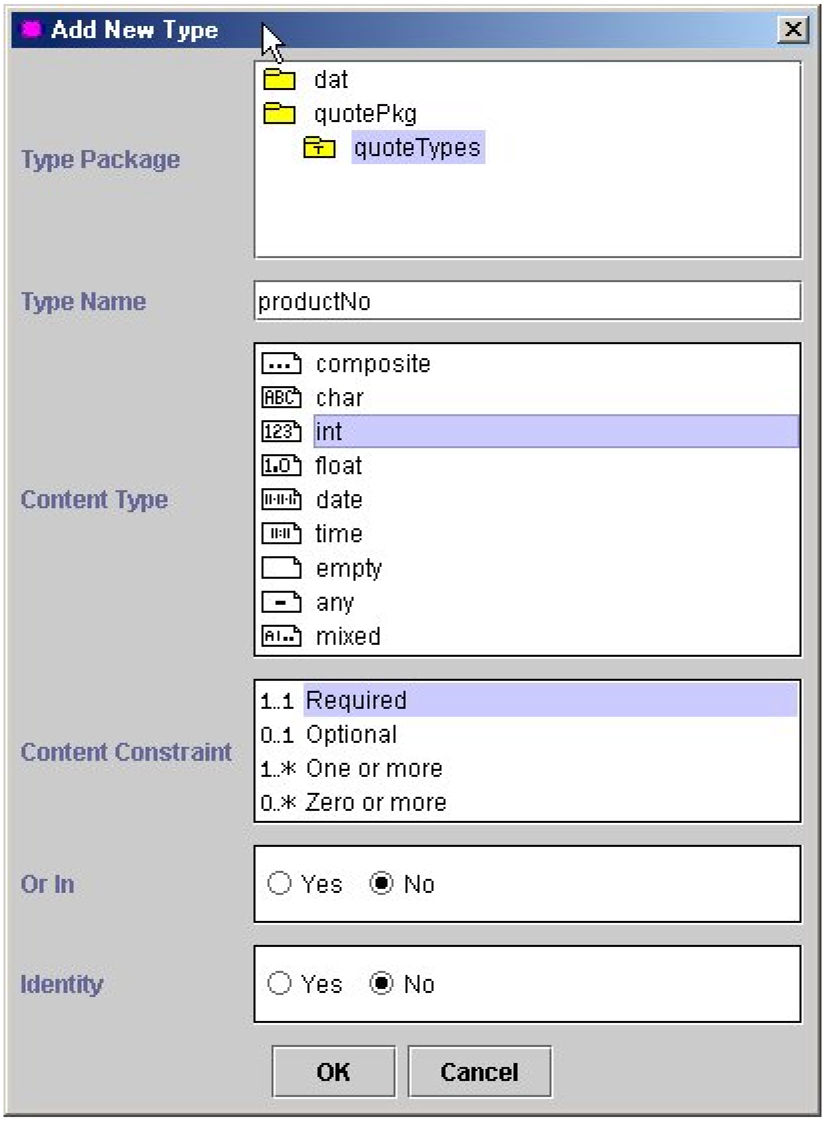
Content Type: composite

Figure 2-4: The quoteRequest type.



5. Next add an element called productNo to the quoteRequest type by clicking the Add New Type () button. Notice the plus sign, which differentiates this button from the “New Type” button you have used before. Complete the dialog as shown in Figure 2-5, and click OK. The quoteRequest type now appears as shown in Figure 2-6.

Figure 2-5: Add a productNo element to the quoteRequest type using the Add New Type dialog.


Add New Type

Type Package: dat, quotePkg, quoteTypes (selected: quoteTypes)

Type Name: productNo

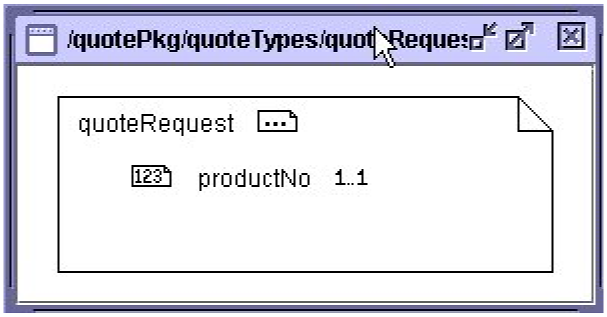
Content Type: composite, char, int, float, date, time, empty, any, mixed (selected: int)

Content Constraint: 1..1 Required, 0..1 Optional, 1..* One or more, 0..* Zero or more (selected: 1..1 Required)

Or In: ○ Yes ● No

Identity: ○ Yes ● No

Figure 2-6: The quoteRequest type after adding the productNo element.

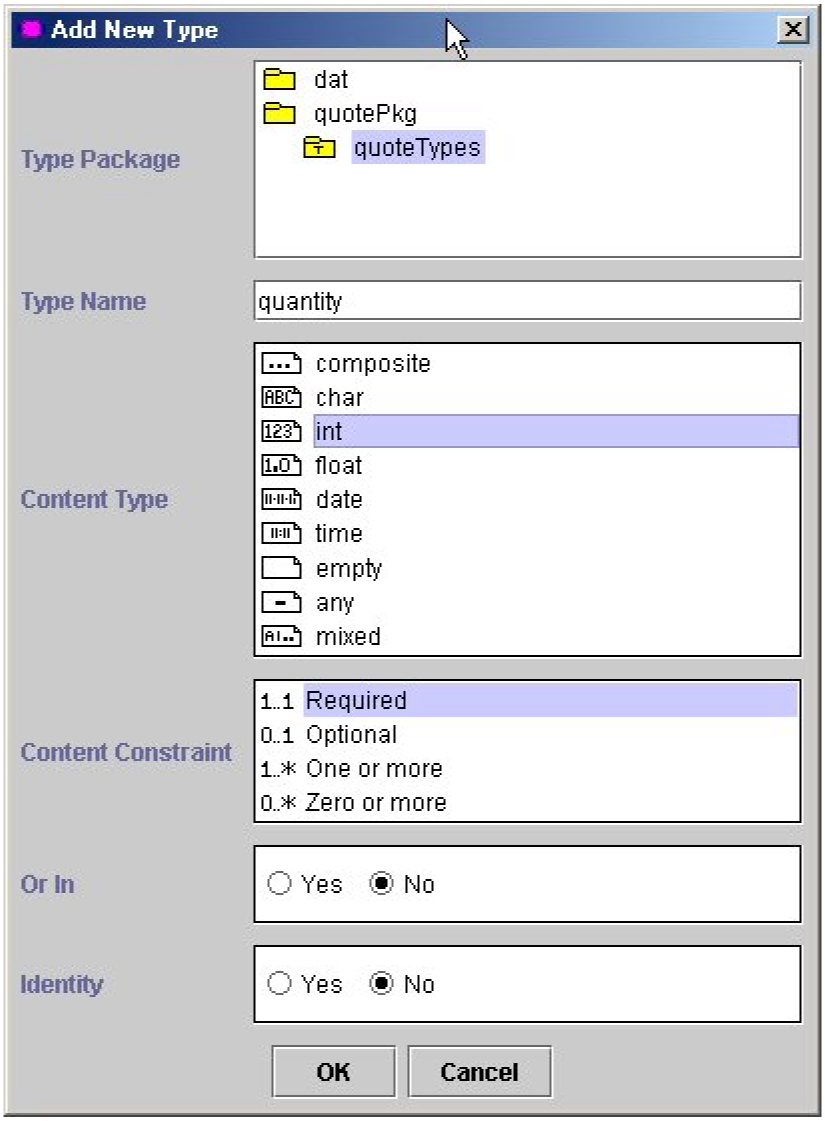

/quotePkg/quoteTypes/quot Reques

quoteRequest

productNo 1..1

6. Add another element called quantity, again using the Add New Type ( ) button. Complete the Add New Type dialog as shown in Figure 2-7 and click OK. The quoteRequest type is now complete, as shown in Figure 2-8.

Figure 2-7: Adding the quantity element to the quoteRequest type.


Add New Type

Type Package: dat, quotePkg, quoteTypes (selected)

Type Name: quantity

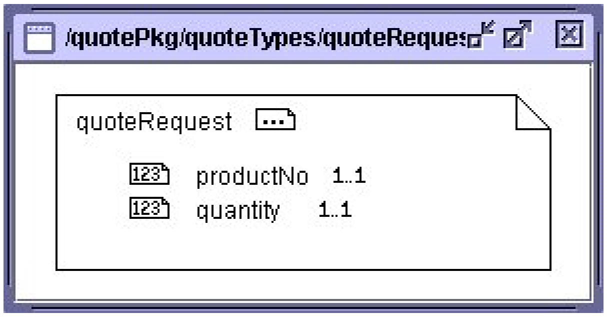
Content Type: composite, char, int (selected), float, date, time, empty, any, mixed

Content Constraint: 1..1 Required (selected), 0..1 Optional, 1..* One or more, 0..* Zero or more

Or In: ○ Yes ● No

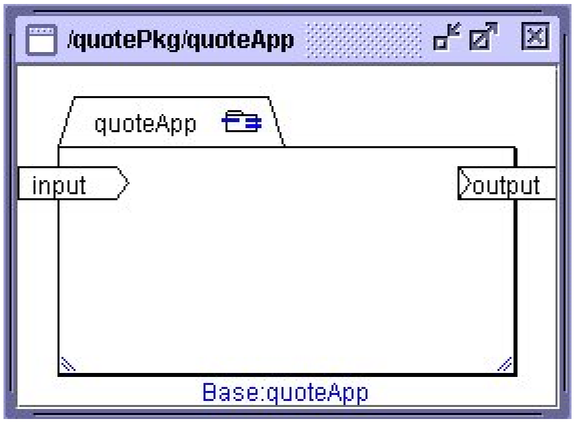
Identity: ○ Yes ● No

Figure 2-8: The completed quoteRequest type with productNo and quantity elements.



7. To test the quoteRequest type, use the **New | New Component** menu option or the button to create a new component called quoteApp in the quotePkg package. Add input and output ports as shown in Figure 2-9 by dragging them from the ports palette and dropping them over the respective margins of you new component.

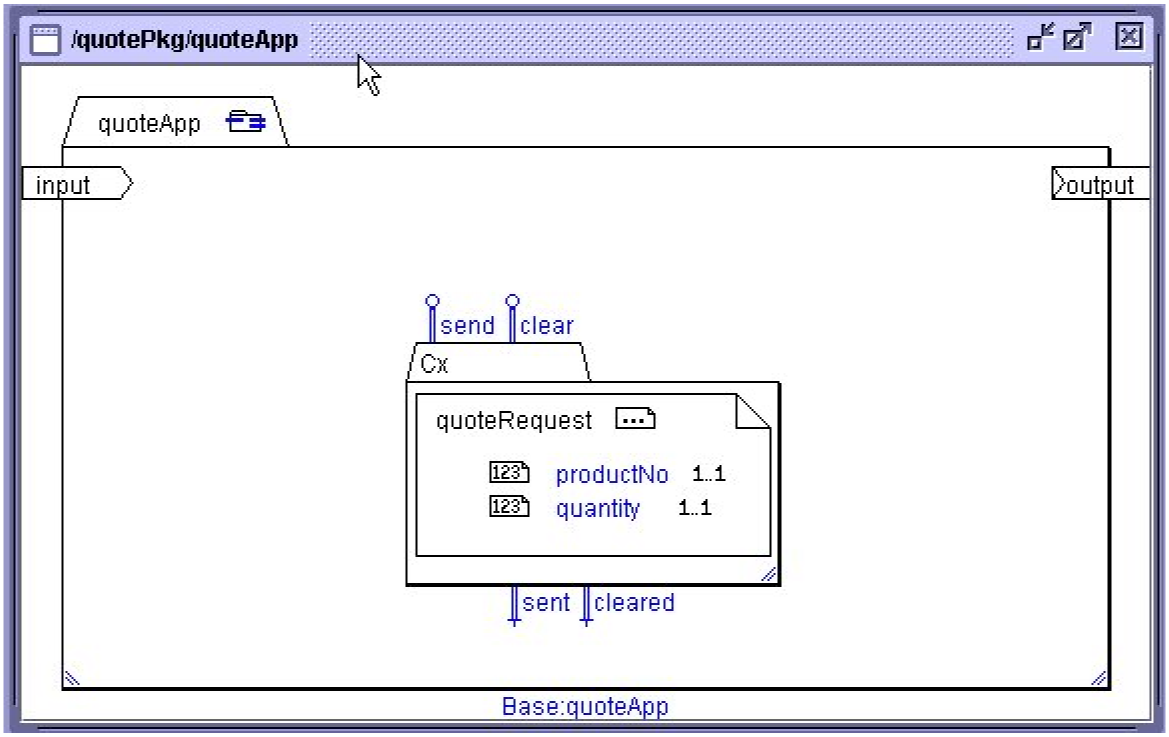
Figure 2-9: The quoteApp component, with input and output ports.



8. Resize the quoteApp component to make it bigger. Drag and drop a quoteRequest type from the component tree into the *center* of the quoteApp component. When you add a type component to a composite component, the type is placed inside of a *variable* component, as shown in Figure 2-10. Rename the variable component to quoteRequest by clicking over the name Cx on the components tab, typing quoteRequest, and pressing the enter key.

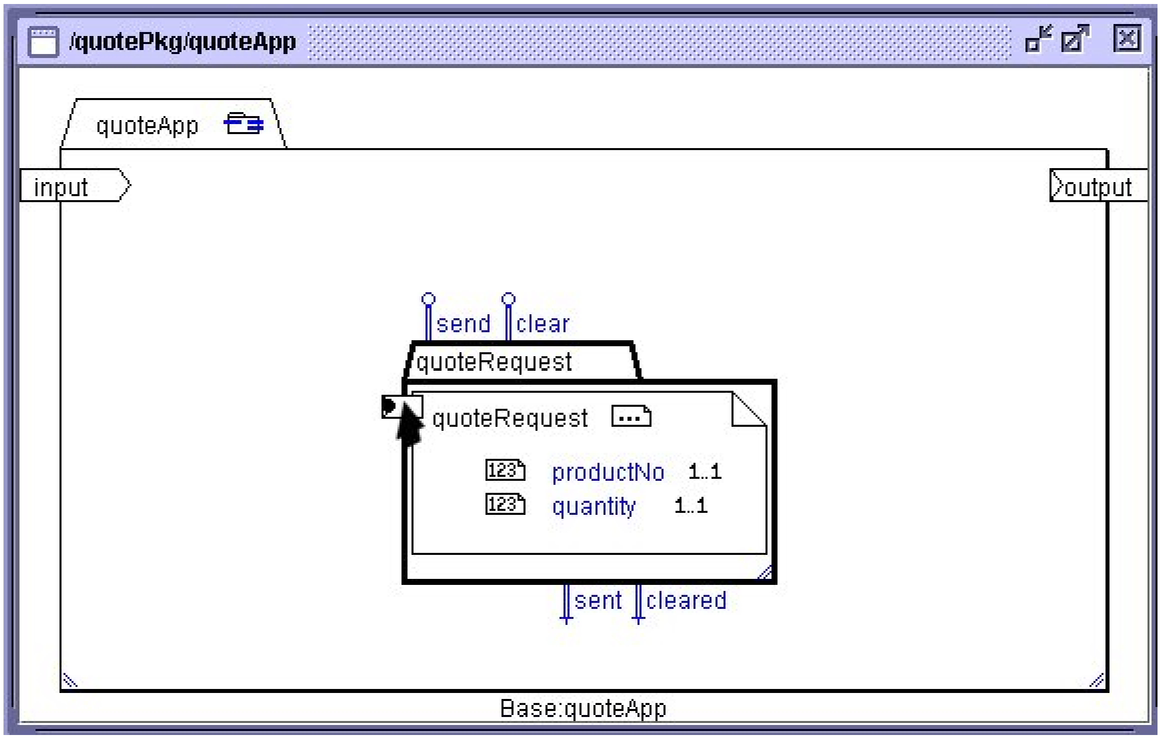
   Note that if the quoteApp component is not big enough, or if you drop the quoteRequest type too close to the edge of the parent container, you'll end up creating a quoteRequest *port*. If you make a mistake, just use the undo button ().

Figure 2-10: When the quoteRequest type is added to the quoteApp, it is placed in a "constant" component type, and behaves like a variable.



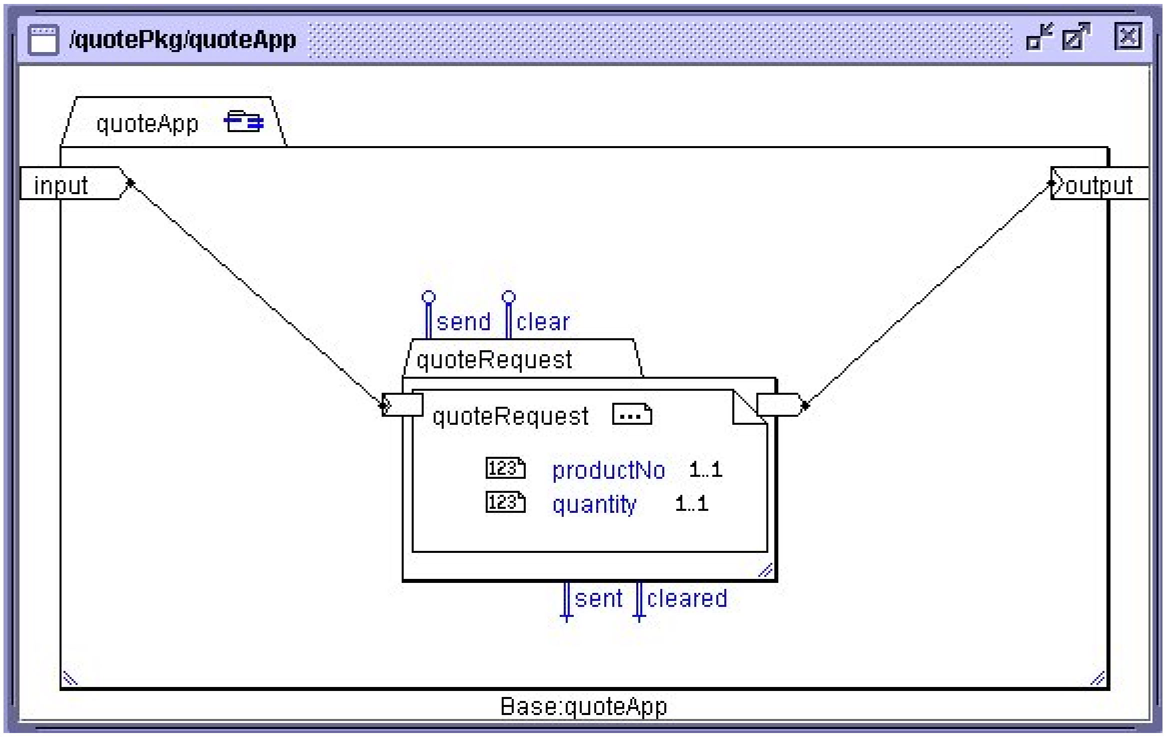
9. Position the mouse to the left of the quoteRequest component as shown in Figure 2-11. An input port will pop out. Drag from this input port to the input port for the quoteApp component. The two input ports will be wired together.

Figure 2-11: When the mouse hovers along the edge of the quoteRequest document, an input port will pop out.



10. Repeat the previous step positioning the mouse on the right side of the quoteRequest component to pop out an output port.. Drag from the pop out output port to the quoteApp output port to wire the two ports together, as shown in Figure 2-12.

Figure 2-12: Wiring the quoteRequest component to the input and output ports.



11. Now test the quoteApp by opening both the input and output ports (double click on each, or use the Open Component ( ) button) to open an XML editor for each.

12. To use the tree view mode to edit XML, you simply create a hierarchy of elements using the buttons on the toolbar. In the input port, click the Element ( ) button to create a new root level element. Change the name to quoteRequest.

13. Click the Child Element ( ) button to create an element under quoteRequest. Open the quoteRequest element by clicking the ⊶ sign, and change its name to productNo. In the value column, enter 1.

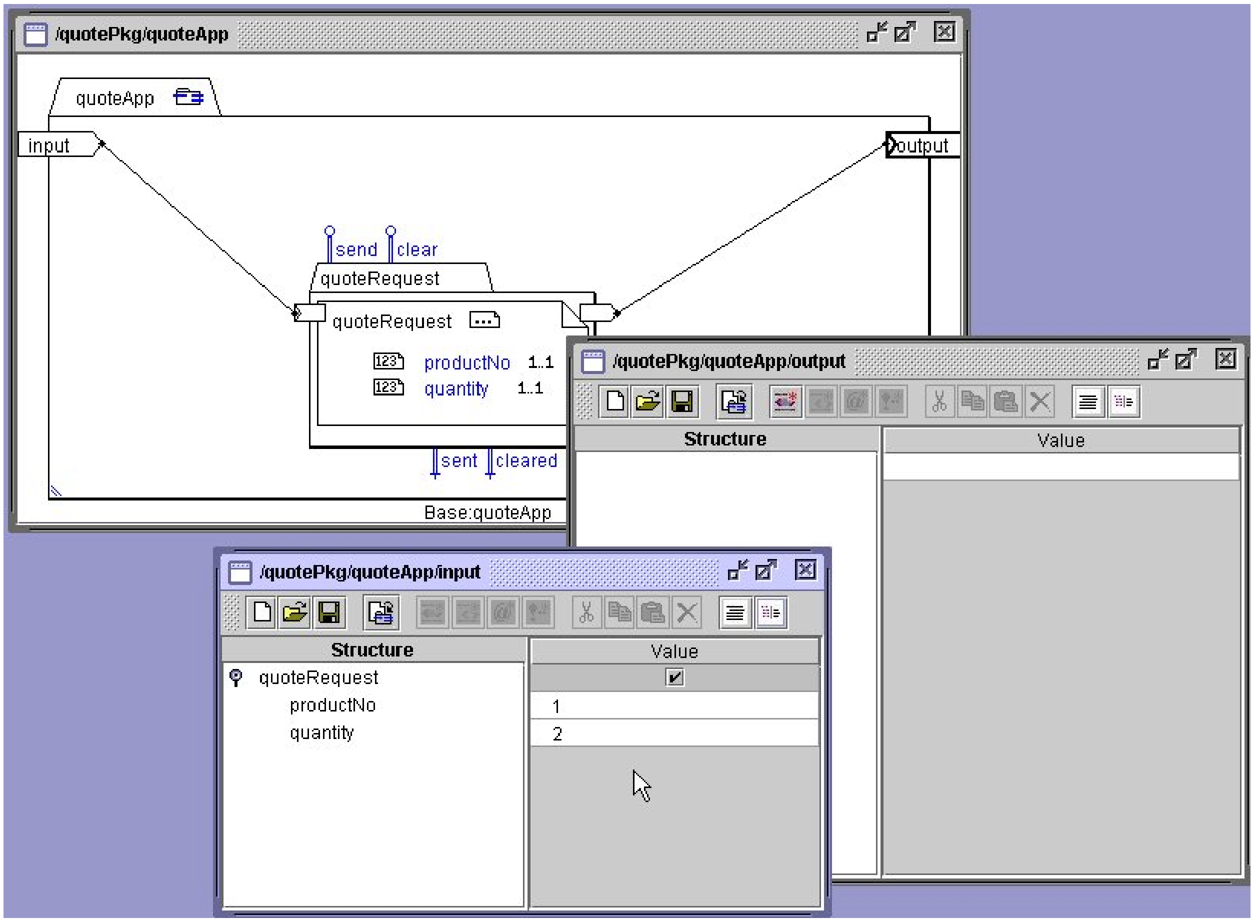
14. Highlight quoteRequest and click the Child Element ( ) button again to create another element under it, and change its name to quantity. In the value column, enter 2.

    The tree structure should appear as shown in
    Figure 2-13.

15. You can view the XML generated by clicking the text view (≣) button. The text view should contain the following:

```
<?xml version="1.0" ?>
<quoteRequest>
      <productNo>1</productNo>
      <quantity>2</quantity>
</quoteRequest>
```

Figure 2-13: Send the xml from the output component. Nothing appears in the output component.



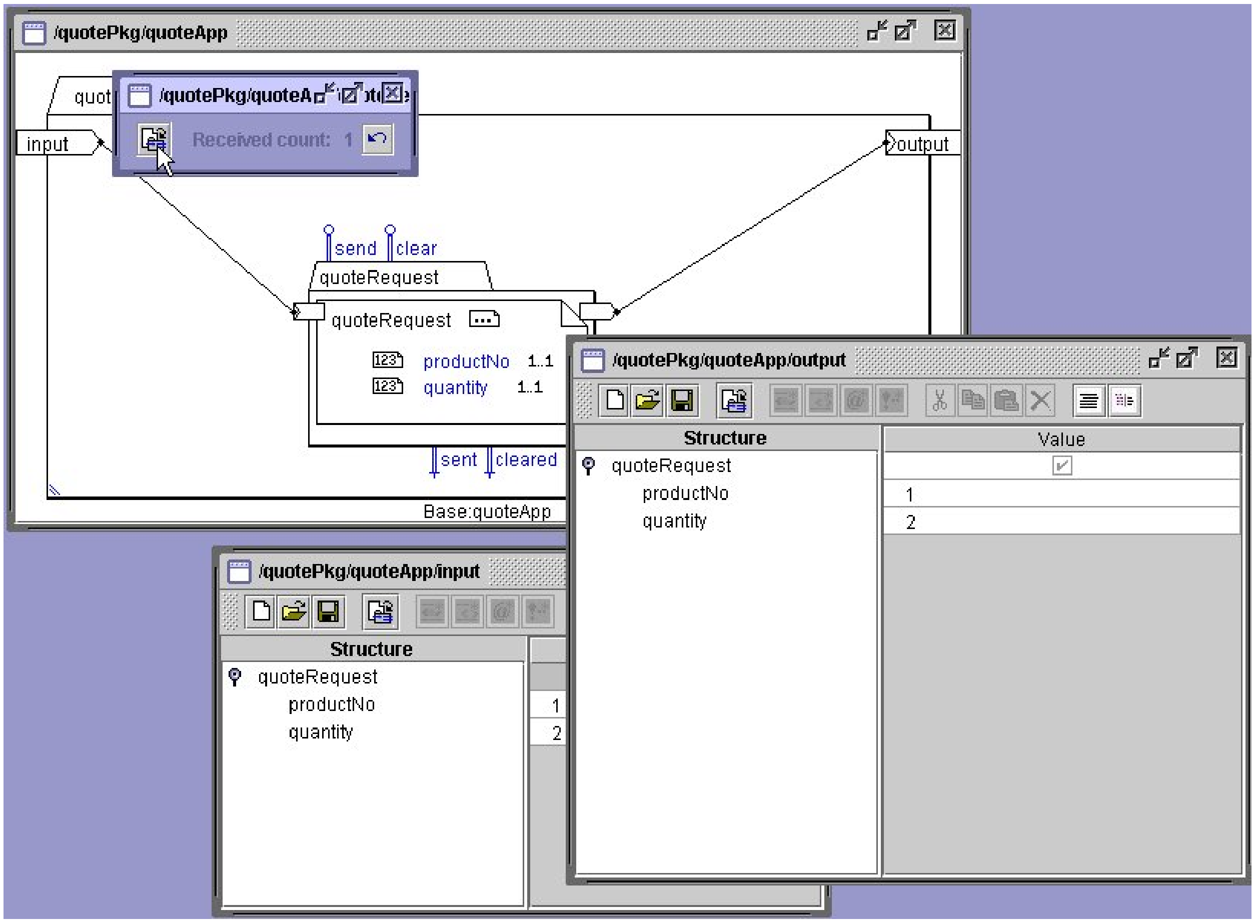
Structure	Value
quoteRequest	☑
productNo	1
quantity	2

16. Click the Send ( ) button. Switch to the output component and you'll see that nothing has happened. This is because the constant component that holds the quoteRequest type does not send the content through its output port until it receives a *Send signal* through the *Send pin*.

17. To trigger the Send signal, open the Send pin at the top of the quoteRequest component by double clicking on the send pin or the Open Component menu item on the send pin’s pop-up menu. Click the Send ( ) button on the new window being displayed.

18. Now look at the output port. The XML that you sent from the input port has now reached the output port, as seen in Figure 2-14.

Figure 2-14: Using the Send button on the send pin to send output from the quoteRequest component.



Structure	Value
quoteRequest	☑
productNo	1
quantity	2

But what makes this example different from the simple hello example covered in Lesson 1? In this case, the input passes through a quoteRequest type, so that the only thing that can be output to the output port is a quoteRequest type. Remember that using Document Type Definitions (DTDs) creates a "contract" with the world, saying, in this case, that nothing will come through a quoteRequest output port that is not a valid quoteRequest.

19. You can test this by sending anything other than a valid quoteRequest from the input port. For example, if you send the XML <test/>, you will get no result.

2.3 Transforming One Document Type to Another Document Type

(The completed project for Section 2.3 and 2.4 is quote2 in the lesson 2 folder of the cxTutorial archive)

In the previous section you learned that a major advantage of using types is the ability to control what is output by your component. But what if you want to *transform* or change the input from one document type definition to another? For example, we want to take a quote request (product number and quantity) as input, and output a quote (quantity, product number, unit price and total price.)

There are many ways that a transformation can be performed, including using XML components, an expression evaluator, or even a Java component for complex transforms. In this case we'll simply map the elements of one type to the elements of another type.

1. Under quoteTypes, create a new composite type called quote using the New Type () button.

The quote document will contain two of the elements that were used in the quoteRequest type -- productNo and quantity -- but they will be renamed.

2. Drag the quantity type from the Component Tree to the quote document.

3. Rename the quantity element to qty by selecting the quantity element within quote document and then clicking the Rename () button.

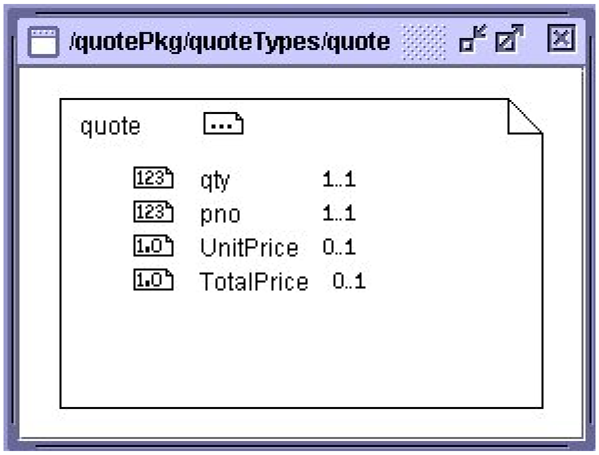
4. Drag the productNo type from the Component Tree to the quote document. Rename it to pno.

5. Add two new type elements to the quote type, as defined in Figure 2-15. The resulting quote type should look like Figure 2-16.

Figure 2-15: Specifications for the unitPrice and totalPrice elements of the quote document.

Type Package	Type Name	Content Type	Content Constraint
QuoteTypes	UnitPrice	Float	0:1 Optional
QuoteTypes	TotalPrice	Float	0:1 Optional

Figure 2-16: The completed quote document type.



/quotePkg/quoteTypes/quote

quote

123 qty 1..1
123 pno 1..1
1.0 UnitPrice 0..1
1.0 TotalPrice 0..1

6. Drag and drop the quote document type from the Component Tree into the quoteApp component and rename it to quote.

7. Since the goal is to output the quote document, not the quoteRequest document, delete the wire between the quoteRequest output port and the quoteApp output port. (To delete the wire, click on it to select it, and then click the Delete (✕) button.)

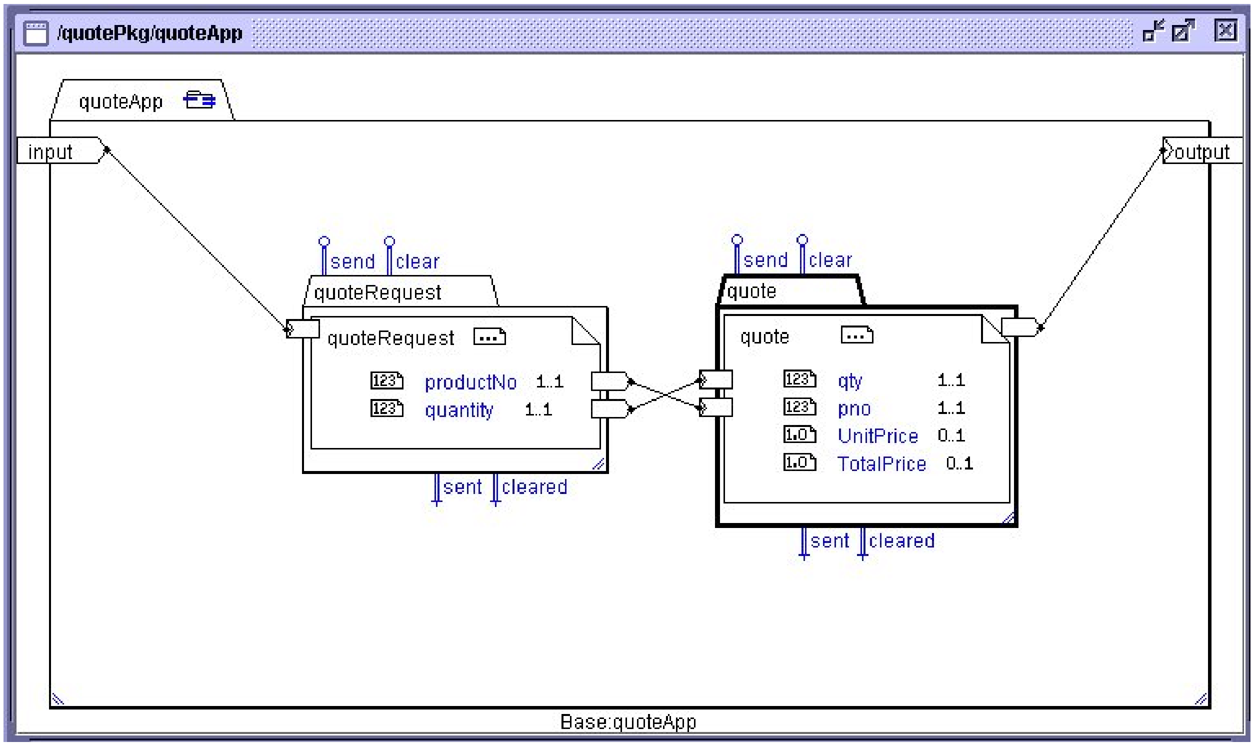
8. Now wire the quote output port to the quoteApp output port by dragging from one port to the other.

9. Next you need to wire the two type components together. However, since the quoteRequest only outputs a quoteRequest document type and the quote type only accepts as input a quote document type, you cannot just link the quoteRequest output port to the quote input port. Instead, you need to wire the individual elements together.

10. Drag from the pop out output port next to the productNo in the quoteRequest component to the pop out pno input port in the quote component.

11. Repeat the previous step for the quantity element. The result should appear as in Figure 2-17.

Figure 2-17: The quote document is added to quoteApp, and wired to the quoteRequest document and the output port.

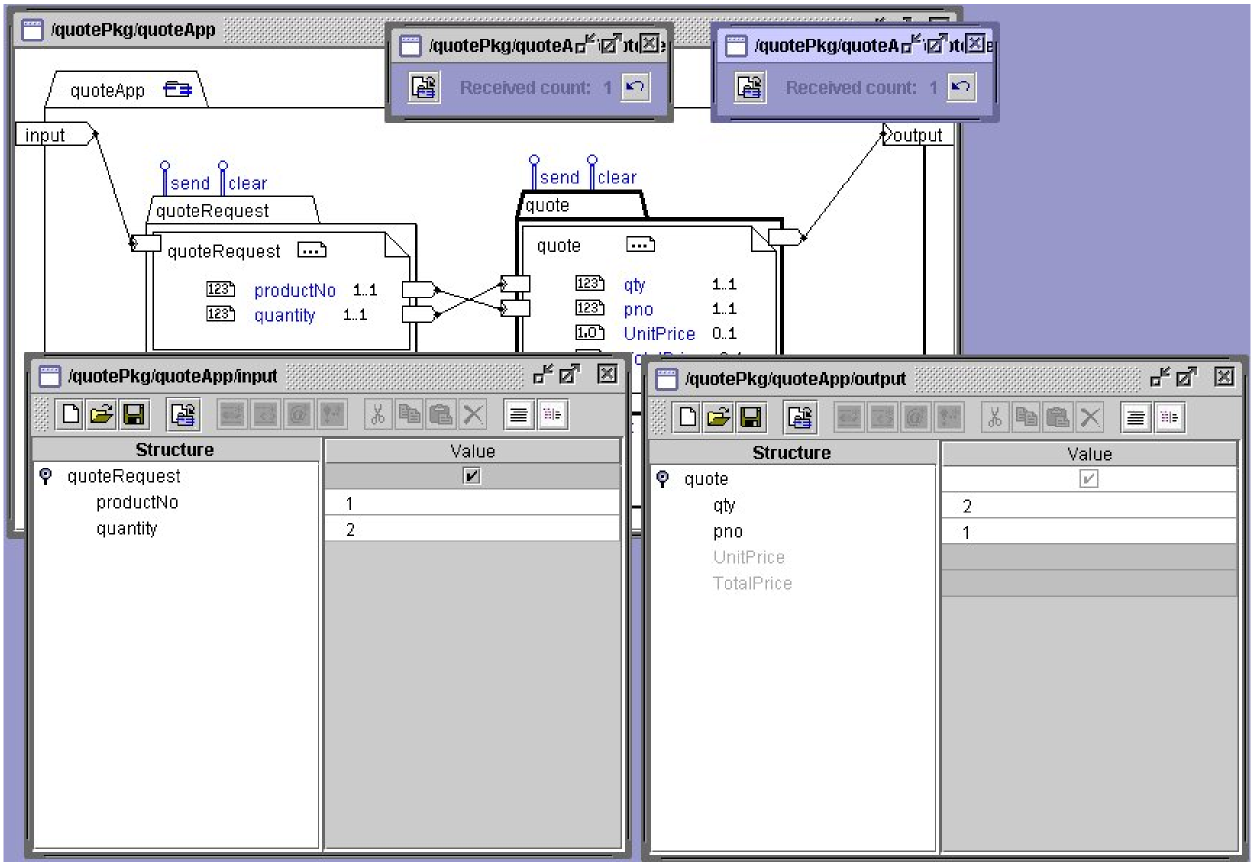


12. Notice that the unitPrice and total Price elements are not wired. But, since these were defined as optional fields, sending a valid quoteRequest into the application will generate a valid quote out. (That is, all the rules of the document type definition have been satisfied.)

13. Test the transformation by opening the quoteApp input and output ports and sending the xml shown. (Remember you can load qr1.xml to save typing.) Don't forget to trigger the send event in both the quoteRequest and quote documents, or you will get no output. The output should appear as in Figure 2-18.

    Pay particular attention to the name of the outer XML element -- it was quoteRequest in the input, and quote in the output.

Figure 2-18: Generating output requires that you issue sends from the input port, the quoteRequest send pin, and the quote send pin. Note the output document is in a different format than the input format.



/quotePkg/quoteApp/input

Structure	Value
quoteRequest	☑
productNo	1
quantity	2

/quotePkg/quoteApp/output

Structure	Value
quote	☑
qty	2
pno	1
UnitPrice	
TotalPrice	

2.4 More on Pins

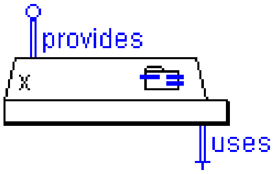
There is still more to do to complete the quote application but, before continuing, a little more on pins is in order. Completing the following steps will mean that you won't have to manually send signals to the quoteRequest and quote documents.

A component *uses* a pin when the pin acts as an output device through which the component signals the outside world that an event, such as a send or a receipt, has occurred.

A component *provides* a pin that the outside world can use to trigger an event (such as a send.)

Figure 2-19 shows a component that has an "input" pin called provides and an "output" pin called uses.

Figure 2-19: A component with provides and uses pins.



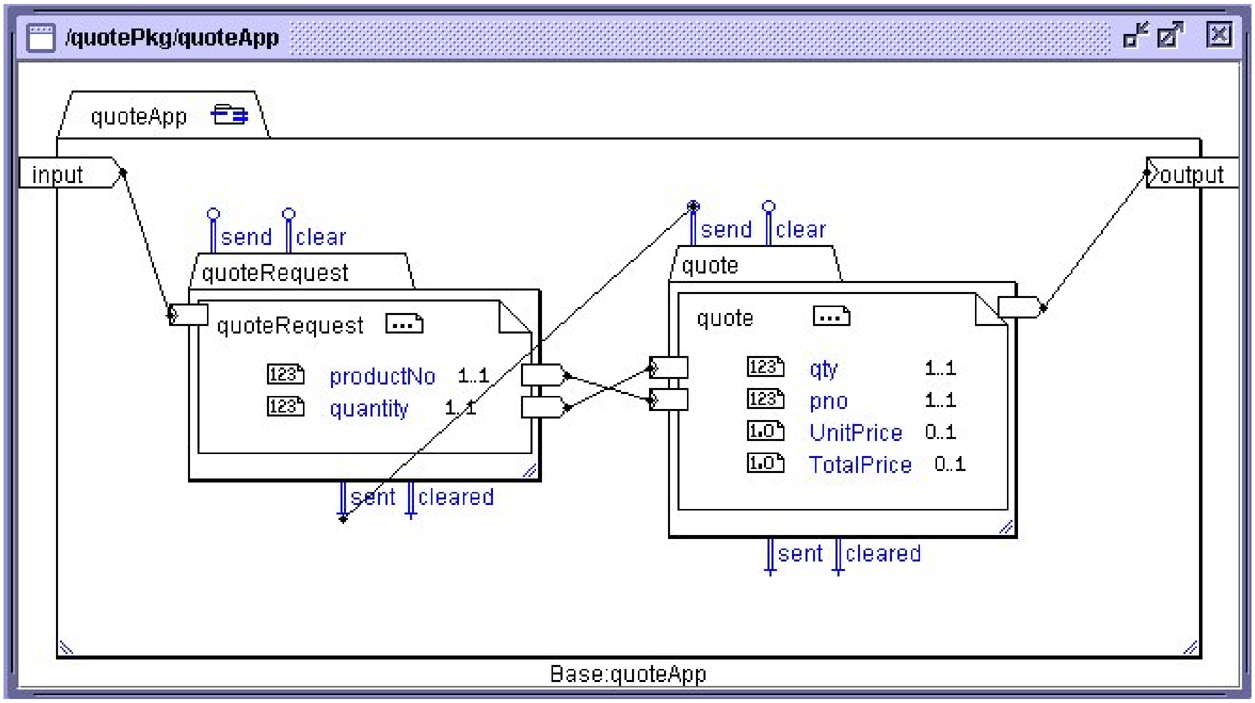
In the previous section you had to trigger a send event in both the quoteRequest and the quote components. You can wire the Sent pin of the quoteRequest component to the Send pin of the quote component. Then when the quoteRequest sends output, it automatically signals the quote component.

✓

1. Try it by wiring the quoteRequest sent pin to the quote send pin, as shown in Figure 2-20.

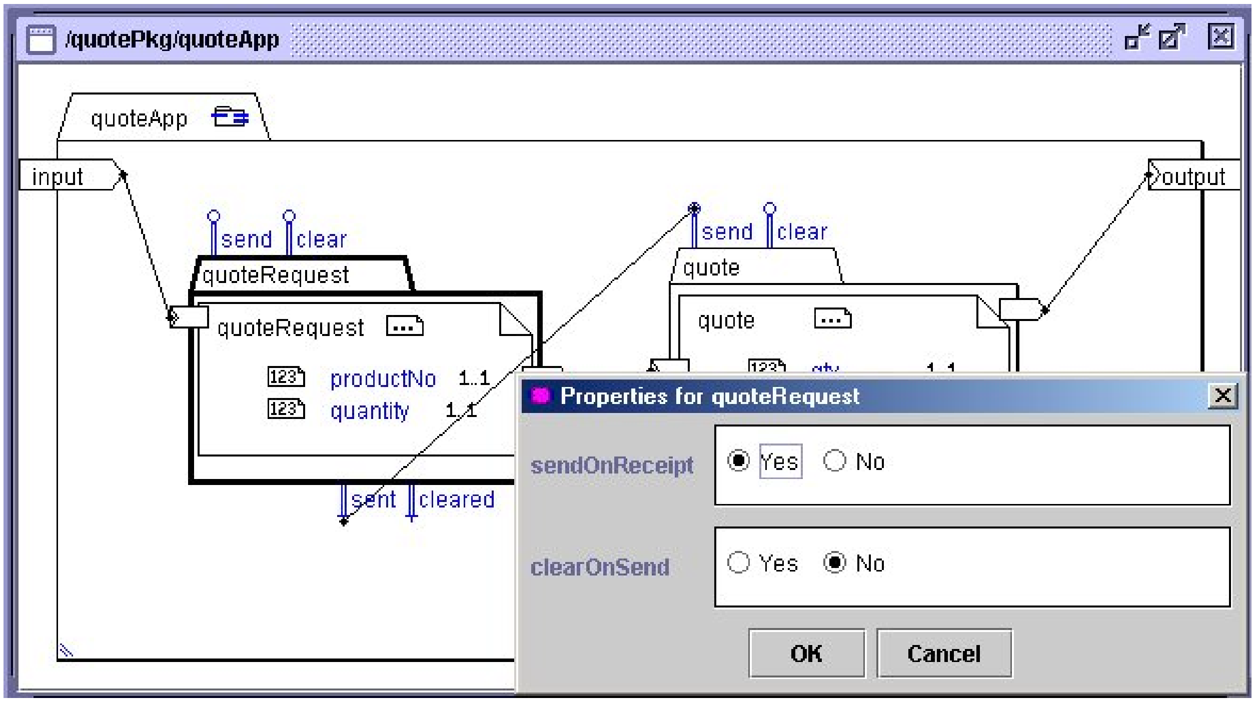
2. Then click the Send ( ) button on the input port component, and the Send button on the quoteRequest send pin. This time you won't have to click the Send button on the quote send pin. The quote document will appear in the output window.

Figure 2-20: Wire the quoteRequest sent pin to the quote send pin.



3. Variables can be configured to automatically send upon receiving input. To configure a component, open a components property window by selecting the component and then clicking the Properties ( ) button. Open the property window for the quoteRequest component and set the sendOnReceipt property to yes as shown in figure 2-21.

Figure 2-21: Setting the sendOnReceipt property.



Properties for quoteRequest

sendOnReceipt ● Yes ○ No

clearOnSend ○ Yes ● No

OK Cancel

4. Now try sending the xml again from the input port. The output report displays the quote document without you having to perform a manual "send" on either of the Send pins.

2.5 Using Types as Ports

(The completed project for this section is quote3 in the lesson 2 folder of the cxTutorial archive)

Earlier in this lesson we mentioned that you needed to drop a document type in the center of a composite component or you would create a port instead of a "variable." Now we are going to experiment with using document types as ports.

1. Delete the input and output ports on quoteApp by clicking to select them, and then using the ✕ button.

2. From the Component Tree quoteTypes Type package, drag the quoteRequest type to the left side of the quoteApp component.

   Note that the input port is named quoteRequest1. A 1 has been appended since the name quoteRequest is already taken by the variable component.

3. Rename the qouteRequest1 port to be quoteRequestIn.

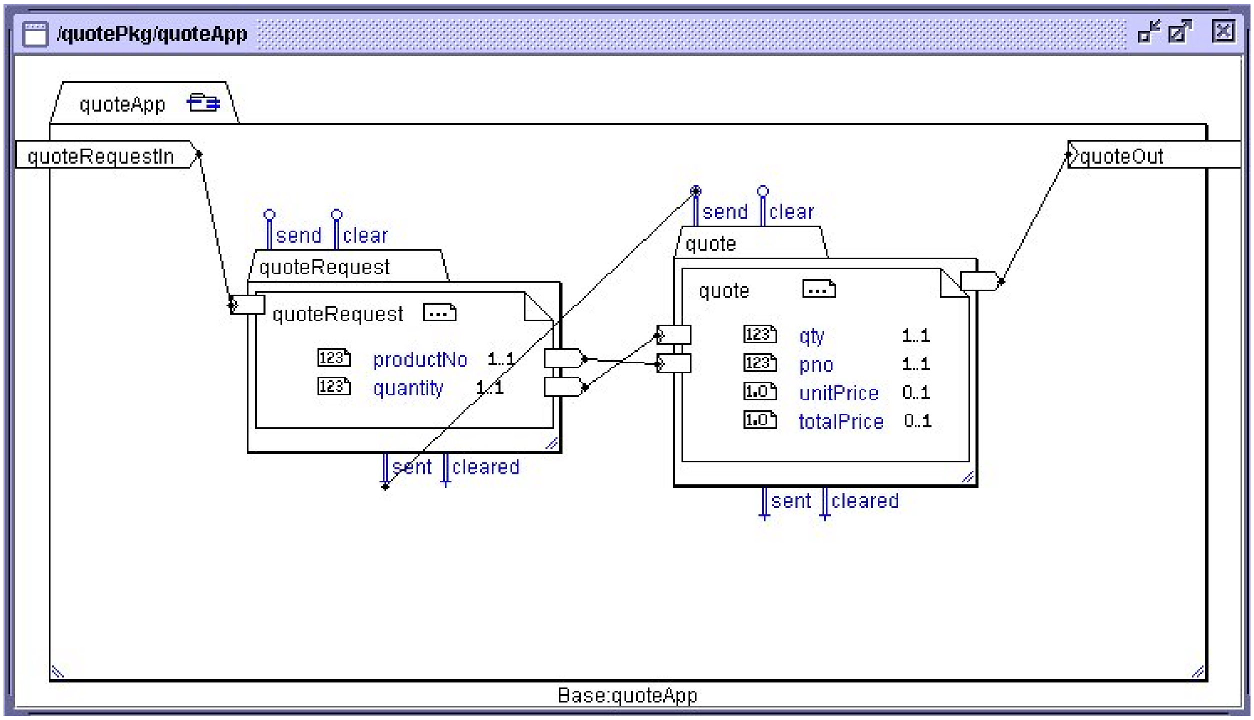
4. Repeat the previous step, this time dragging the quote type to the right side of the quoteApp component and rename it to be quoteOut.

5. Wire the new quoteRequestIn port to the input port on the quoteRequest document.

6. Wire the new quoteOut port to the port on the output port on the quote document. At this time your quoteApp component should look like figure 2-22.

7. Test the changes to the quoteApp as you did in the previous section. The results will be the same. However there is a major difference. Now what is "exposed" on the quoteApp is a set of well defined inputs and outputs that form a "contract" with the world. Anyone who wants to use this component can rely on it only accepting a quoteRequest type as input and only producing a quote type as output.

Figure 2-22: Create ports out of document types by dropping them on the edge of the quoteApp component, and rewiring.



2.6 Including Projects

(The completed project for this section is quote4 in the lesson 2 folder of the cxTutorial archive)

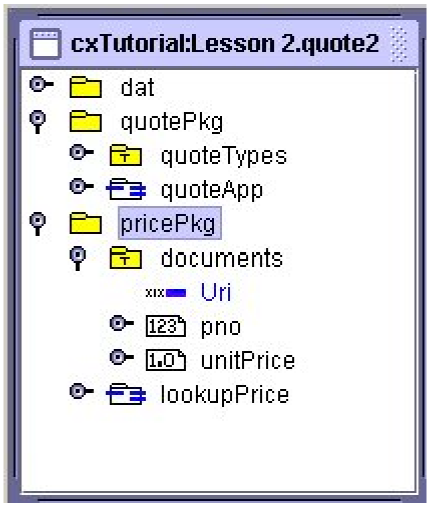
The quote application still doesn't provide the unit price and total price information. Rather than build that functionality from scratch, we'll include another project containing a component that will provide this functionality. The internals that make the included component work is immaterial right now. What is important is the interface the components provide, as you will see.

1. Use the **Project | Include** menu option.

2. When the Include dialog appears, select the `pricePkg` project in the lesson 2 folder of the cxTutorial archive.

3. Examine the Component Tree. You'll notice that the new package was added to the root. This is because it was originally created in the root. In the pricePkg package you will find a lookupPrice component. You will use this component to perform a price lookup in your quoteApp Component.

Figure 2-23: The imported components will appear in the pricePkg in the Component Tree.



About Included Projects

When you include a project, all components defined in the included project can be used within the project you are currently editing. However, you cannot edit the base definitions of the components defined in the included project. When opening the base definition the component display will be read only.

The core set components supplied with Component X are defined in a set of projects found within the cx archive. These are separate projects so that you only need to include those projects that contain the components you need for the functionality of the components you are writing.

When you create a new project, any projects defined in the **Set Default Project Properties** dialog are automatically included. You can add and remove projects from this default set by using the **Options | Set Default Project Properties** menu item. You can add and remove included projects from your project by using the **Project | Properties** menu item.

4. Drag the lookupPrice component from the Component Tree to the quoteApp component.

5. Wire the productNo output port of the quoteRequest component to the productNo input port on the lookupPrice component.

6. Wire the unitPrice port of the lookupPrice component to the unitPrice input port of the quote document component.

7. Drag an evaluator component from the filter palette to the quoteApp component. The evaluator component will be used to perform the computation needed to calculate the totalPrice. In order to perform this computation, the evaluator component will need to be configured.

8. Open the properties dialog for the evaluator component by first selecting the evaluator component and then click the Properties (▤) button.

9. Enter /unitPrice * $quoteRequest/ /quantity in the expr field. This expression will calculate a value by multiplying the unitPrice found in the input document by the quantity found in the quoteRequest variable.

About Expressions

Many of the core components supplied with Component X use expression that evaluate against XML documents. The form of expressions supported include extended XPath/XSL (extensible stylesheet language) expressions and Java like expressions.

A synopsis of XPath/XSL expressions is:

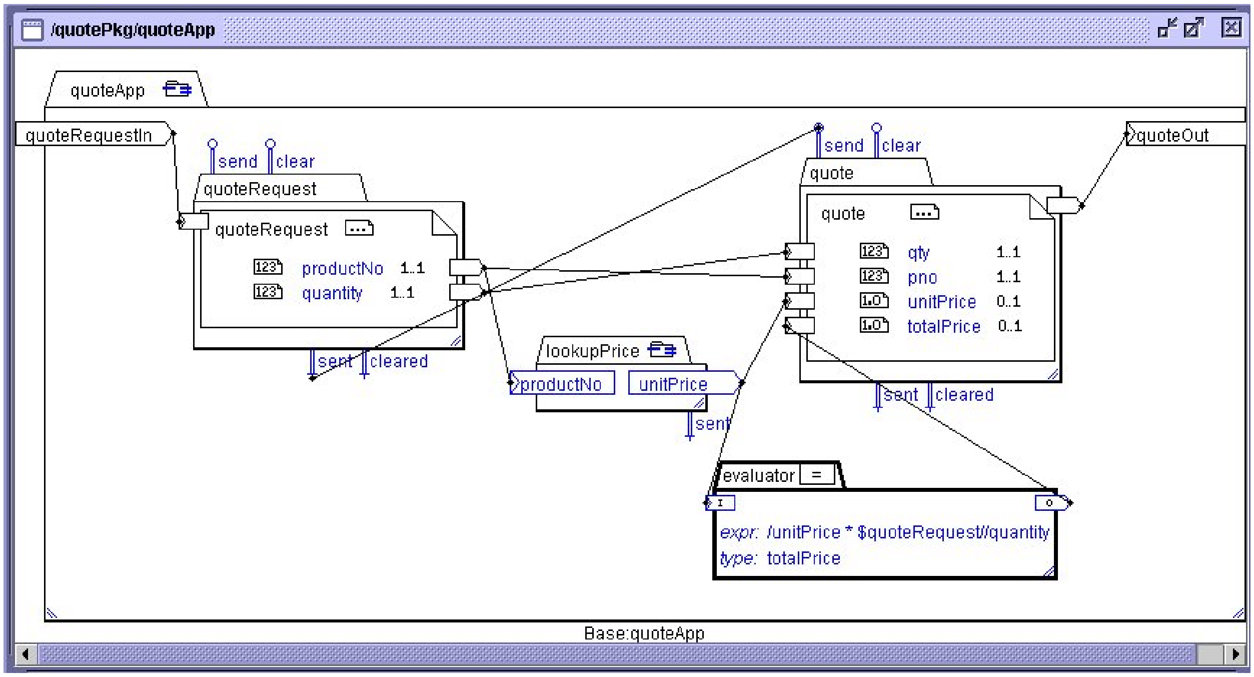
name	refers to an element at a given level in the document structure
/	at the beginning of a name path selection selects the document root.
/	between names selects the child by the second name of the parent by the first name.
//	selects any descendent by the name following.
@	selects the attribute by the name following.
+	performs addition.
-	performs subtraction
*	performs multiplication.
div	performs division.

The XPath/XSL expressions have been extended such that $*name* refers the a Component X variable.

10. Enter totalPrice in the type field and click OK.

11. Wire the unitPrice port of the lookupPrice component to the input port of the evaluator and the output of the evaluator to the totalPrice input port of the quote document, as shown in Figure 2-24

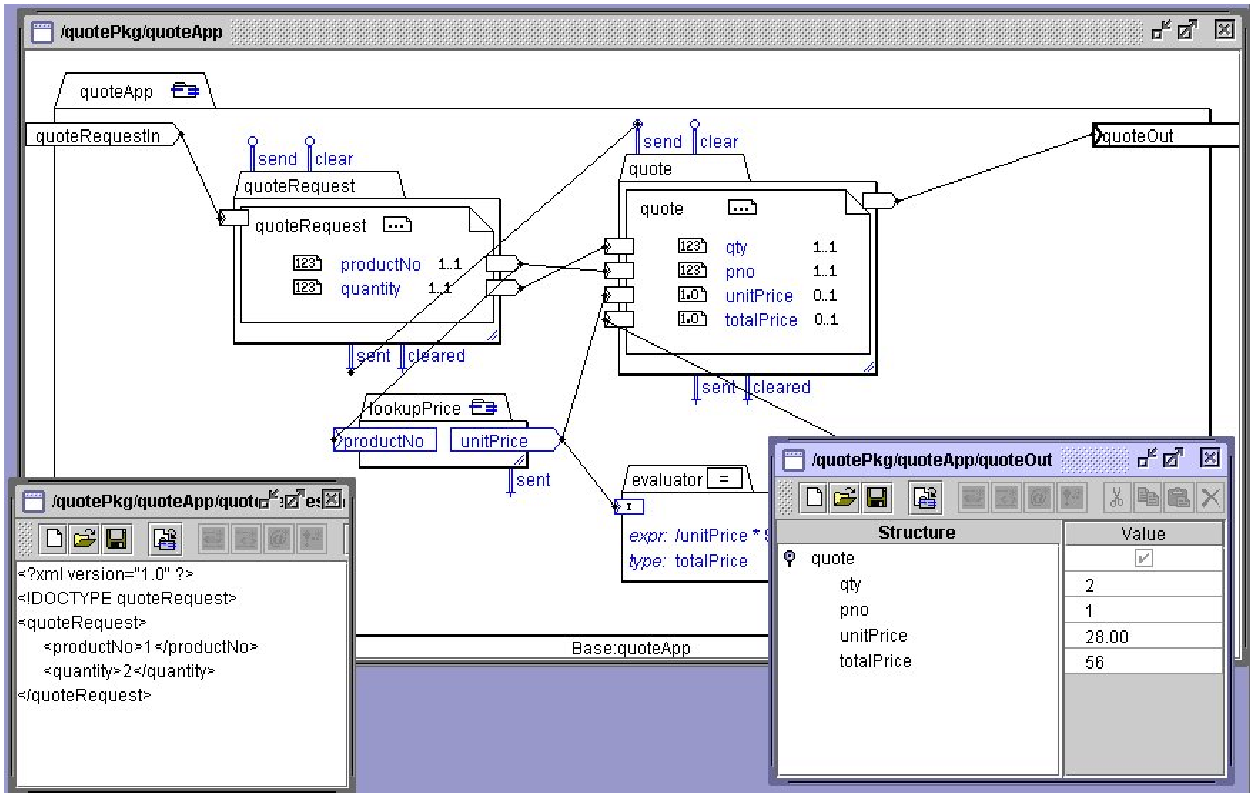
Figure 2-24: Add the lookupPrice component and evaluator component to quoteApp, and wire it to the quoteRequest and quote documents, matching the appropriate ports.



expr: /unitPrice * $quoteRequest//quantity
type: totalPrice

12. Now test the quoteApp by opening the quoteApp input (quoteRequestIn) and output (quoteOut) ports. From the quoteRequest port, open qr1.xml and send it. When the output appears, note that the unitPrice and totalPrice information is now filled in.

Figure 2-25: Testing the completed quote project.



expr: /unitPrice * $
type: totalPrice

```
<?xml version="1.0" ?>
<!DOCTYPE quoteRequest>
<quoteRequest>
    <productNo>1</productNo>
    <quantity>2</quantity>
</quoteRequest>
```

Structure	Value
quote	✓
qty	2
pno	1
unitPrice	28.00
totalPrice	56

13. Save your project as quoteApp. Your project now defines a component that you could use in other projects. To do so, you would simply include the quoteApp project and then create instances of your quoteApp component anywhere its functionality is required.

2.7 Review

In this lesson you learned how to define and use document types as "variables" and ports, and how to transform one document into another. You learned how pins work, and how to include other projects.

A key feature of XML is the ability to extend the language infinitely by adding your own custom types. The document type definitions (DTDs) lay out the structure and rules for the XML, and form a "contract" that others can rely on when developing applications to interface with yours.

Using types, you can easily accomplish transformations from one document type to another.

Types must be created in type packages, rather than in convenience packages. A composite type can hold other elements or attributes. You'll learn how to add attributes to types in lesson 3. Given a component with a published interface that accepts the desired input and produces the desired output, you don't need to know all the details of the processing going on behind the scenes. You can view such components as "black boxes."

Component X Studio relieves you of the chore of hand coding type definitions, and the XML to perform transformations between document types.

In the next lesson you'll make use of this knowledge to create a Market application that models a market of buyers and sellers.

2.8 Challenge Yourself

Test what you learned in this lesson with the following:

1. Try creating a component that transforms the quote back into a quoteRequest. Expect to lose some data in the process. (The completed project is in cxTutorial:lesson 2.q1.)

2. In a new project, include your original project then create a new component adding to it the completed quoteApp component and the component from the exercise above. Wire them together and test it out. (the completed project is in cxTutorial:lesson 2.q2.)