SAP Exchange Infrastructure 3.0 - Graphical Mapping Introduction and New Features

William Li

SAP Labs, LLC

Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions

- **Function Usage**
- **Simple Function Examples**
- **Node Function Examples**

Introduction to Most–Common Java Usage

User-defined Functions

- **Usage**

Advanced User–Defined Functions

- **Usage**
- **ResultList, Container Object, GlobalContainer Object, MappingTrace Object**

Message Mapping “Patterns”

- **Summarization**
- **Sequence Number Generation**
- **Duplicating Subtrees**

Multi–Mappings

Mapping Templates

Summary

Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions
- Function Usage
- Simple Function Examples
- Node Function Examples

Introduction to Most–Common Java Usage

User-defined Functions
- Usage

Advanced User–Defined Functions
- Usage
- ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”
- Summarization
- Sequence Number Generation
- Duplicating Subtrees

Multi–Mappings

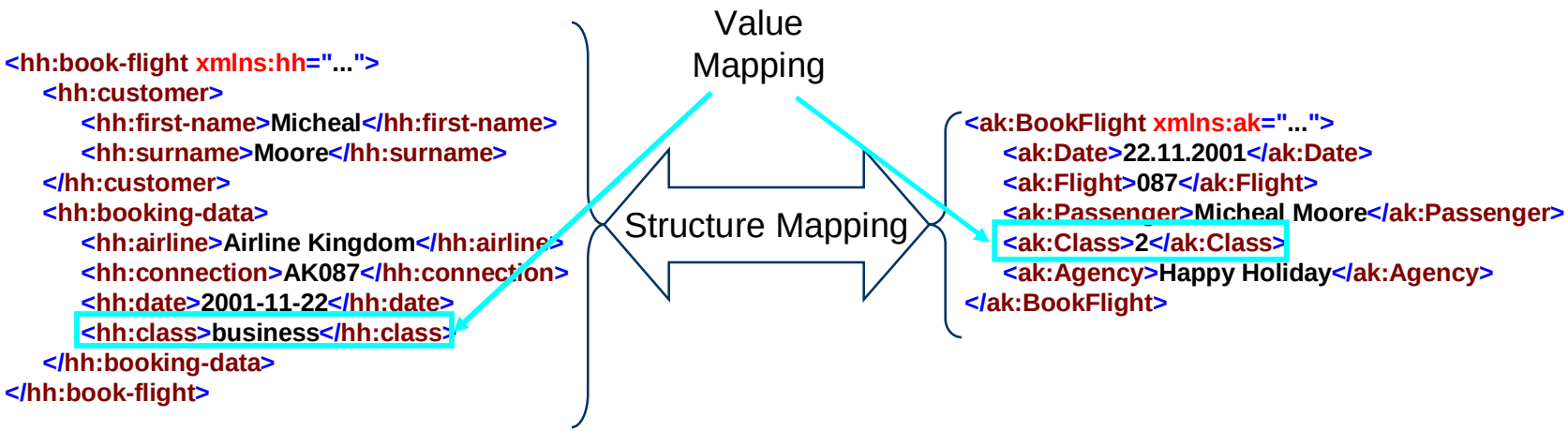
Mapping Templates

Summary

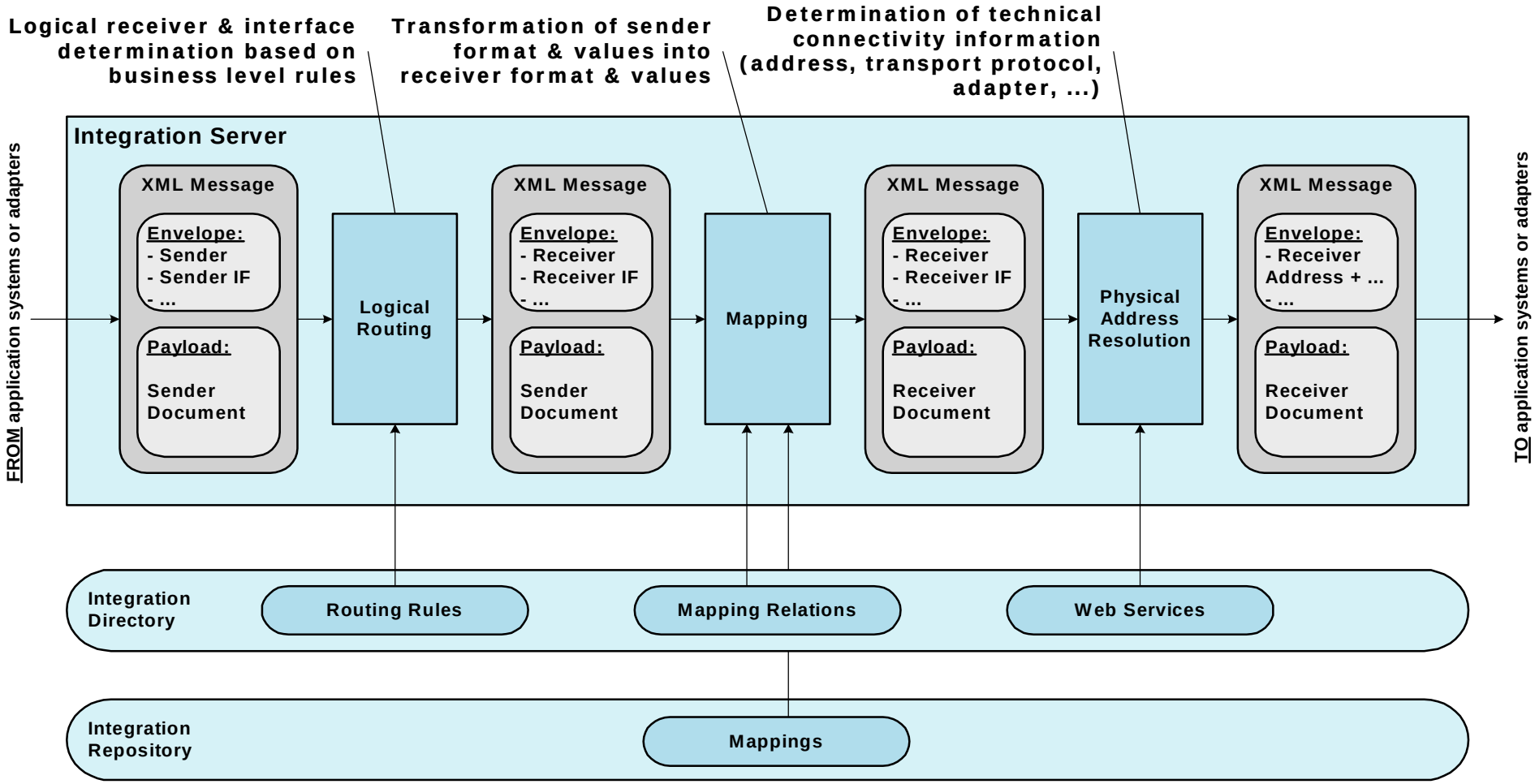
Concepts - General Mapping Types

Mapping

- Transformation from one message structure to another
- Transformation of one key value to another

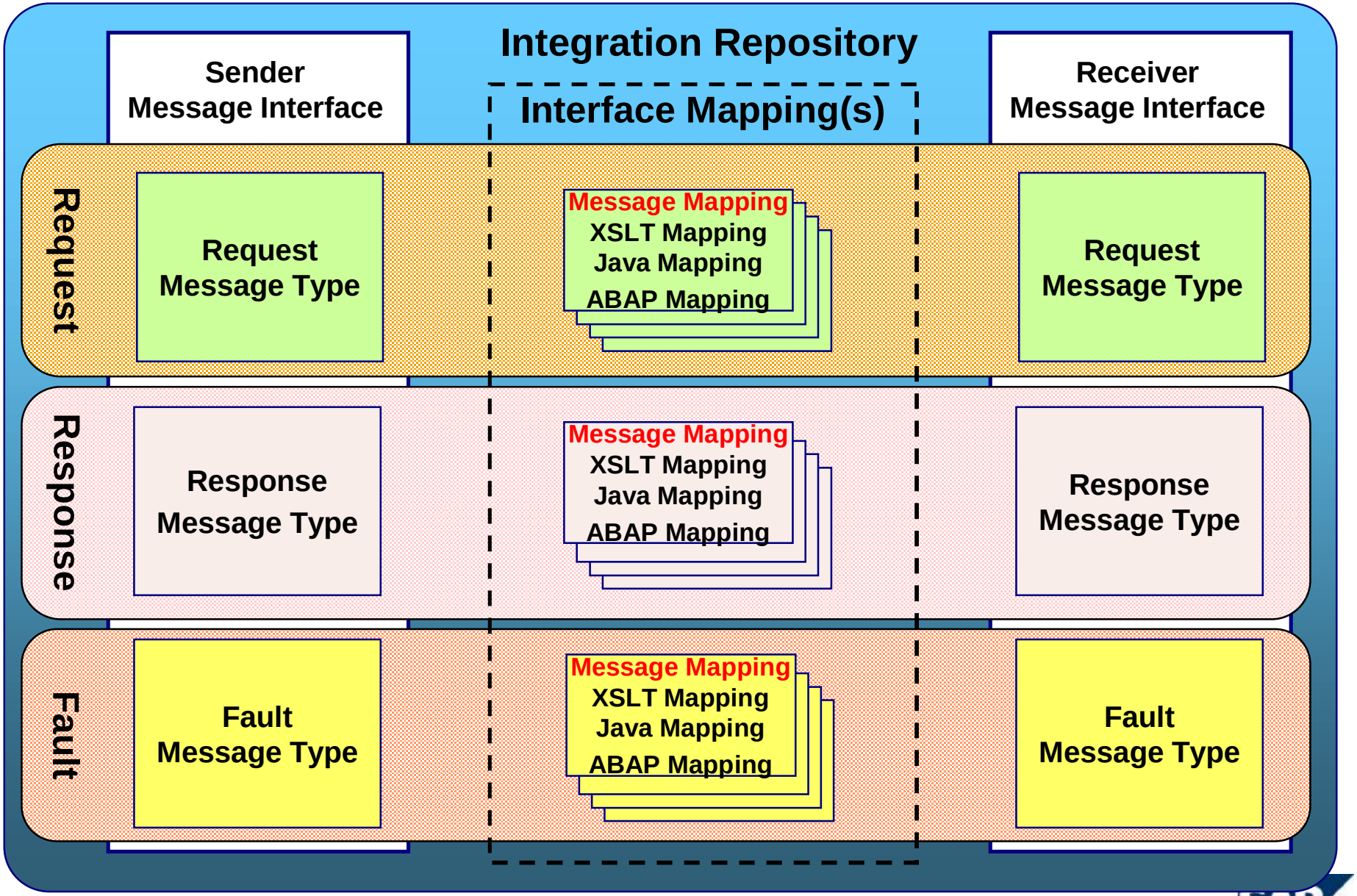


Stateless Processing Model – Logical View



Technical activities (sending, receiving, queuing,...) omitted

Concepts - Relate Mapping Programs to Interfaces



Concepts - Mapping in SAP XI

Message Mapping

- Designed by using the graphical mapping editor of Integration Builder
- Result: Generated Java Code.
- NOT memory–intensive, can “theoretically” handle GB–sized messages

Imported Archives

- Import externally defined mapping programs into repository
- Java mapping:
  Implemented by using a specific interface
- XSLT mapping:
  - Runtime supports XSLT processor
  - Java methods can be called from within a Style Sheet

ABAP Workbench

- ABAP Mappings in ABAP objects
- XSLT Mappings (ABAP Engine)

You can execute mapping programs in a sequence
(Only for a request or a response message)

Overview – Message Mapping

Use

- Design structure mapping between any 2 XML structures
- Connect to “value” mapping
- Generates java source code and jar files to be used during runtime.
- Can load the following into the structure overview:
  - A schema from the Integration Repository
    - Message Types
    - External Definitions
    - Imported Objects (RFCs and IDOCs)
  - XML or XSD files from a local file (*import* and *include* are ignored in XSD)

Overview – Message Mapping Functions

Standard Functions

- E.g. string processing, number/date formatting, boolean, …

Value Mapping

Node functions

User-defined functions

- Java method

Icon Status Display

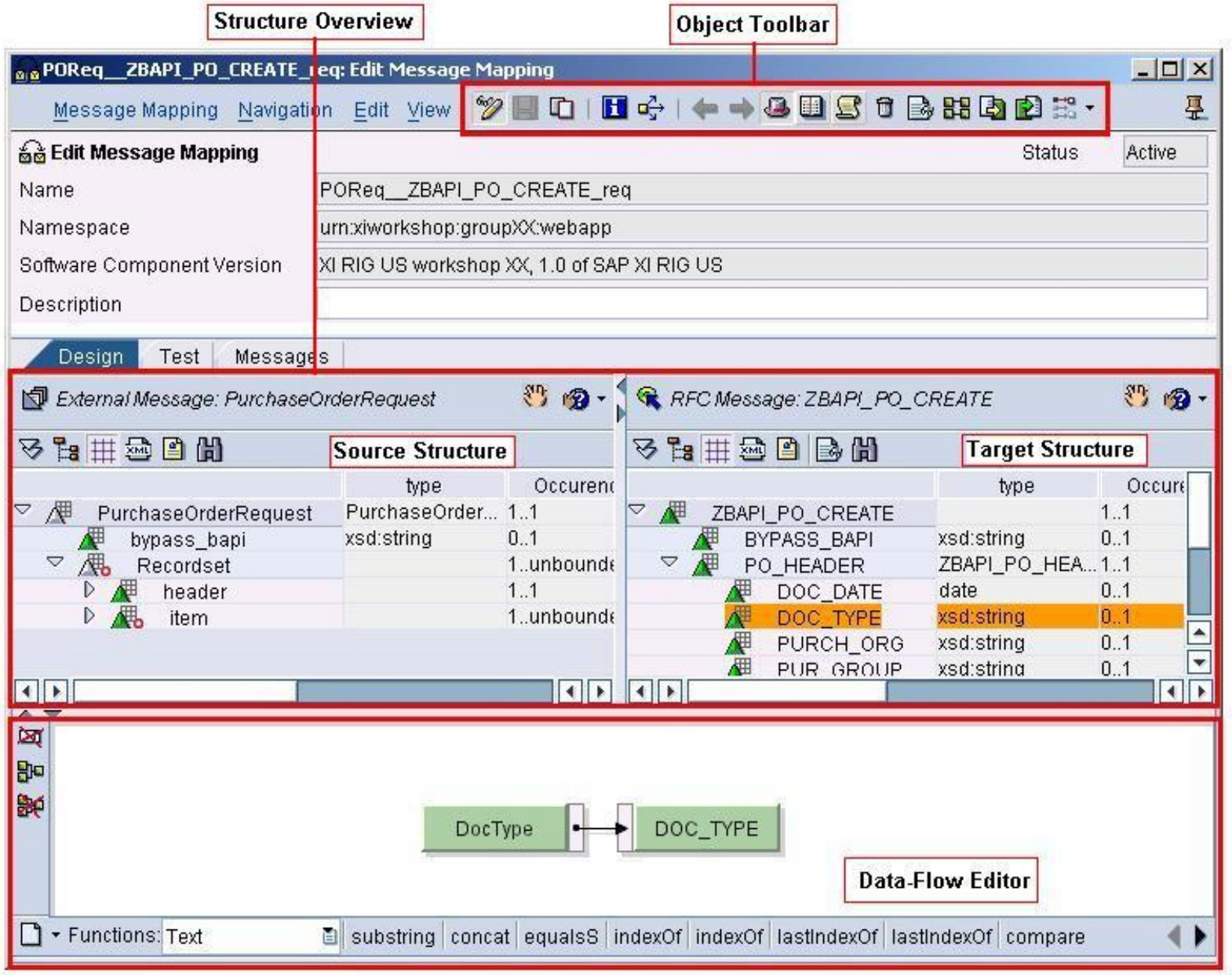
Icon Types

Icon	Meaning
	Attribute
	Element
	Element with more than 1 occurance

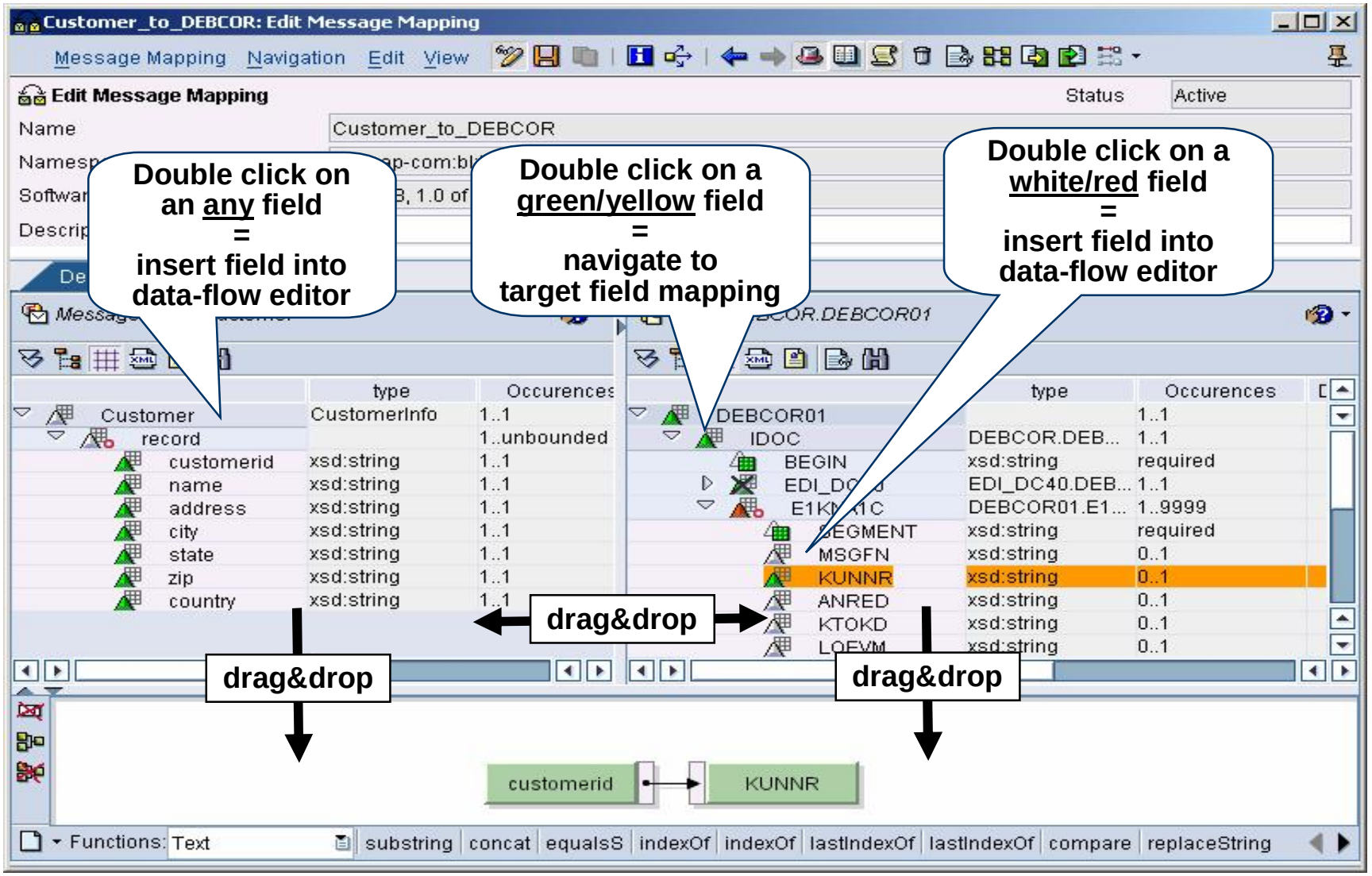
Icon Colors

Color	Meaning
White	Optional attribute or element not assigned
Red	Attribute or element must be assigned to complete the mapping
Yellow	Attribute or element has already been assigned but the corresponding mapping in the data-flow editor is not complete
Green	Mapping to target field complete

Overview –Mapping Editor



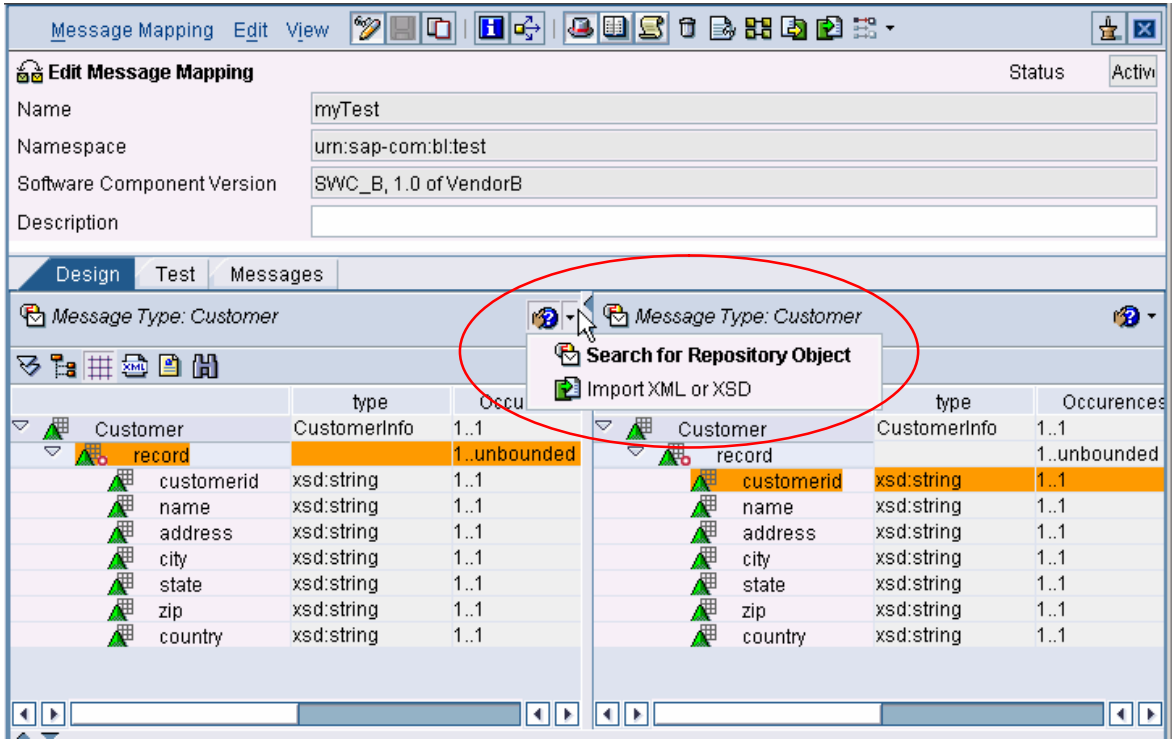
Overview – Assignment of Fields



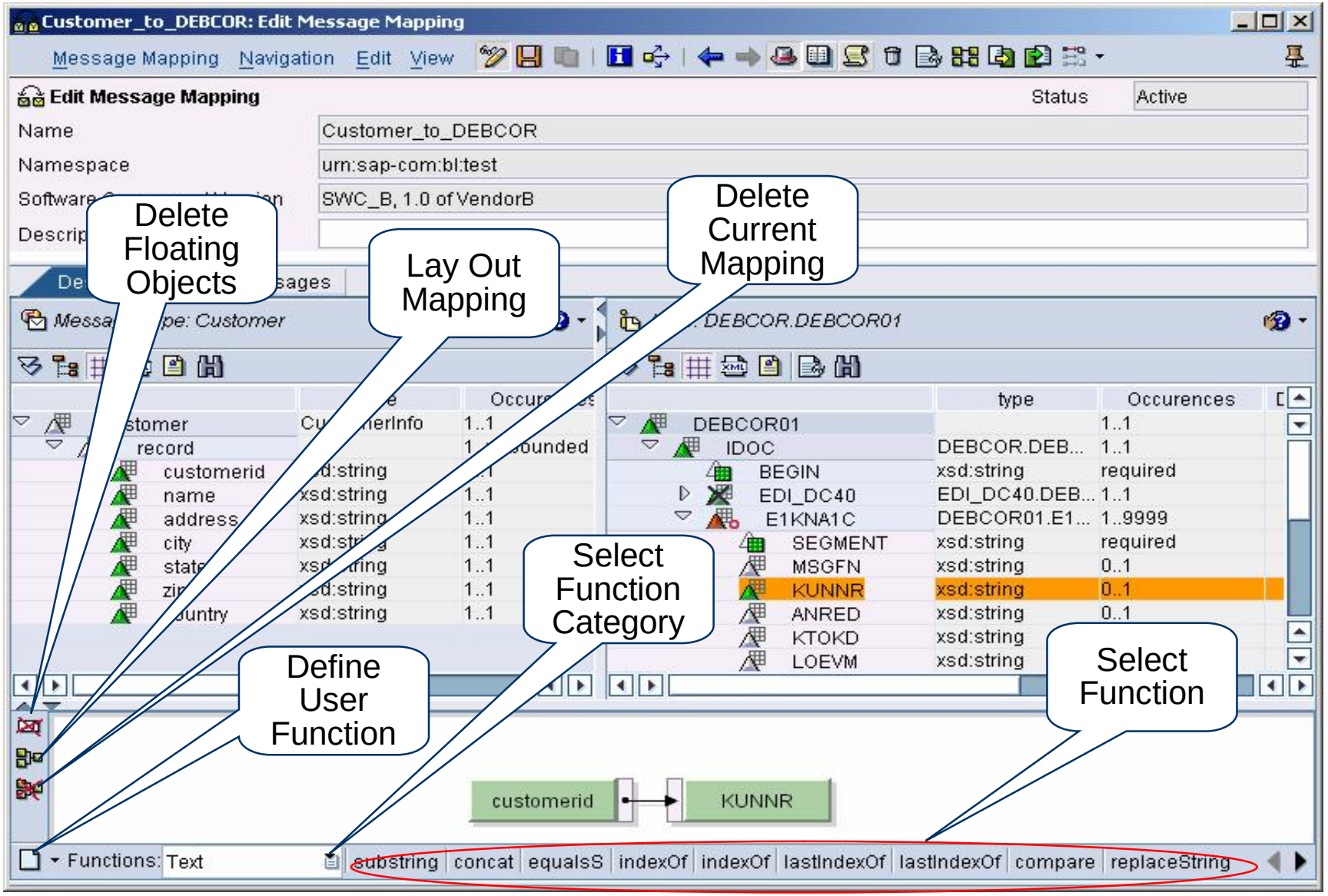
Overview – Assign Source/Target Message Types

There are 3 ways message types can be assigned to the Message Mapping.

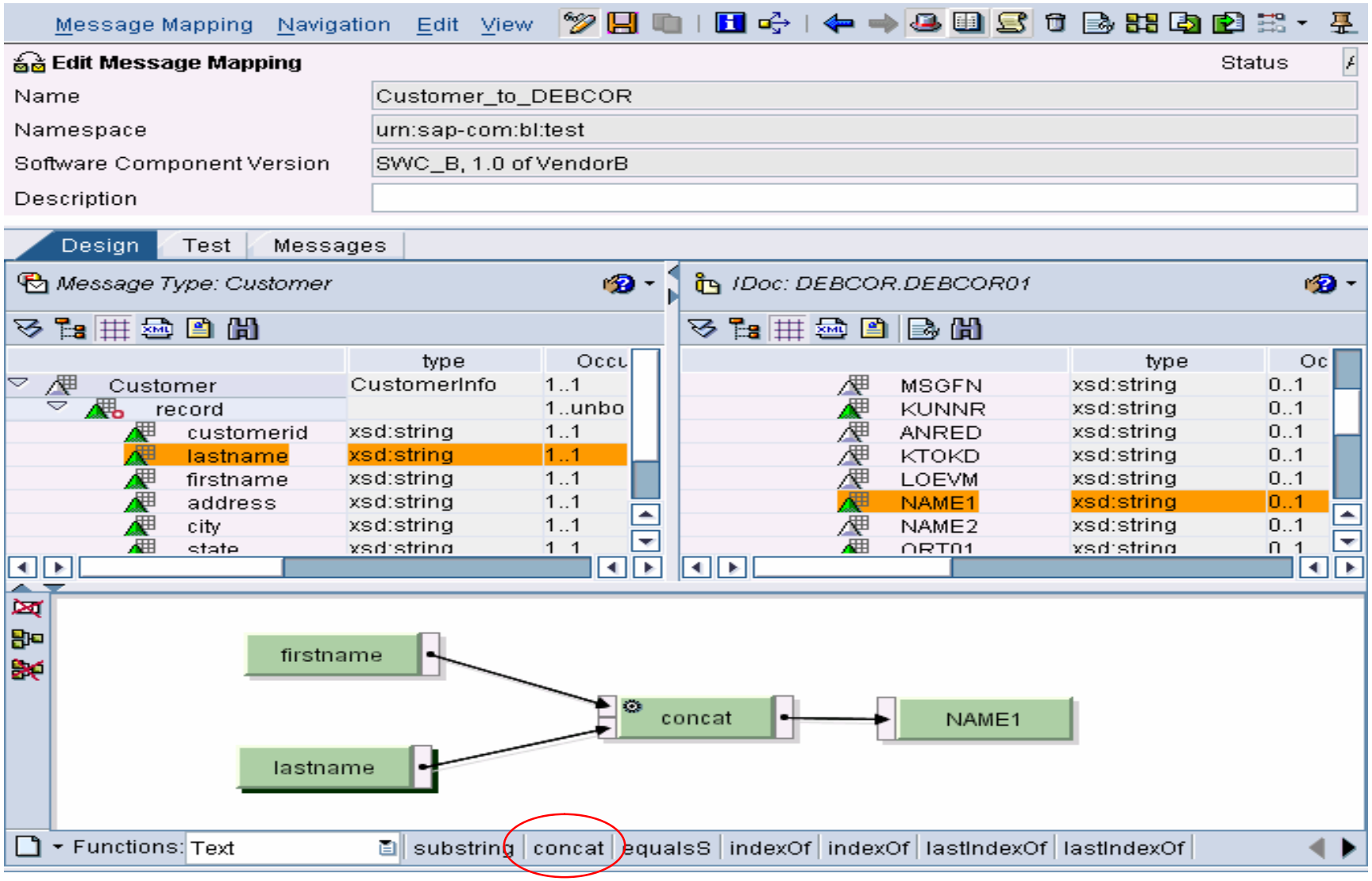
1. **By using an existing object already created in the Integration Repository (e.g. Message Type, External Definitions)**
2. **By using an RFC or IDoc meta data imported from SAP**
3. **By importing XML or XSD files from local file system**



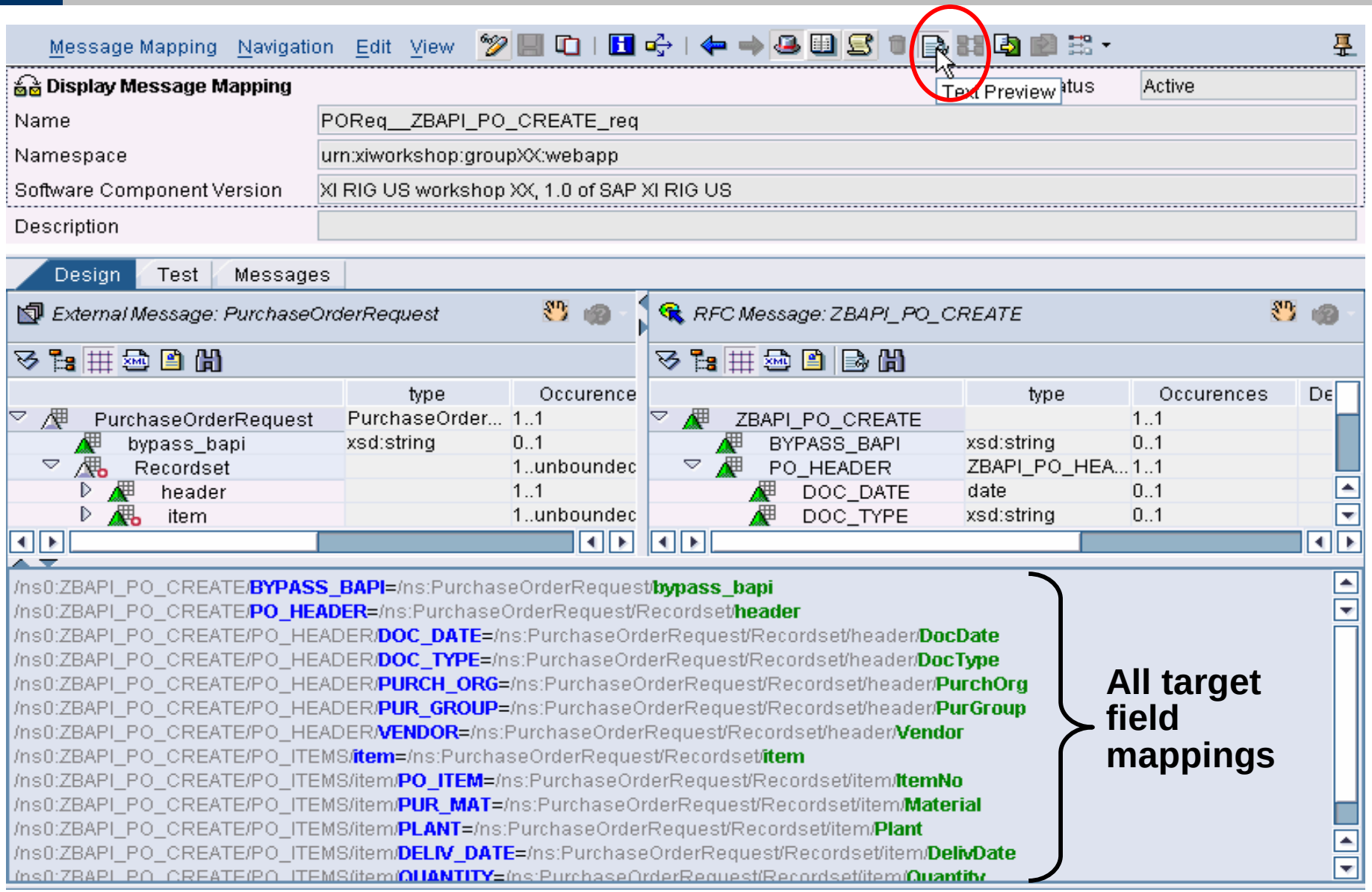
Overview – Data–Flow Editor



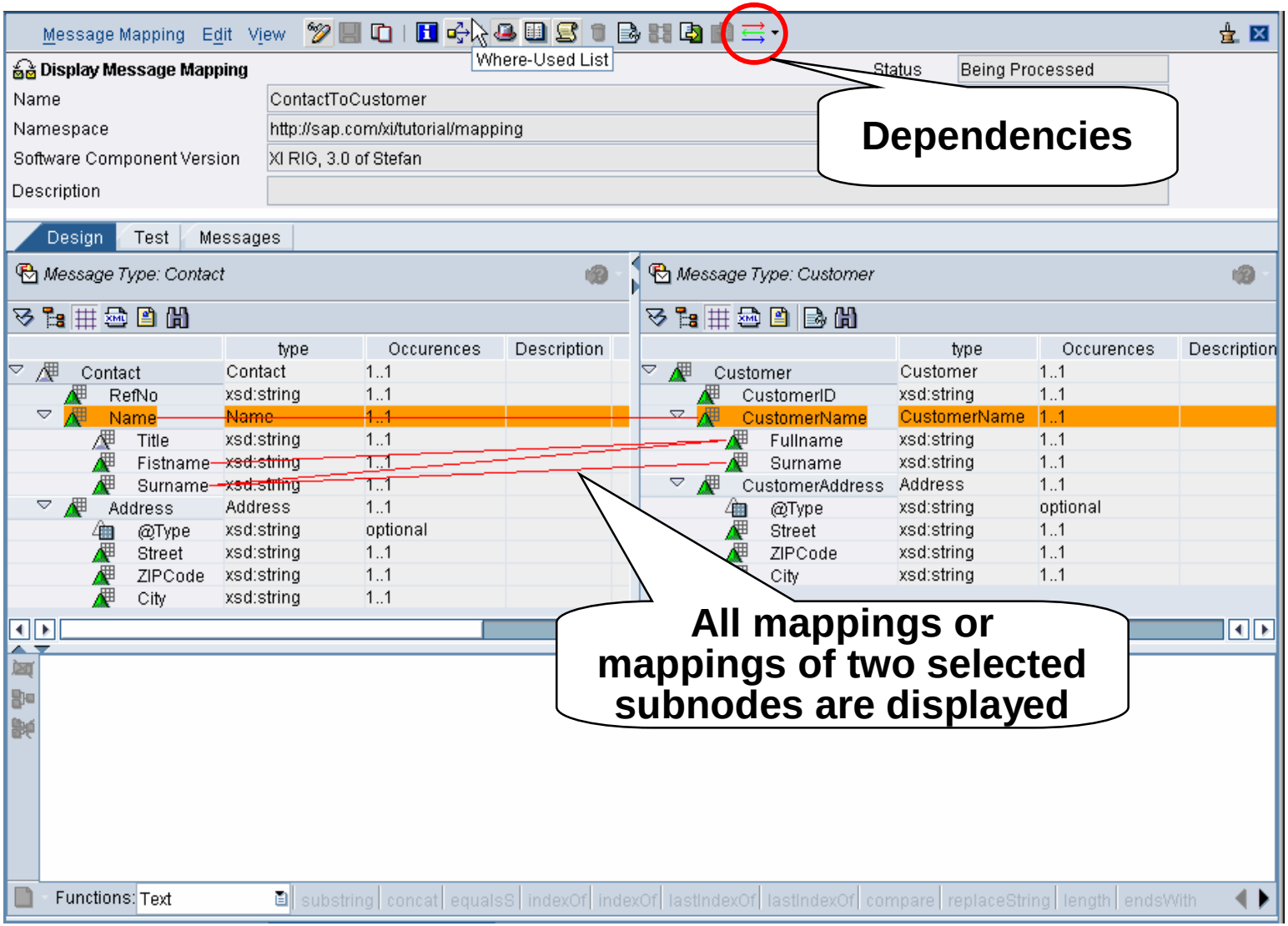
Overview – Data–Flow Editor – Example



Overview – Text Preview



Overview – Dependencies



Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions
- Function Usage
- Simple Function Examples
- Node Function Examples

Introduction to Most–Common Java Usage

User-defined Functions
- Usage

Advanced User–Defined Functions
- Usage
- ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”
- Summarization
- Duplicating Subtrees
- Table/Value Lookup (within source, not external)
- Tree–Reversal

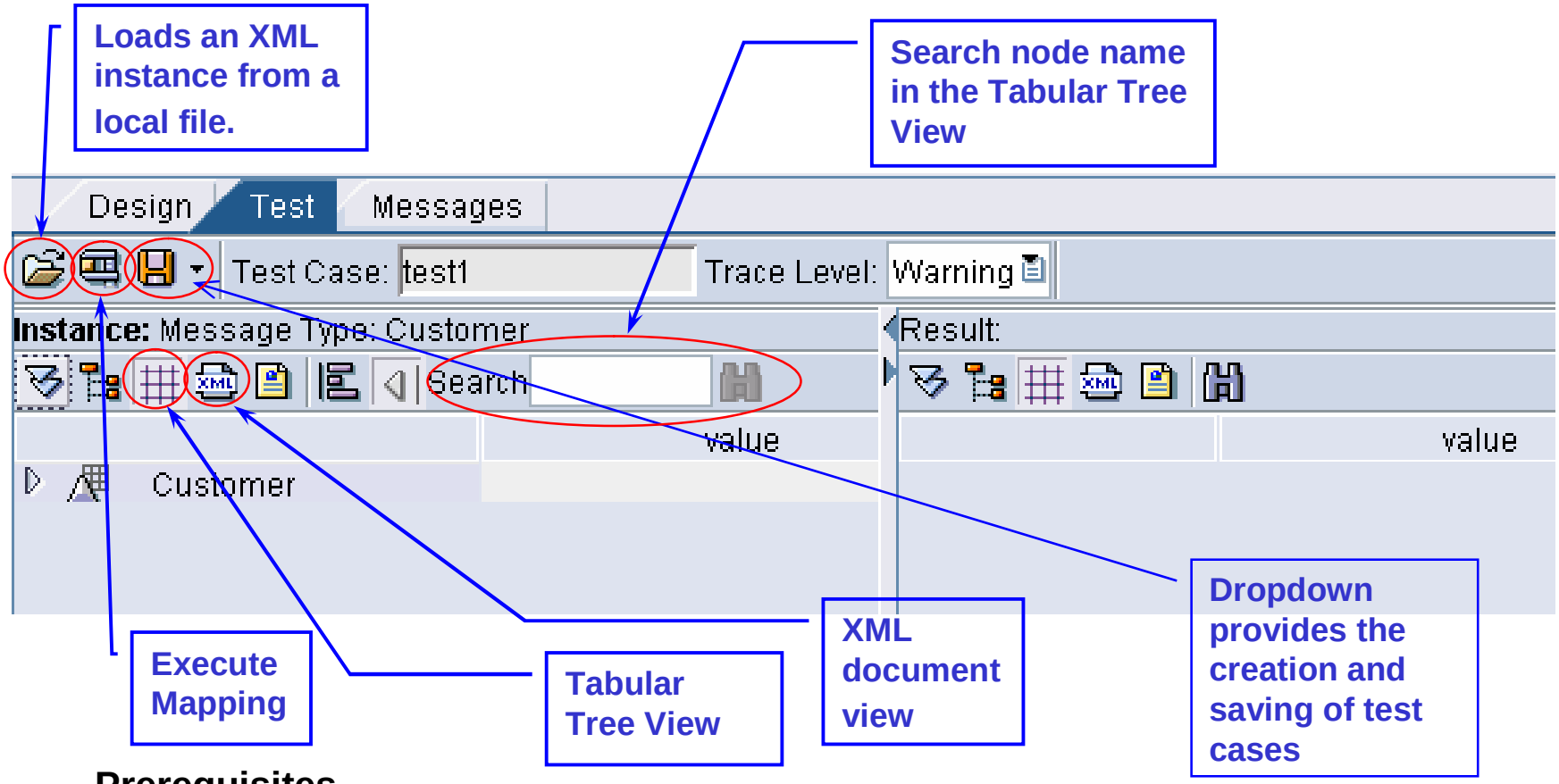
Multi–Mappings

Mapping Templates

Summary

Test/Debugging Environment

A completed Message Mapping can be tested by using the “Test” tab.

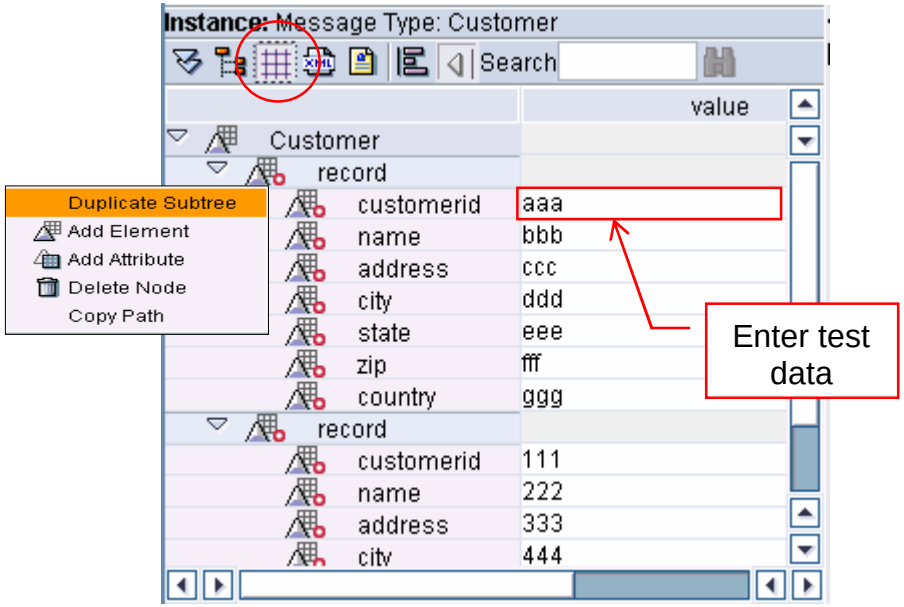


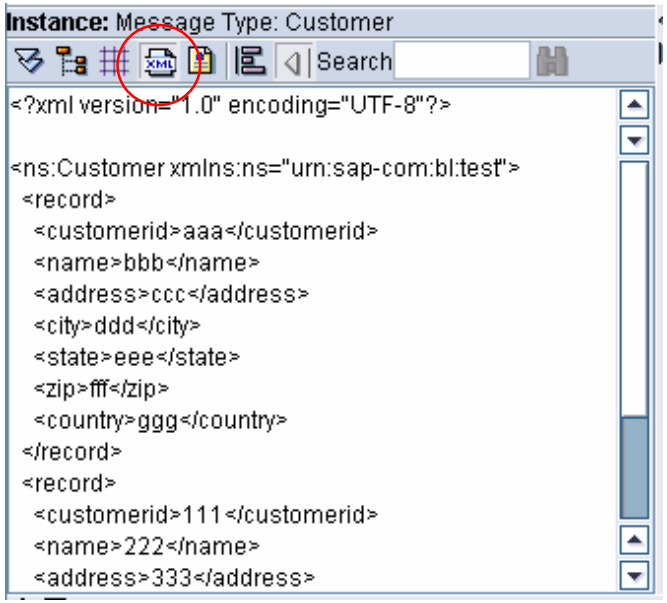
Prerequisites

1. The defined message mapping must be *complete.* This means that each mandatory target field must be assigned to one or more source fields.
2. There must not be any unassigned arguments when using functions.

Test/Debugging Environment

Editing the XML instances to use in test cases:



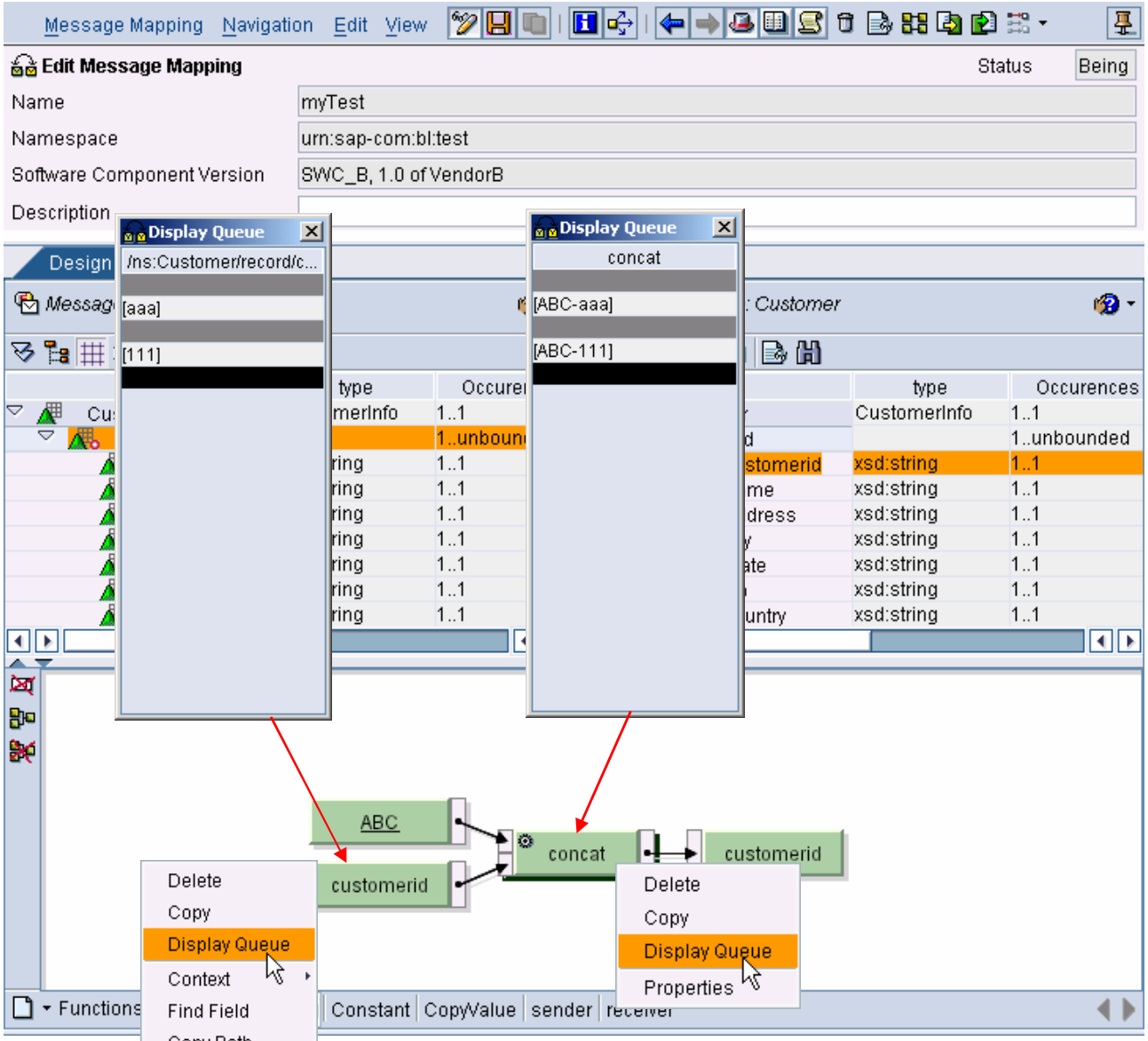


Using the context menu in the (*Tabular*) *Tree View*, you can copy sub-trees, delete nodes, and add elements and attributes. In the *Value* column in the tabular tree view you can also enter values for fields.

Using the editor for the XML view, you can manually edit elements and attributes or their values here. Use *CTRL C*, *CTRL X*, and *CTRL V* respectively to copy, cut, and paste parts of an XML instance.

Test/Debugging Environment

Using the “Display Queue” to examine each step of the mapping:



Note:
The “context” display and usage in the queues will be discussed in later slides.

Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions

- **Function Usage**
- **Simple Function Examples**
- **Node Function Examples**

Introduction to Most–Common Java Usage

User-defined Functions

- Usage

Advanced User–Defined Functions

- Usage
- ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”

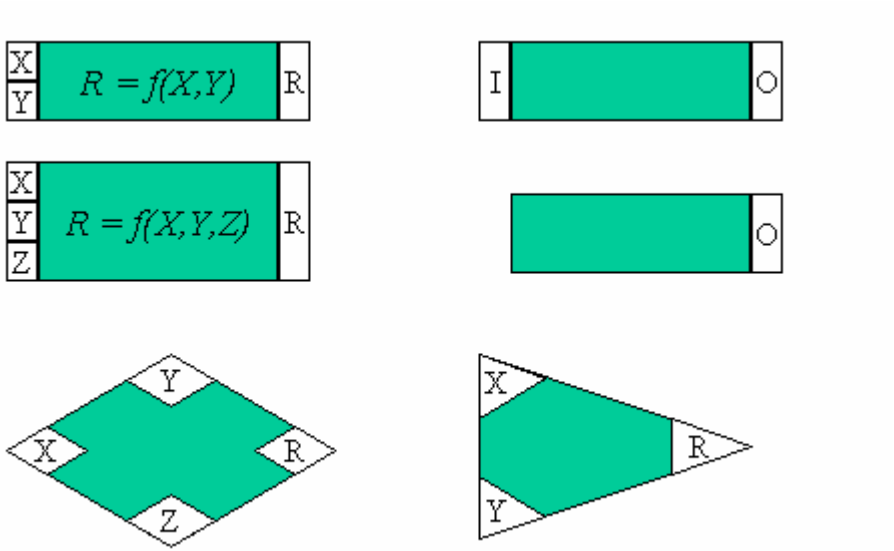
- Summarization
- Duplicating Subtrees
- Table/Value Lookup (within source, not external)
- Tree–Reversal

Multi–Mappings

Mapping Templates

Summary

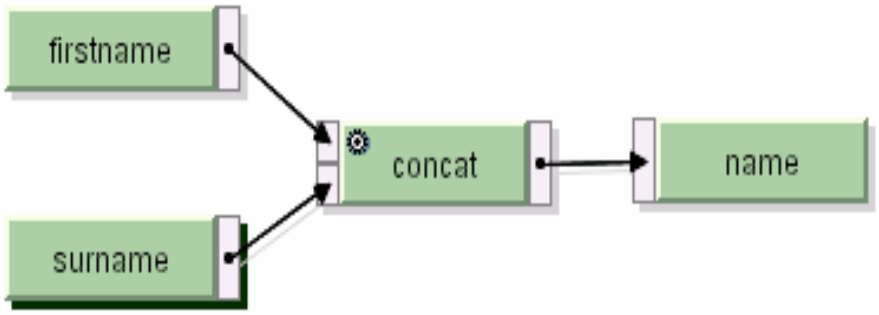
Standard Functions: Introduction



- Most data-flow objects have two or three inbound channels on the left-hand side, and one outbound channel on the right-hand side.
- If functions are shaped as rhombuses or triangles.
- There are also conversion functions with one inbound and outbound channel and functions that get a value without making any entries.
- Standard functions that require additional specifications are indicated with a star ( ). To call the dialog for *Function Properties*, double click the data-flow object.

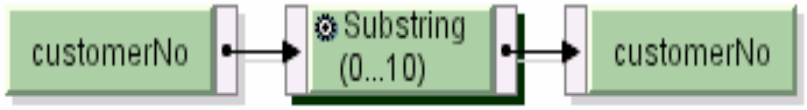
Simple Standard Functions: concat

Function category: Text

Source Message	Mapping	Target Message
<?xml version="1.0" encoding="UTF-8" ?> <Customer> <firstname> Harry </firstname> <surname> Potter </surname> </Customer>	 **concat properties** Delimiter=" "	<?xml version="1.0" encoding="UTF-8" ?> <Contact> <name> Harry Potter </name> </Contact>

Simple Standard Functions: substring

Function category: Text

Source Message	Mapping	Target Message
<?xml version="1.0" encoding="UTF-8" ?> <Customer> <customerID> 0123456789-181170 </customerID> </Customer>	 **substring properties** start from: 0 count: 10	<?xml version="1.0" encoding="UTF-8" ?> <Contact> <customerNo> 0123456789 </customerNo> </Contact>

Position count for all text functions start at 0!

Simple Standard Functions: DateTrans

Function category: Date

Source Message	Mapping	Target Message
<?xml version="1.0" encoding="UTF-8" ?> <Customer> <custBirthday> 181170 </custBirthday> </Customer>	 **DateTrans properties** src Format: ddMMyy dst Format: MM/dd/yyyy	<?xml version="1.0" encoding="UTF-8" ?> <Contact> <birthday> 11/18/1970 </birthday> </Contact>

- uses standard Java date template

Function Categories

Arithmetic: add, substract, equalsA, abs, sqrt, sign, sqrt, etc.

Boolean: And, Or, Not, Equals, notEquals, if, ifWithoutElse

Constants: Constant, CopyValue, sender, receiver

Conversions: FixValues, Value mapping

Date: currentDate, DateTrans, DateBefore, DateAfter, CompareDate

Node Functions: createIf, removeContexts, replaceValue, exists, SplitByValue, collapseContexts

Statistic: sum, average, count

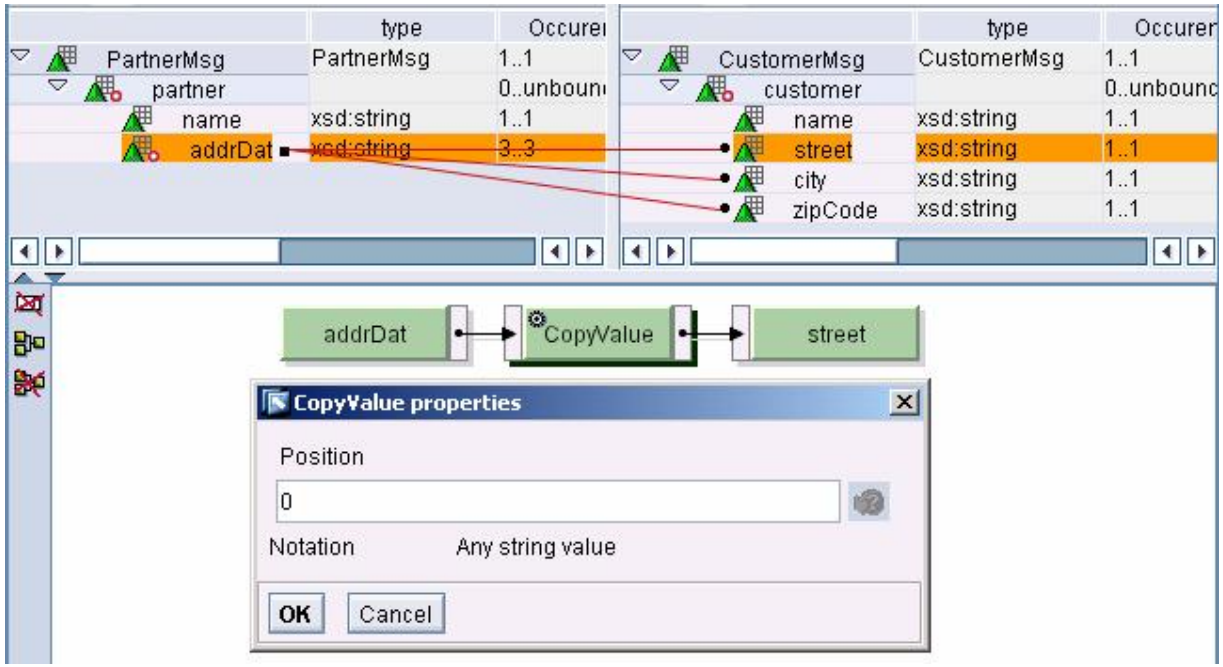
Text: substring, concat, equalsS, indexOf, lastIndexOf, compare, etc.

(The functions marked in blue will be explored in detail.)

Constant Function – CopyValue

Use CopyValue() to copy a repeating value's position in the source structure and assign it to a target field.

Example:



Result:

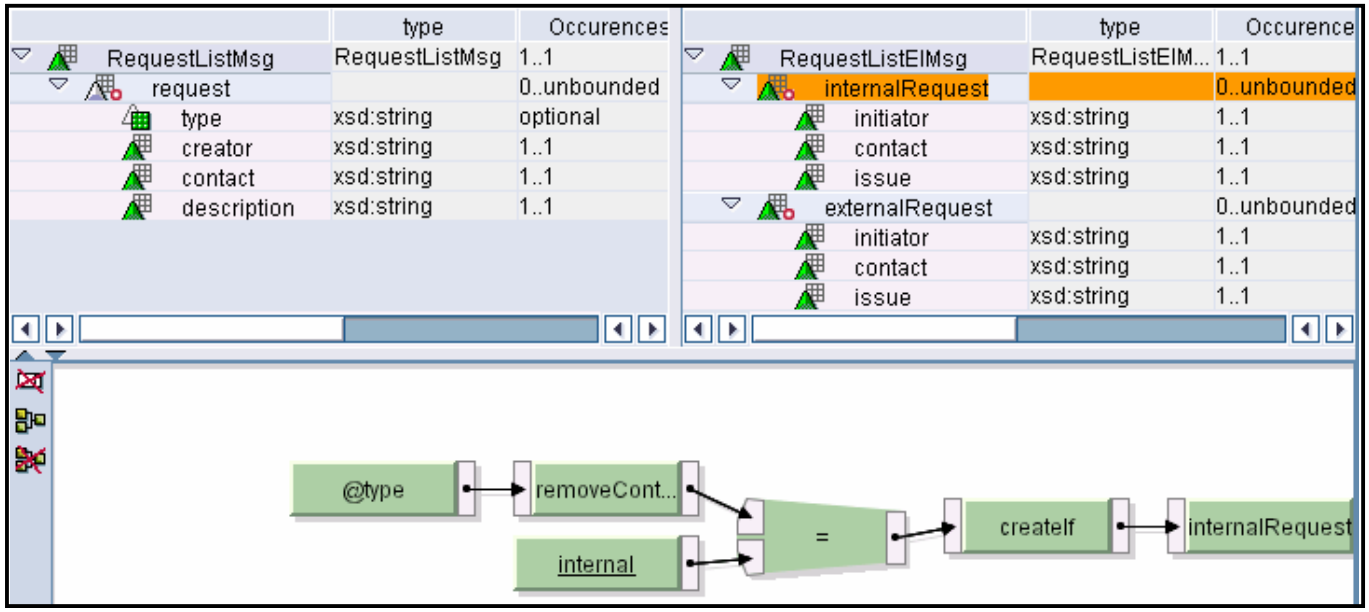
	value
PartnerMsg	
partner	
name	Joe
addrDat	123 Main Street
addrDat	Smallville
addrDat	12345

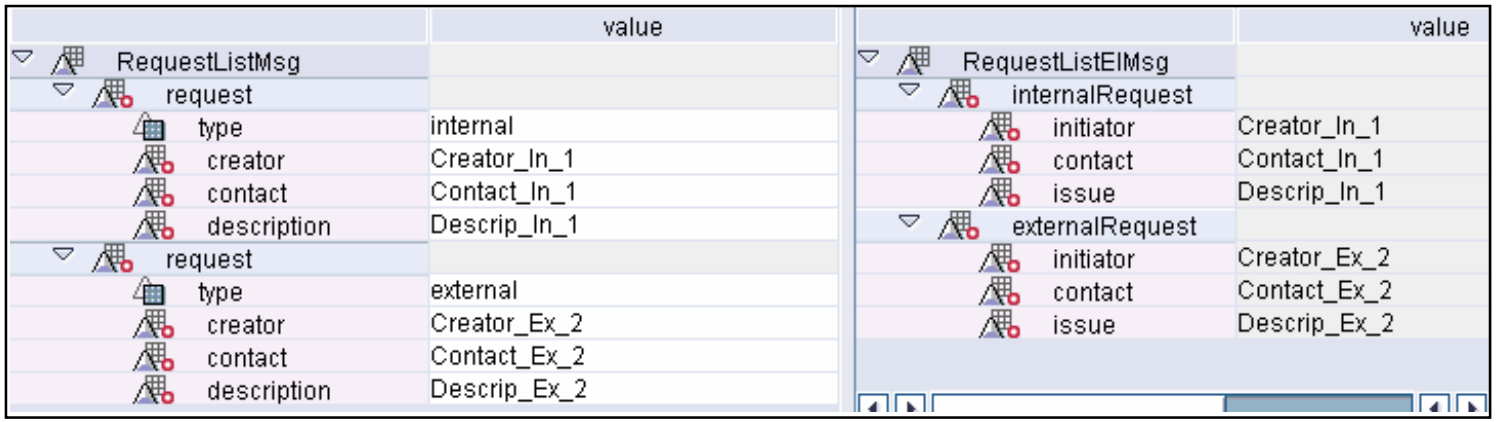
	value
CustomerMsg	
customer	
name	Joe
street	123 Main Street
city	Smallville
zipCode	12345

Node Function – createIf

Use createIf() to create a tag in the target structure using a condition.

Example: Depending on the value of type (internal or external), the target tag either internalRequest or externalRequest will be created.

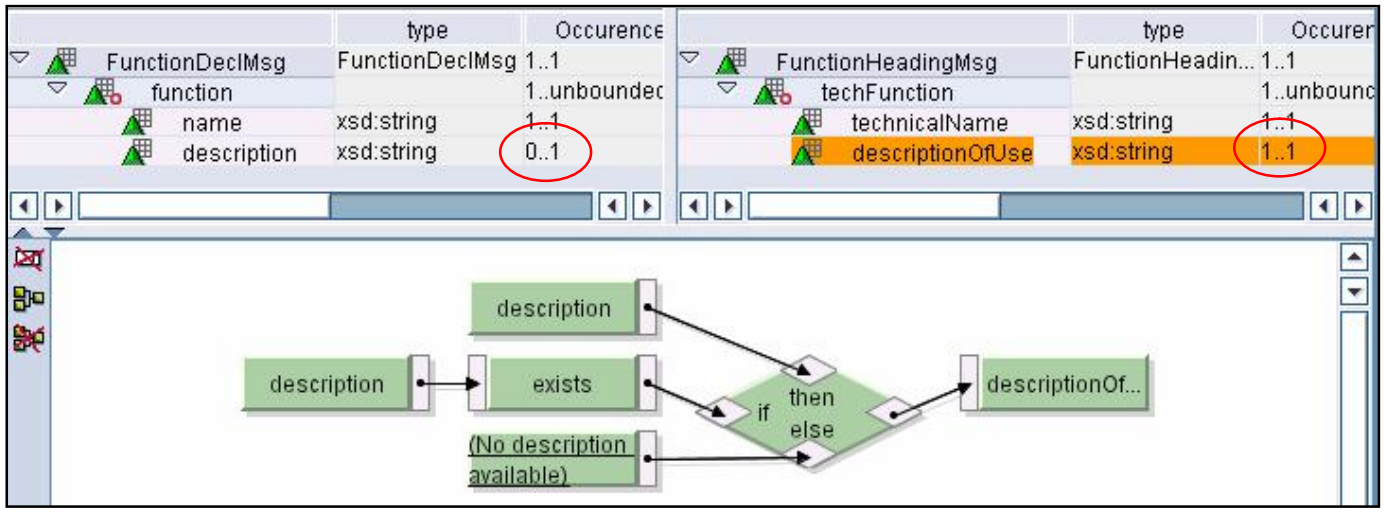




Node Function– exists

Use exists() to determine whether a particular source field exists in the XML instance to be processed. If does, exists() returns *true*, otherwise, *false.*

Example: In the source, description is optional. However, in the target, description is mandatory.



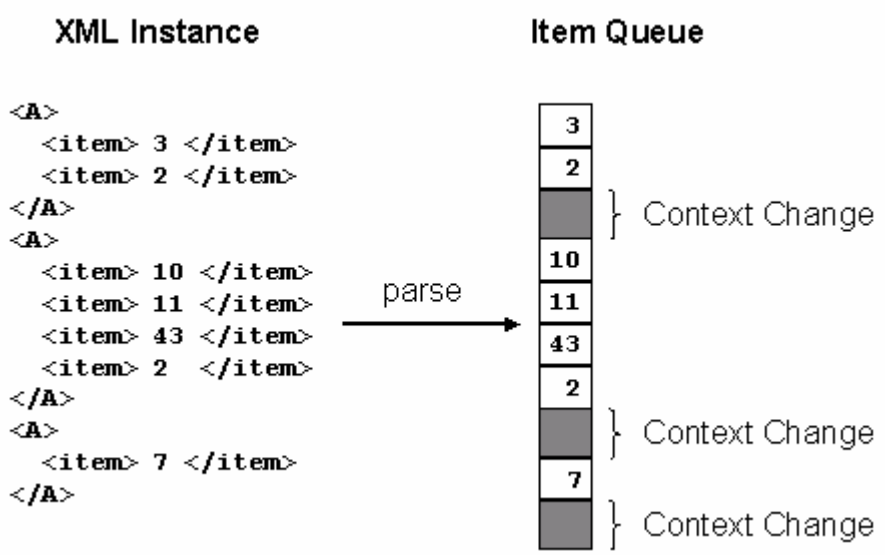
	value
FunctionDeclMsg	
function	
name	name1
function	
name	name2
description	description2

	value
FunctionHeadingMsg	
techFunction	
technicalName	name1
descriptionOfUse	(No description available)
techFunction	
technicalName	name2
descriptionOfUse	description2

Node Functions – Contexts

Context Change

- Message mapping works internally by using queues
- If no further elements are imported at a particular hierarchy level, a *Context Change* is inserted in the queue
- Use node functions to handle changes in the message hierarchy.



`removeContexts`

- deletes all context changes of a queue

`SplitByValue`

- insert additional context changes in a queue

Node Functions – removeContexts (I)

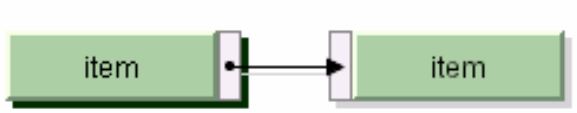
With *removeContexts*

Source Message	Mapping	Target Message
<?xml version="1.0" encoding="UTF-8" ?> <Test_Out_Remove> <header name="A"> <item>A.one</item> <item>A.two</item> <item>A.three</item> </header> <header name="B"> <item>B.one</item> <item>B.two</item> </header> </Test_Out_Remove>		<?xml version="1.0" encoding="UTF-8" ?> <Test_In_Remove> <item>A.one</item> <item>A.two</item> <item>A.three</item> <item>B.one</item> <item>B.two</item> </Test_In_Remove

§use `removeContext()` to delete the parent context of an element.

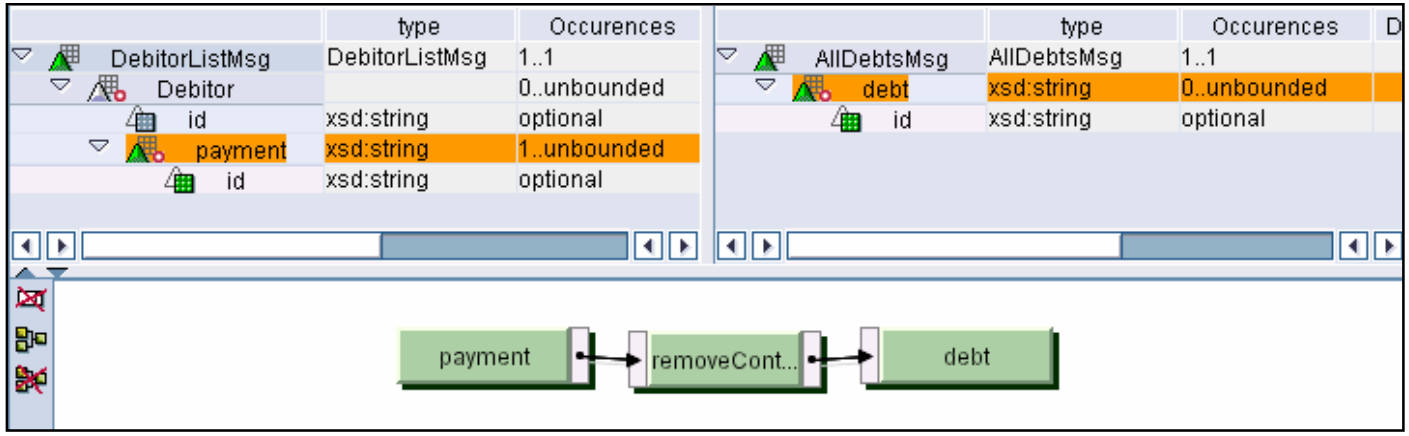
Node Functions – removeContexts (II)

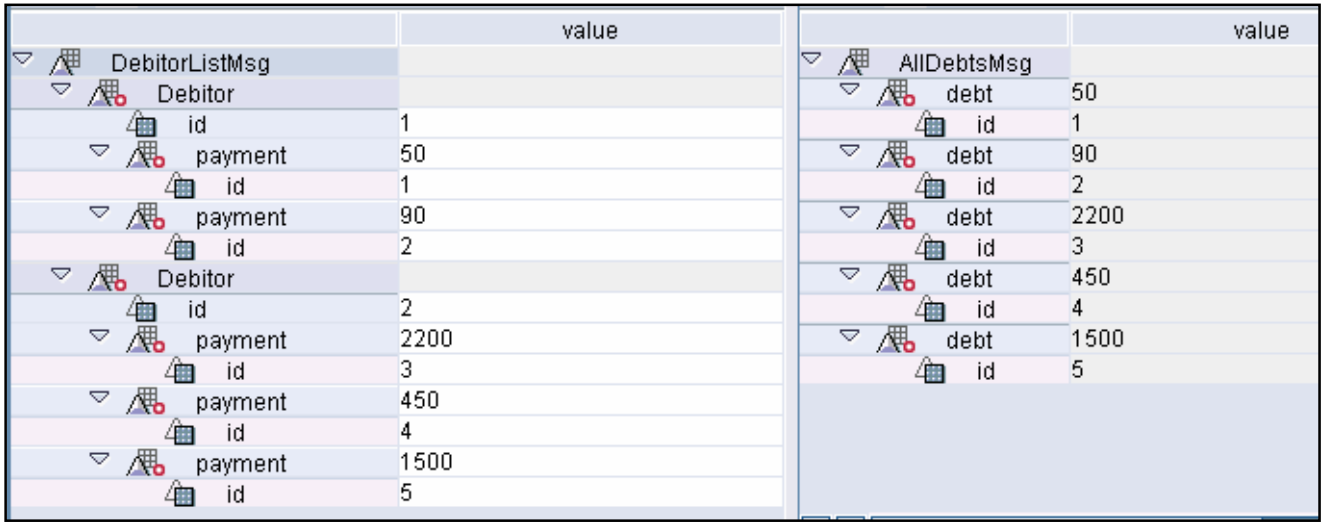
Without *removeContexts*

Source Message	Mapping	Target Message
<?xml version="1.0" encoding="UTF-8" ?> <Test_Out_Remove> <header name="A"> <item>A.one</item> <item>A.two</item> <item>A.three</item> </header> <header name="B"> <item>B.one</item> <item>B.two</item> </header> </Test_Out_Remove>	 	<?xml version="1.0" encoding="UTF-8" ?> <Test_In_Remove> <item>A.one</item> <item>A.two</item> <item>A.three</item> </Test_In_Remove>

Node Function – removeContext

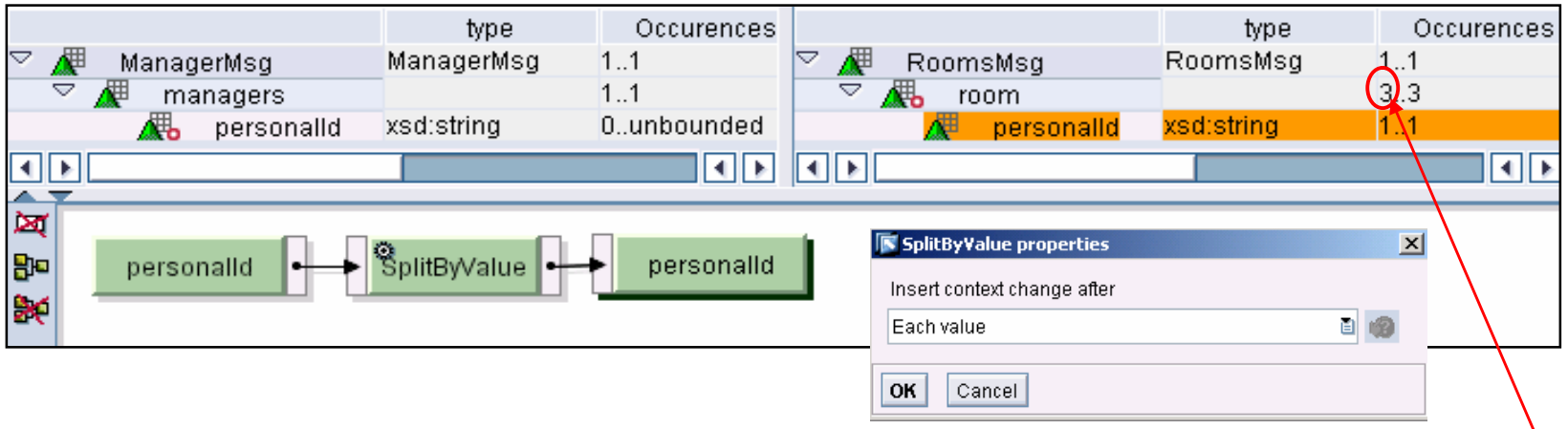
Use removeContext() to delete all the higher contexts for an element. This deletes all higher hierarchy levels, so that all elements of the target queue are assigned to a root element of the source queue.



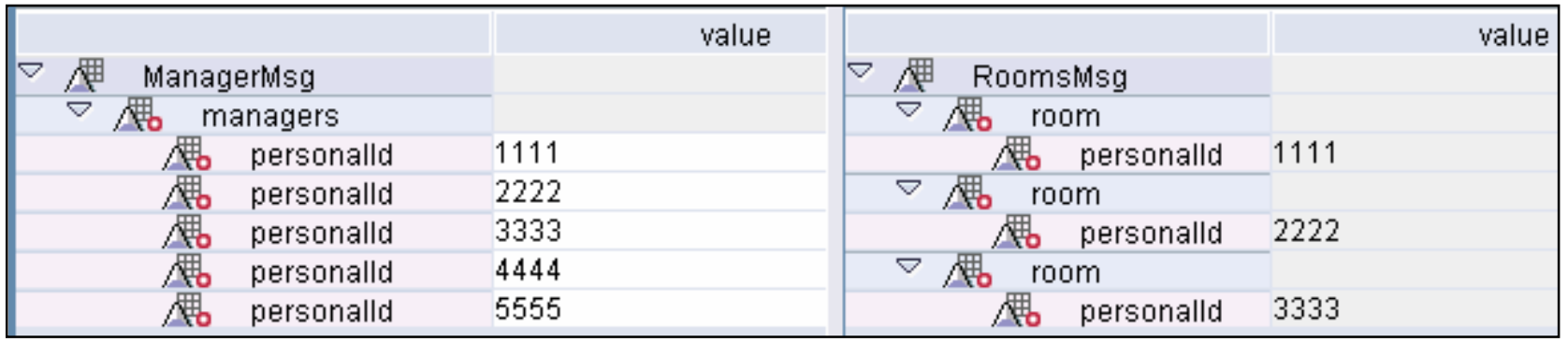


Node Function– SplitByValue

SplitByValue() is the counterpart to removeContexts(): Instead of deleting a context, you can insert a context change in the source value queue.

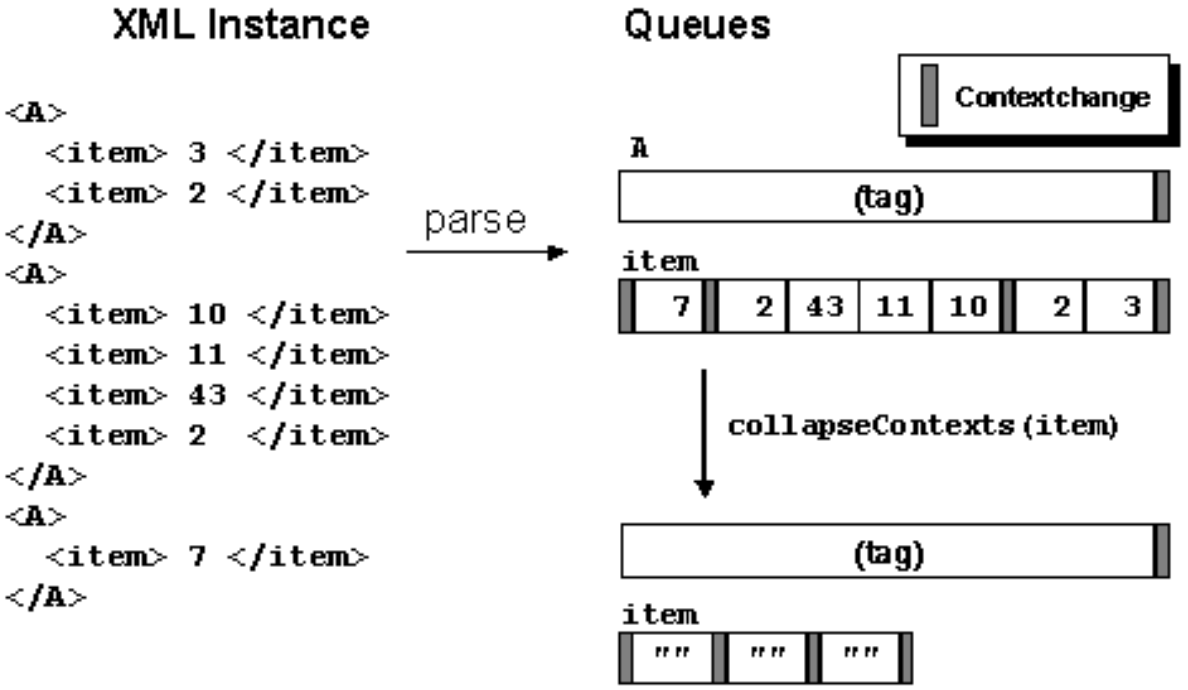


Only this number of elements will be created. If the number in the source is less, error exception will occur.



Node Function – collapseContexts

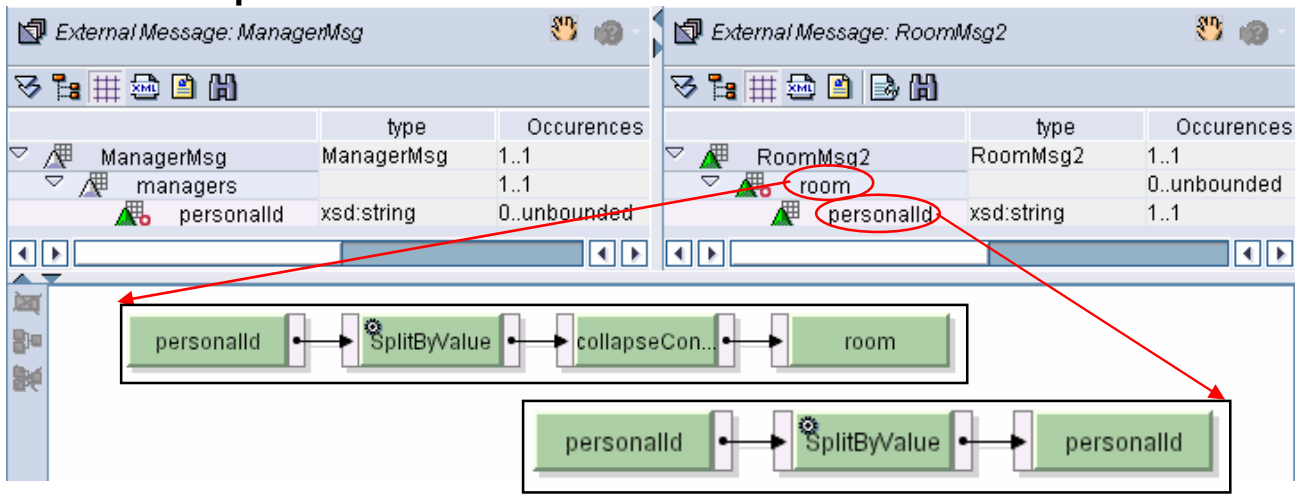
Using collapseContexts() replaces all values of all contexts with an empty string:



This can be useful if each time an upper-level node is to be created if a lower-level node exists, for example. The function is useful in combination with the function SplitByValue().

Node Function – collapseContexts (cont.)

To avoid previous example's limitation of just using SplitByValue(), when exact number of elements had to be provided.



	value
ManagerMsg	
managers	
personalId	1111

	value
RoomsMsg2	
room	
personalId	1111

Example when source has only 1 element.

	value
ManagerMsg	
managers	
personalId	1111
personalId	2222
personalId	3333
personalId	4444
personalId	5555

	value
RoomsMsg2	
room	
personalId	1111
room	
personalId	2222
room	
personalId	3333
room	
personalId	4444
room	
personalId	5555

Example when source has multiple elements.

Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions
- Function Usage
- Simple Function Examples
- Node Function Examples

Introduction to Most–Common Java Usage

User-defined Functions
- Usage

Advanced User–Defined Functions
- Usage
- ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”
- Summarization
- Duplicating Subtrees
- Table/Value Lookup (within source, not external)
- Tree–Reversal

Multi–Mappings

Mapping Templates

Summary

Intro to Java Usage

Using Java in user functions will extend the capability of Message Mapping and, many time, will be necessary.

To effectively use Java in Message Mapping:

- Do not need to be an expert Java programmer
- Should be familiar with classes in packages:
  - java.lang.* (e.g. String)
  - java.util.* (e.g. Vector, Collections)
- Should be familiar with String and array operations:
  - length() and length
  - compare (do not use ==, instead, use equals())
- Should be familiar with for…next and while loops.

Intro to Java Usage – String

String

- All parameters passed in user functions are as String or array of String:
  - Single String variable is passed in simple user function
  - String array is passed in advanced user function
- String has the following useful methods:
  - substring
  - concat
  - equals and equalsIgnoreCase
  - length
- String comparisons:
  - DO NOT use:

```
String a;
if (a == “PRODUCT”) …
```

  - DO use:

```
String a;
if (a.equals(“PRODUCT”)) …
or
if (a.equalsIgnoreCase(“PRODUCT”)) …
```

Intro to Java Usage – arrays

Arrays

- Parameters in advanced user functions are passed as arrays.
- Most common processing of arrays is the use of *for…next*:

```
String[ ] a;
for (int i=0; i<a.length; i++) {
    if (a[i].equals(“PRODUCT”) …
    …..
}
```

Integer

- Use Integer class to convert integer to String

```
int i = 100;
String a = Integer.toString(i);
```

Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions
- Function Usage
- Simple Function Examples
- Node Function Examples

Introduction to Most–Common Java Usage

User-defined Functions
- **Usage**

Advanced User–Defined Functions
- Usage
- ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”
- Summarization
- Duplicating Subtrees
- Table/Value Lookup (within source, not external)
- Tree–Reversal

Multi–Mappings

Mapping Templates

Summary

User-Defined Functions – Overview

User–Define Functions:

- Functional enhancements: if standard functions do not fulfill requirements
- Is only visible in the message mapping in which you created it
- Can use Java programs from imported archives of the same software component version
- Use just like standard functions

User-Defined Functions – Overview

User–Defined Function types:

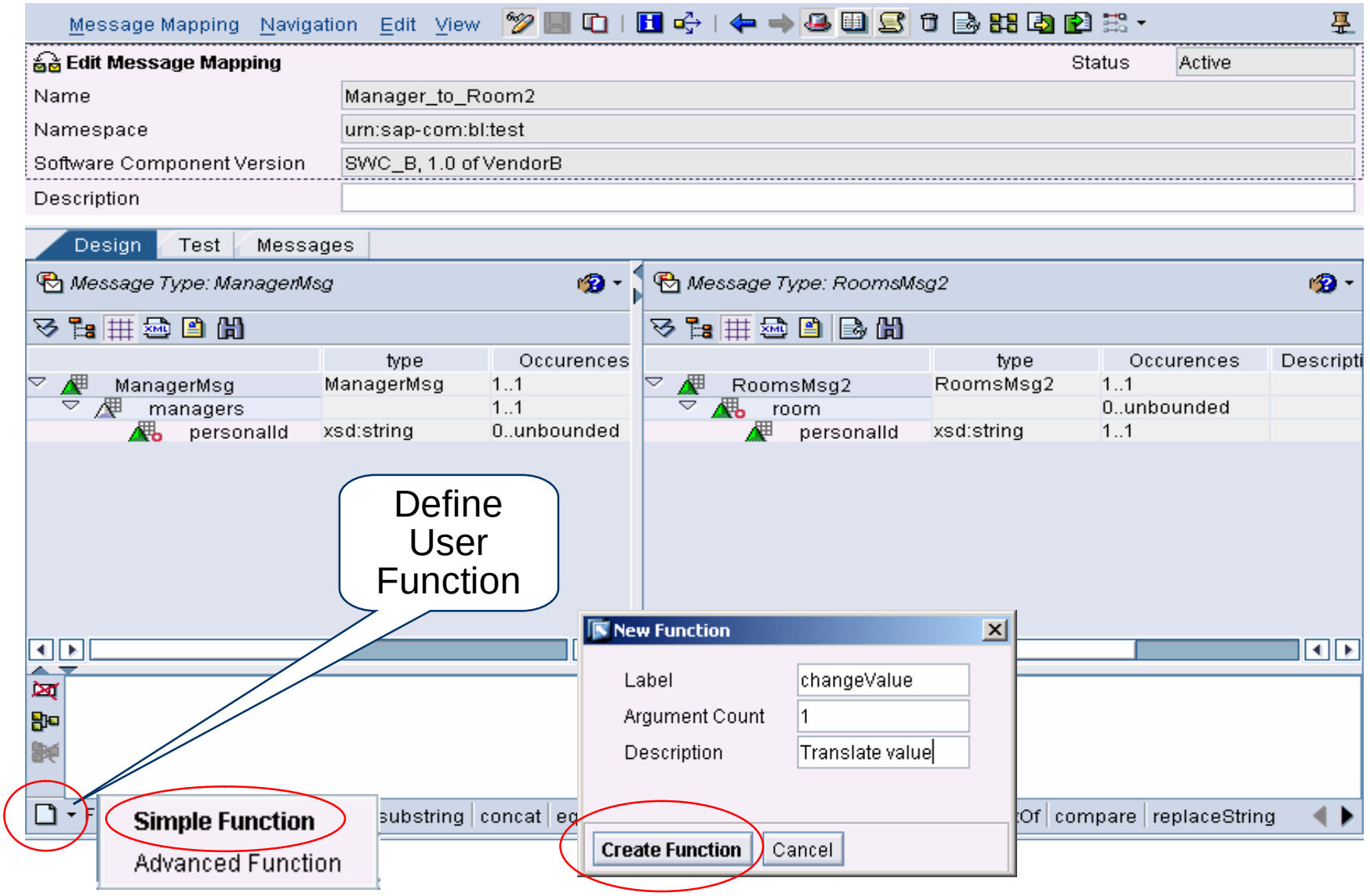
- *Simple functions*, which can process individual field input values for each function call. Simple functions, therefore, expect strings as input values and return a string.
- *Advanced functions*, which can process non–single string field input values (or arrays) for each function call. You can pass either all field values of a context or the whole queue for the field in an array when calling the function. Each input field is passed as an array of String. Returned values are stored in a String array, ResultList.

User-Defined Functions – Overview

Objects available to Simple and Advanced Functions

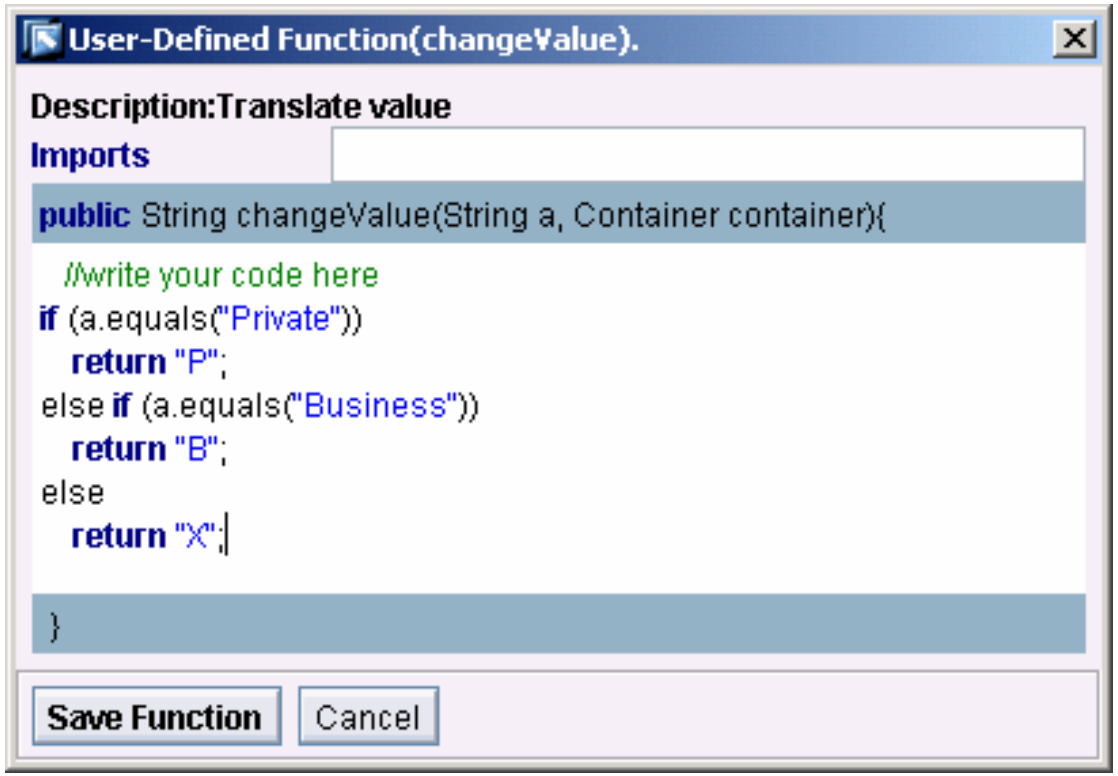
Object	Use
Container	This object enables you to cache the values that you want to read again when you next call the same user-defined function.
GlobalContainer	This object enables you to cache the values that you want to read again when you next call any user-defined function that is in the same message mapping.
MappingTrace	This object enables you to transfer information for the mapping trace during mapping to a container that can be viewed by users in the message monitoring.

User-Defined Functions – Editor



User-Defined Simple Function

- Select Simple Function
- Single return value is of type `java.lang.String`



- Import Java packages if required,

  e.g. com.company.sap.xi.mytools.*; com.company.sap.xi.myutils.*;

Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions
- Function Usage
- Simple Function Examples
- Node Function Examples

Introduction to Most–Common Java Usage

User-defined Functions
- Usage

Advanced User–Defined Functions
- **Usage**
- **ResultList, Container Object, GlobalContainer Object, MappingTrace Object**

Message Mapping “Patterns”
- Summarization
- Duplicating Subtrees
- Table/Value Lookup (within source, not external)
- Tree–Reversal

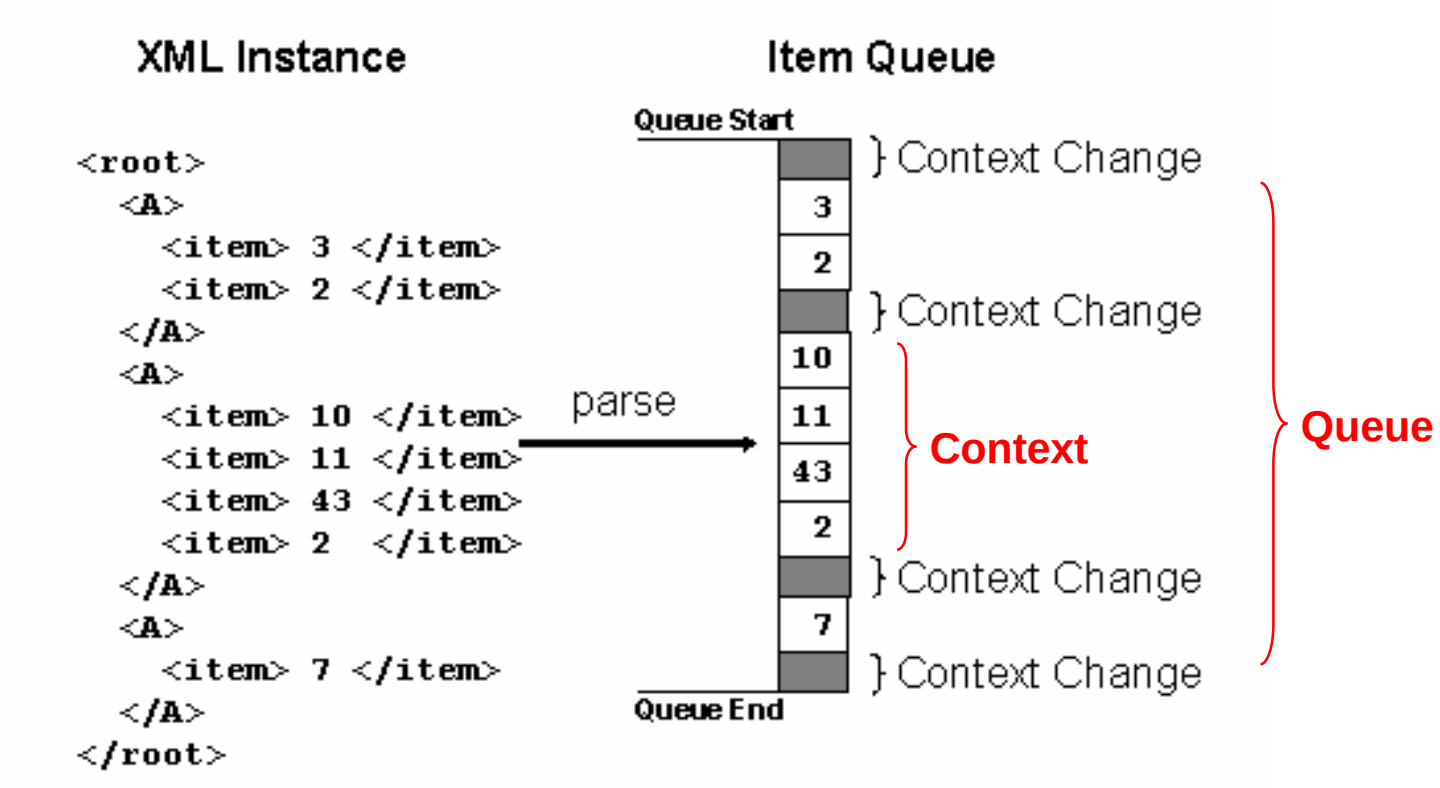
Multi–Mappings

Mapping Templates

Summary

Advanced User–Defined Functions

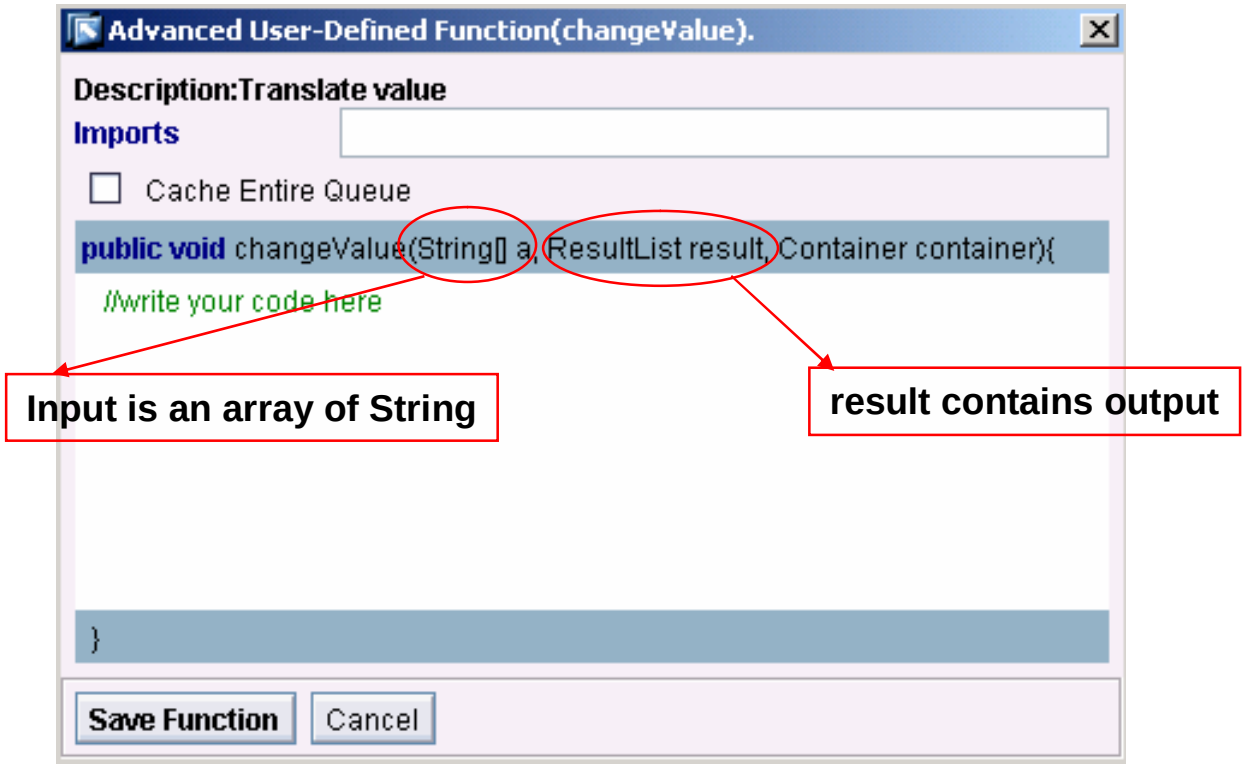
Advanced user–defined functions can access more than just individual values. A complete context or an entire queue can be accessed.



Message Mapping works by using queues. There is a queue for each element.

User-Defined Advanced Function

- Select Advanced Function
- Values to be returned are in the object result



- Import Java packages if required,

  e.g. com.company.sap.xi.mytools.*; com.company.sap.xi.myutils.*;

Advanced User–Defined Functions

Advanced user–functions can import either just one context or the complete queue into input String arrays.

To import the complete queue, check the “Cache Entire Queue” checkbox in the function editor.

Working with Contexts or Queues

Information in Cache	Implications
Context	Do not have identifiable context change.
Queue	Contains context change indicator. Much more memory intensive.

Context changes at the beginning and end of the context or queue are implicit and cannot identified.

Advanced User–Defined Functions

A queue can have the following entries:

Value	Meaning
(empty string)	In the queue, there is an entry with an empty string each time the field appears in the XML instance, e.g. <A />, <root />.
(string)	Value of a value field, e.g. <item>2</item>
ResultList.CC	Constant that shows a context change.
ResultList.SUPPRESS	Constant that causes a field and its sub node to be ignored during processing.

Special structures can be created by adding or removing context changes, e.g. by using functions, SplitByValue() or removeContexts() or manually inserting the constant.

Advanced User–Defined Functions – ResultList Object

This object is used in advanced user–defined functions to return the result of the function. It can be used to either return a queue or to return the context of a queue.

Methods of the `ResultList` Object

Method	Use
void addValue(String value);	Append a value to the results list.
void addContextChange();	Appends a context change to the list. This is the same as append ResultList.CC by using addValue().
void addSuppress();	Appends the constant ResultList.SUPPRESS to the list.
void clear();	Deletes all previously appended values from the list.

Advanced User–Defined Functions – ResultList Object

The ResultList.CC constant can be used to find context changes in the input array, when the “Cache Entire Queue” is checked. For example:

```
if ( a[index].equals(ResultList.cc) ) …
```

Note:

- **When the input array contains only context, no context change exists.**
- **When the input array contains the entire queue, context changes can exist, but not at the beginning or end of the queue.**

Advanced User–Defined Functions – Container Object

Use Container object to:

Cache (or save) data which can be accessed again when the user–function is called again.

Methods	Use
void setParameter(String parName, object obj);	Saves the *obj* object under the name *parName* in a container for a user–specific function.
Object getParameter (String parName);	Returns the parameter that was saved under the *parName* name in this user–specific function by using the **setParameter** method. If no such parameter exists, *null* is returned.
MappingTrace getTrace();	Returns a *MappingTrace* object with which messages can be written to the mapping trace in the monitor. (to be discussed)
GlobalContainer getGlobalContainer();	Returns a GlobalContainer object with which data can be saved and read by any user–defined functions in the same message mapping. (to be discussed)
Java.util.Map getTransformationParameters();	Returns a map with the mapping runtime constants. (to be discussed)

Advanced User–Defined Functions – Container Object

Use Container object:

To access mapping runtime constants, e.g. sender or receiver.

- **The constants are attributes of com.sap.aii.mapping.api.StreamTransformationConstants. Therefore, it must be imported in the user–defined function.**
- **The following values can be obtained:** PROCESSING_MODE, MESSAGE_ID, REF_TO_MESSAGE_ID, TIME_SENT, INTERFACE, INTERFACE_NAMESPACE, SENDER_PARTY, SENDER_SERVICE, RECEIVER_NAME, RECEIVER_NAMESPACE, RECEIVER_PARTY, RECEIVER_SERVICE**, etc. (additional ones can be found in the Java Mapping documentation)**
- **Usage example:**

```
String constant;
java.util.Map map;
// get constant map
map = container.getTransformationParameters();
constant = (String) map.get(
        StreamTransformationConstants.INTERFACE_NAMESPACE);
return constant;
```

(during design time, test string is used.)

Advanced User–Defined Functions – GlobalContainer Object

This object enables you to cache the values that you want to read again when you next call any user-defined function that is in the same message mapping.

Note that the sequence in which user-defined functions are called is predefined. It depends on the position of the target fields that the function was assigned to.

To get a global container object, call the *getGlobalContainer()* method for the *Container* Object.

Methods	Use
void setParameter(String parName, object obj);	Saves the *obj* object under the name *parName* in a global container for a user–specific function.
Object getParameter (String parName);	Returns the parameter that was saved under the *parName* name in this user–specific function by using the **setParameter** method. If no such parameter exists, *null* is returned.

Advanced User–Defined Functions – MappingTrace Object

This object enables the writing of messages to the mapping trace in the message monitoring (SXMB_MONI).

Trace Level	Method	Use
0		No trace
1	void addInfo(String message);	Add message with trace level *info*.
2	void addWarning(String message);	Add message with trace level *warning*.
3	void addDebugMessage(String message);	Add message with trace level *debug*.

Trace level can be set in the Pipeline configuration. Each trace level will include messages from lesser trace levels, e.g. *debug* will include messages from *warning* and *info*.

Usage example:

```
MappingTrace trace = container.getTrace();
trace.addInfo("This is a trace message.");
```

Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions
- Function Usage
- Simple Function Examples
- Node Function Examples

Introduction to Most–Common Java Usage

User-defined Functions
- Usage

Advanced User–Defined Functions
- Usage
- ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”
- **Summarization**
- **Sequence–Number Generation**
- **Duplicating Subtrees**

Multi–Mappings

Mapping Templates

Summary

Mapping “Patterns”

Mapping “patterns” are some of the common mapping requirements we found in the past which XSLT was used, because we did not think that Message Mapping provided the functionality

- **Summarization**
- **Sequence–Number Generation**
- **Duplicating Subtrees**

Mapping “Patterns” – Summarization

Summarization is when we try to consolidate detailed information into total/subtotals and counts.

The count and sum functions will act on the content of a context. Therefore, the context of the group to be summarized must be selected appropriately.

Example:

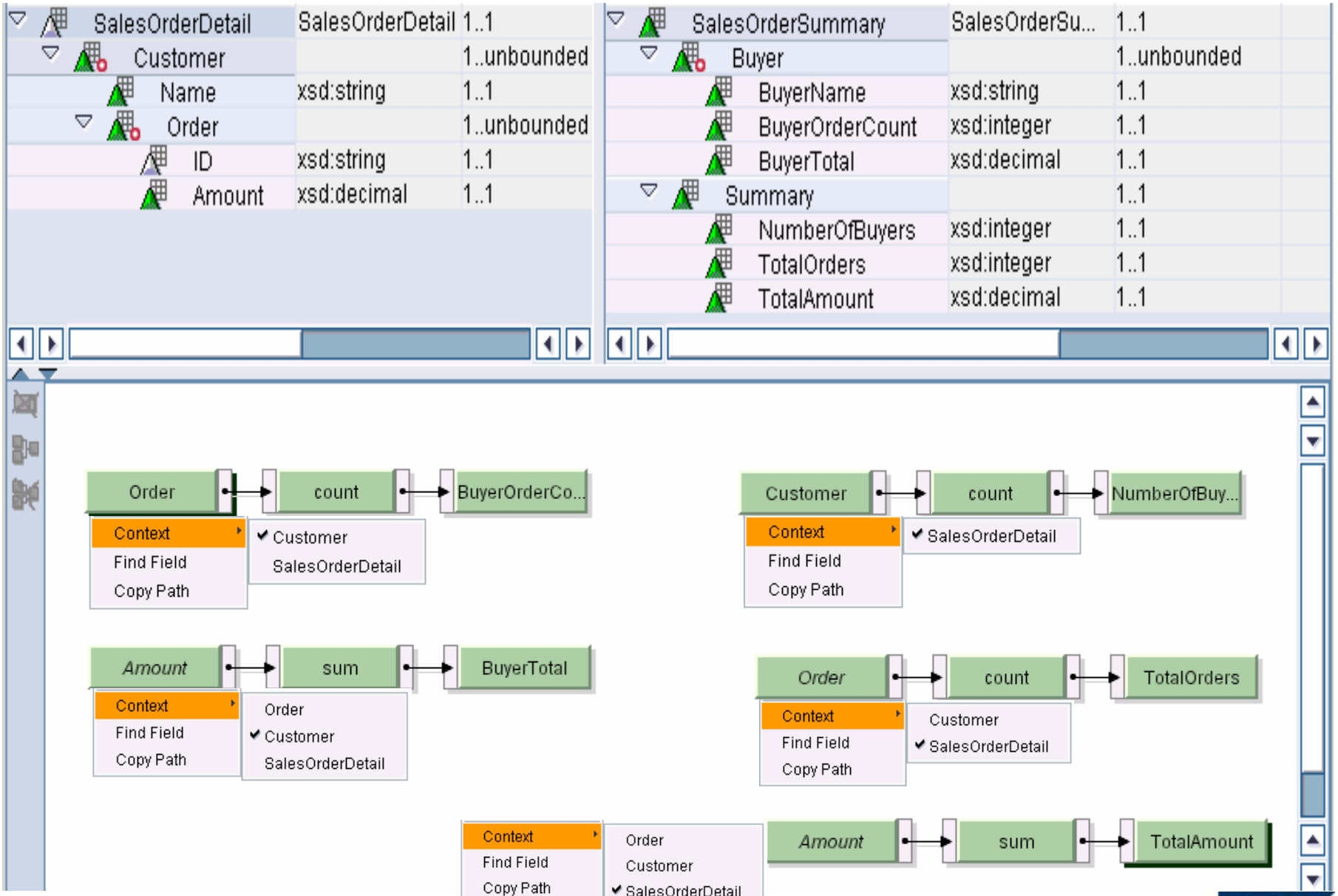
Source: Order details containing customer, order numbers and order amount

Structure	Category	Type	Occurrence
SalesOrderDetail	Complex Type		
Customer	Element		1..unbounded
Name	Element	xsd:string	1
Order	Element		1..unbounded
ID	Element	xsd:string	1
Amount	Element	xsd:deci...	1

Target: Total and count the order by each customer and a summary of all orders

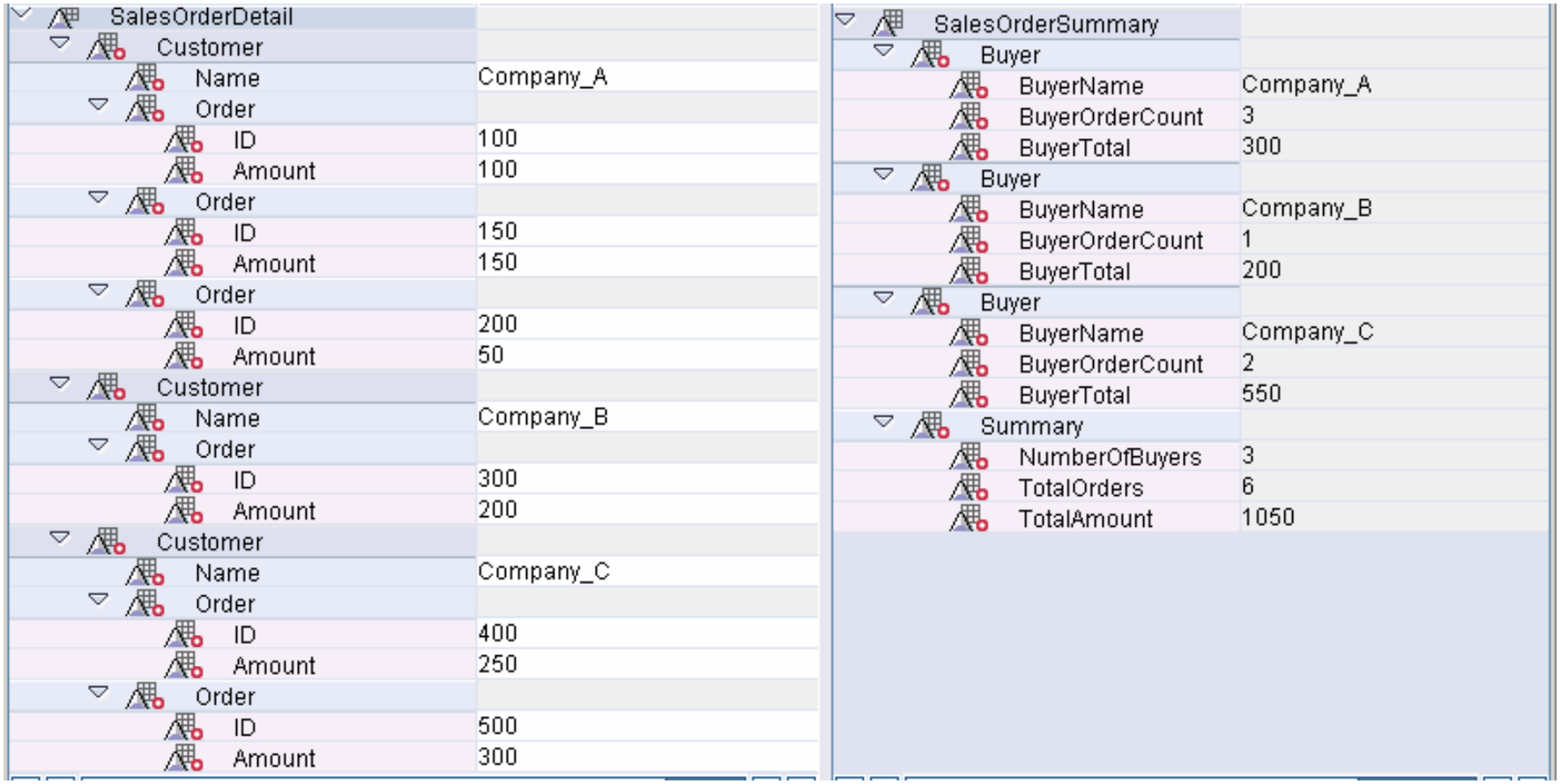
Structure	Category	Type	Occurrence
SalesOrderSummary	Complex Type		
Buyer	Element		1..unbounded
BuyerName	Element	xsd:string	1
BuyerOrderCount	Element	xsd:integ...	1
BuyerTotal	Element	xsd:deci...	1
Summary	Element		1
NumberOfBuyers	Element	xsd:integ...	1
TotalOrders	Element	xsd:integ...	1
TotalAmount	Element	xsd:deci...	1

Mapping “Patterns” – Summarization



Mapping “Patterns” – Summarization

Mapping Result:



Mapping “Patterns” – Sequence–Number Generation

There are situations when sequence numbers had to be generated based on the number of occurrences of the source data. This is frequently used during mapping to SAP IDocs or BAPIs when item number, which is no available from the source, had to be added.

Example:

Source: Order information.

Structure	Category	Type	Occurrence
ItemSeqNoIn	Complex Type		
row	Element		1..unbounded
name	Element	xsd:string	1
ordertype	Element	xsd:string	1

Target: Two complex elements had to be populated from the source. Each one contains a sequence number which matches the position of the data in the source.

Structure	Category	Type	Occurrence
ItemSeqNoOut	Complex Type		
item	Element		1..unbounded
itemNo	Element	xsd:integer	1
name	Element	xsd:string	1
row	Element		1..unbounded
seqNo	Element	xsd:integer	1
type	Element	xsd:string	1

Sequence number to be generated.

Mapping “Patterns” – Sequence–Number Generation

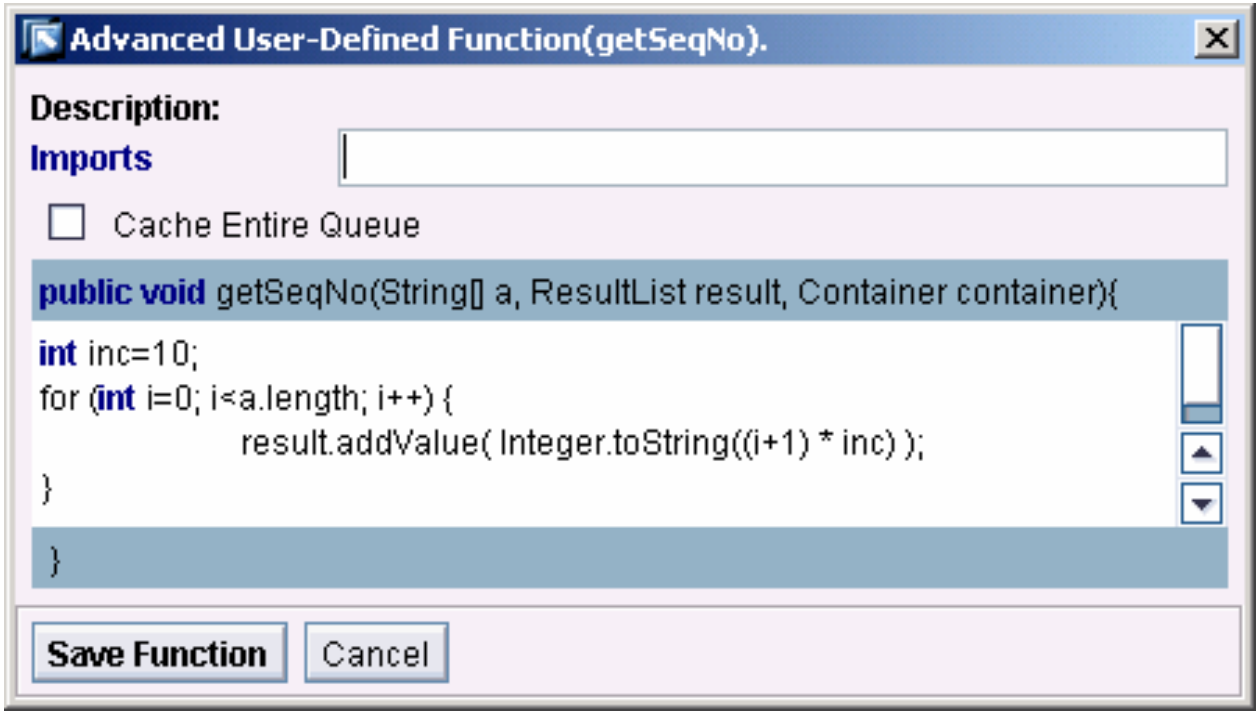
There are two ways the sequence number can be generated.

1. **Use Advanced User–Defined function to generate the sequence number all at once.**

2. **Use Simple User–Defined function to generate the sequence number for each occurrence separately, keeping track of the sequence number value previously generated.**

Mapping “Patterns” – Sequence–Number Generation

1. **Use Advanced User–Defined function to generate the sequence number all at once.**

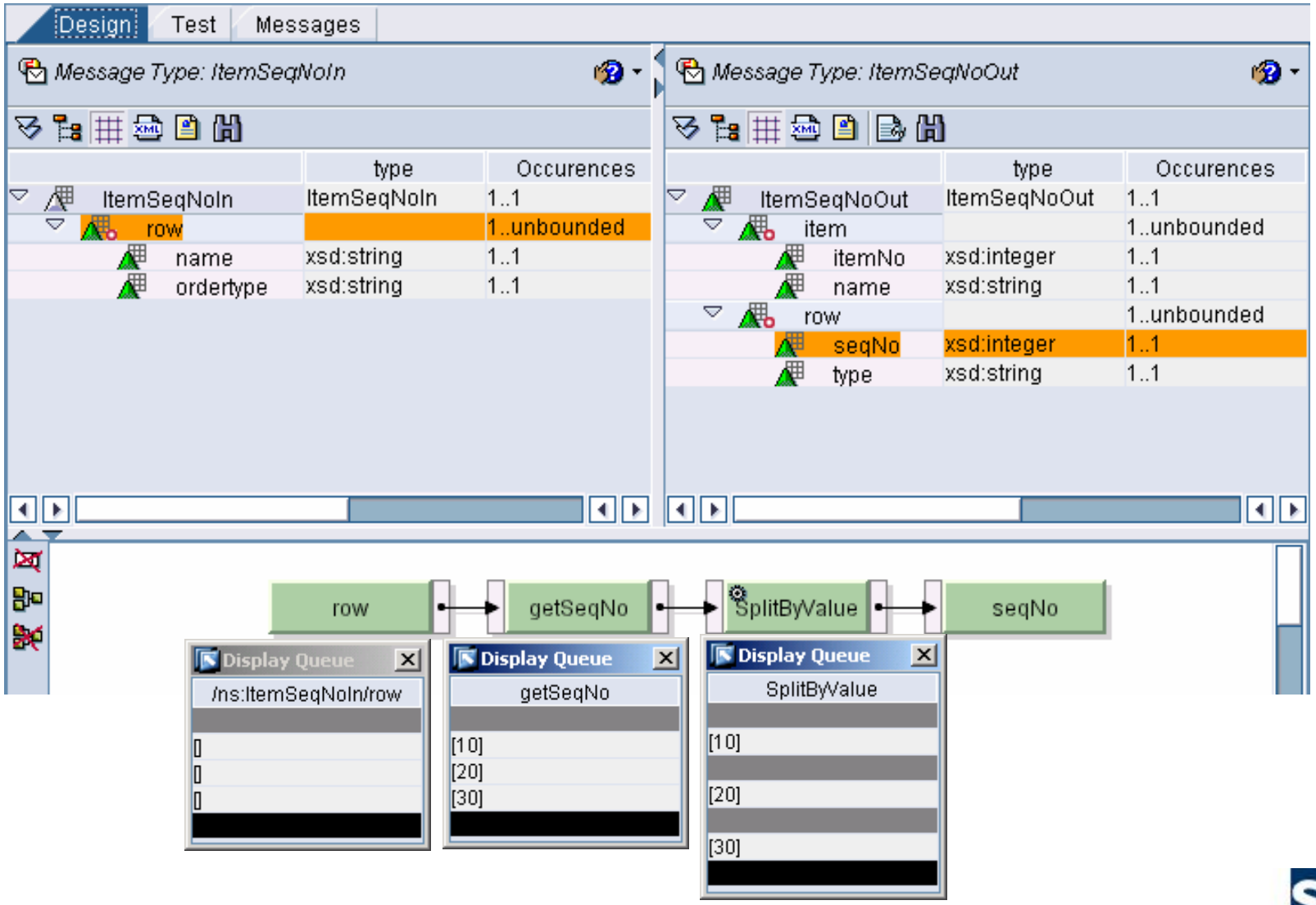


The java function creates the sequence number all at once based on the number of elements in the source.

The sequence numbers are written to the ResultList object.

Mapping “Patterns” – Sequence–Number Generation

1. **Use Advanced User–Defined function to generate the sequence number all at once.**



Mapping “Patterns” – Sequence–Number Generation

1. **Use Advanced User–Defined function to generate the sequence number all at once.**

Test Results:

	value
ItemSeqNoIn	
row	
name	aaa
ordertype	111
row	
name	bbb
ordertype	222
row	
name	ccc
ordertype	333

	value
ItemSeqNoOut	
item	
itemNo	10
name	aaa
item	
itemNo	20
name	bbb
item	
itemNo	30
name	ccc
row	
seqNo	10
type	111
row	
seqNo	20
type	222
row	
seqNo	30
type	333

Mapping “Patterns” – Sequence–Number Generation

2. **Use Simple User–Defined function to generate the sequence number for each occurrence separately, keeping track of the sequence number value previously generated.**

User-Defined Function(getNextSeqNo).

Description:

Imports

```
public String getNextSeqNo(Container container){
int inc = 10;
Integer seqNo = (Integer) container.getParameter("seqNo");
if (seqNo == null) {
                seqNo = new Integer( inc );
}
else {
                int number = seqNo.intValue() + inc;
                seqNo = new Integer(number);
}
container.setParameter("seqNo", seqNo);
return seqNo.toString();

}
```

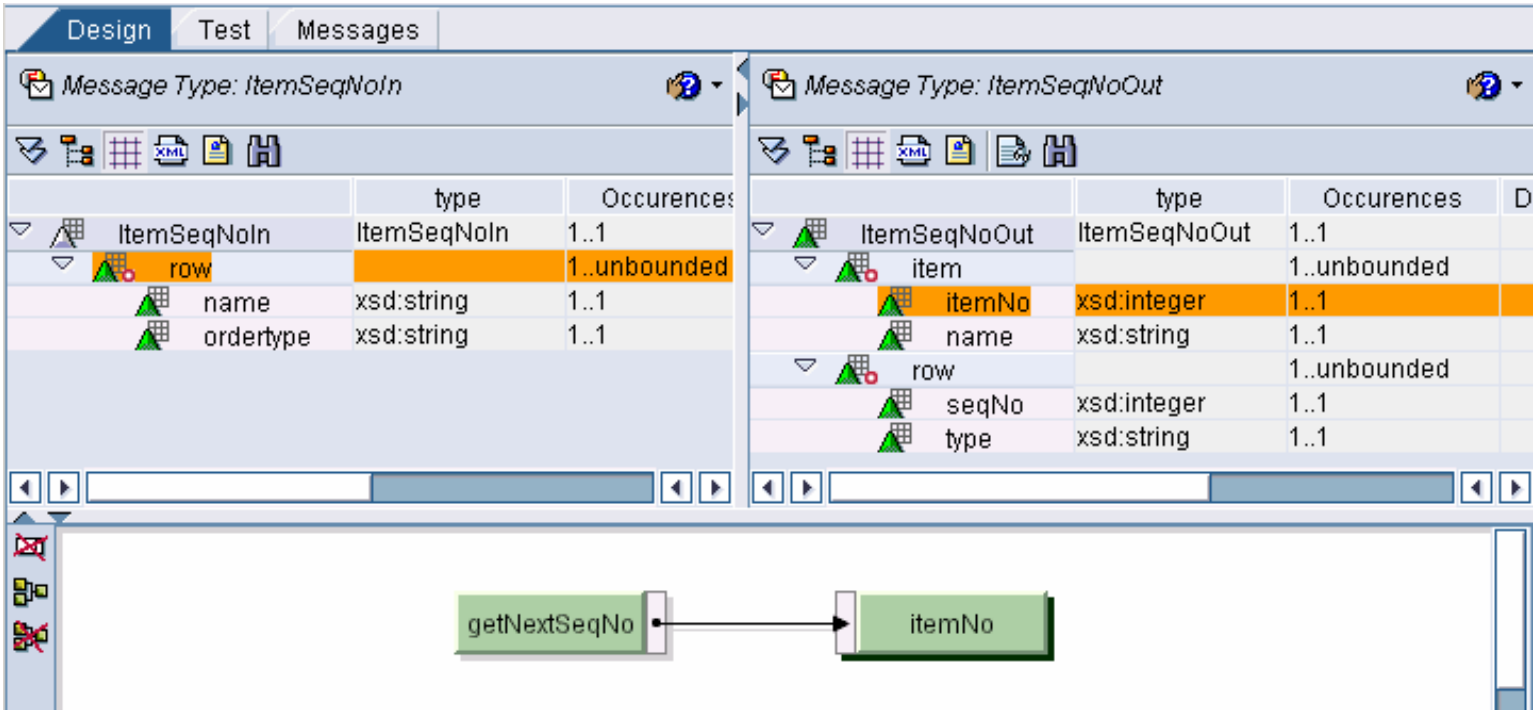
Save Function | Cancel

The sequence number is stored in the container object. Each time it is retrieved, incremented and saved.

Note: There is no input required.

Mapping “Patterns” – Sequence–Number Generation

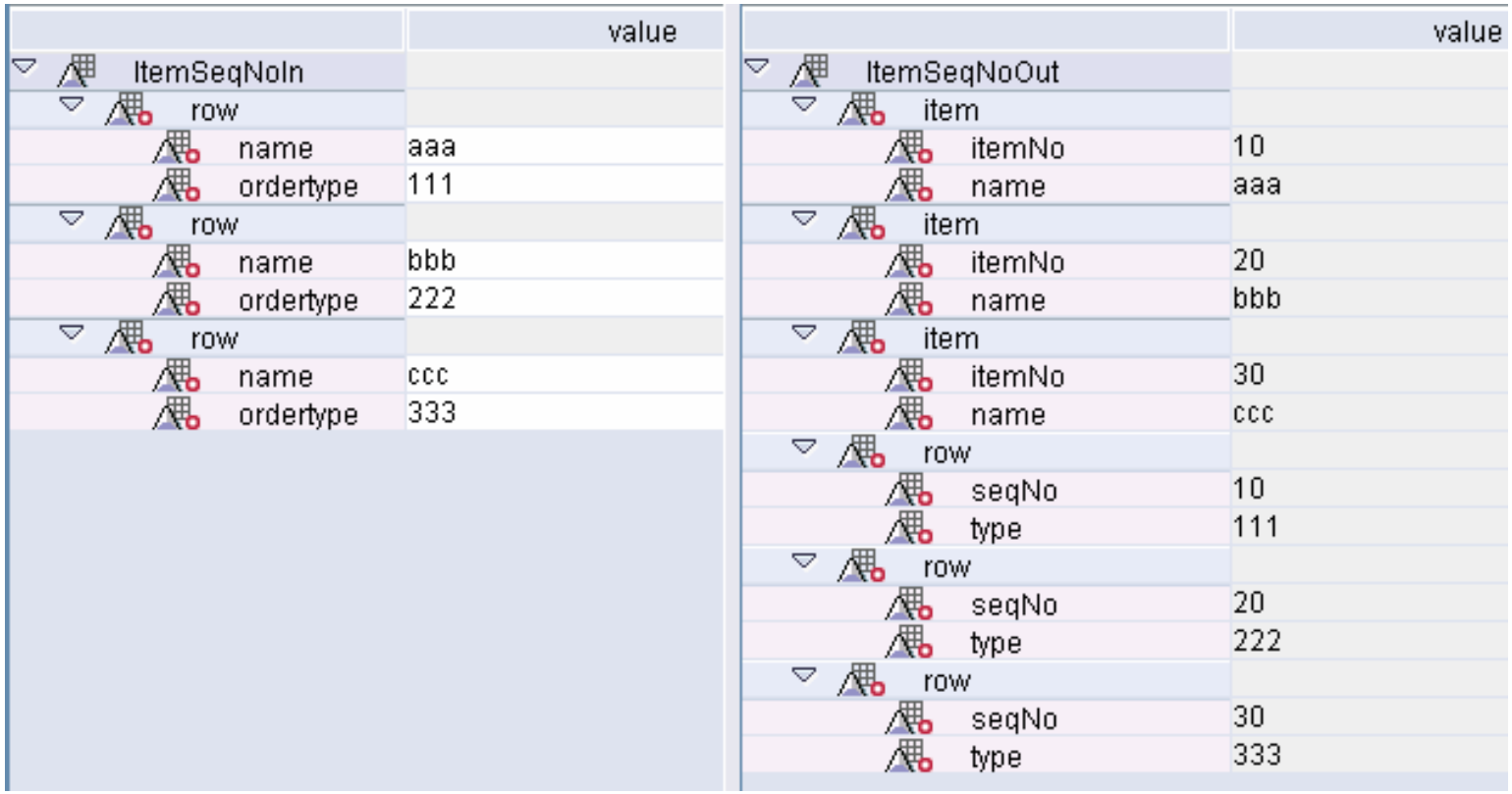
2. **Use Simple User–Defined function to generate the sequence number for each occurrence separately, keeping track of the sequence number value previously generated.**



We cannot display the queue to examine the result of the mapping.

Mapping “Patterns” – Sequence–Number Generation

2. **Use Simple User–Defined function to generate the sequence number for each occurrence separately, keeping track of the sequence number value previously generated.**



	value
ItemSeqNoIn	
row	
name	aaa
ordertype	111
row	
name	bbb
ordertype	222
row	
name	ccc
ordertype	333

	value
ItemSeqNoOut	
item	
itemNo	10
name	aaa
item	
itemNo	20
name	bbb
item	
itemNo	30
name	ccc
row	
seqNo	10
type	111
row	
seqNo	20
type	222
row	
seqNo	30
type	333

Mapping “Patterns” – Duplicating Subtrees

Even if elements are shown to occur more than once in the XML instance according to XML Schema Definition, they are only displayed once in the structure overview.

To assign source field(s) to multiple positions of an element in the target structure, the element or the entire subree can be duplicated (copied) using the context menu in the target structure.

Example:

Source: Accounting information with adjustment type and amount.

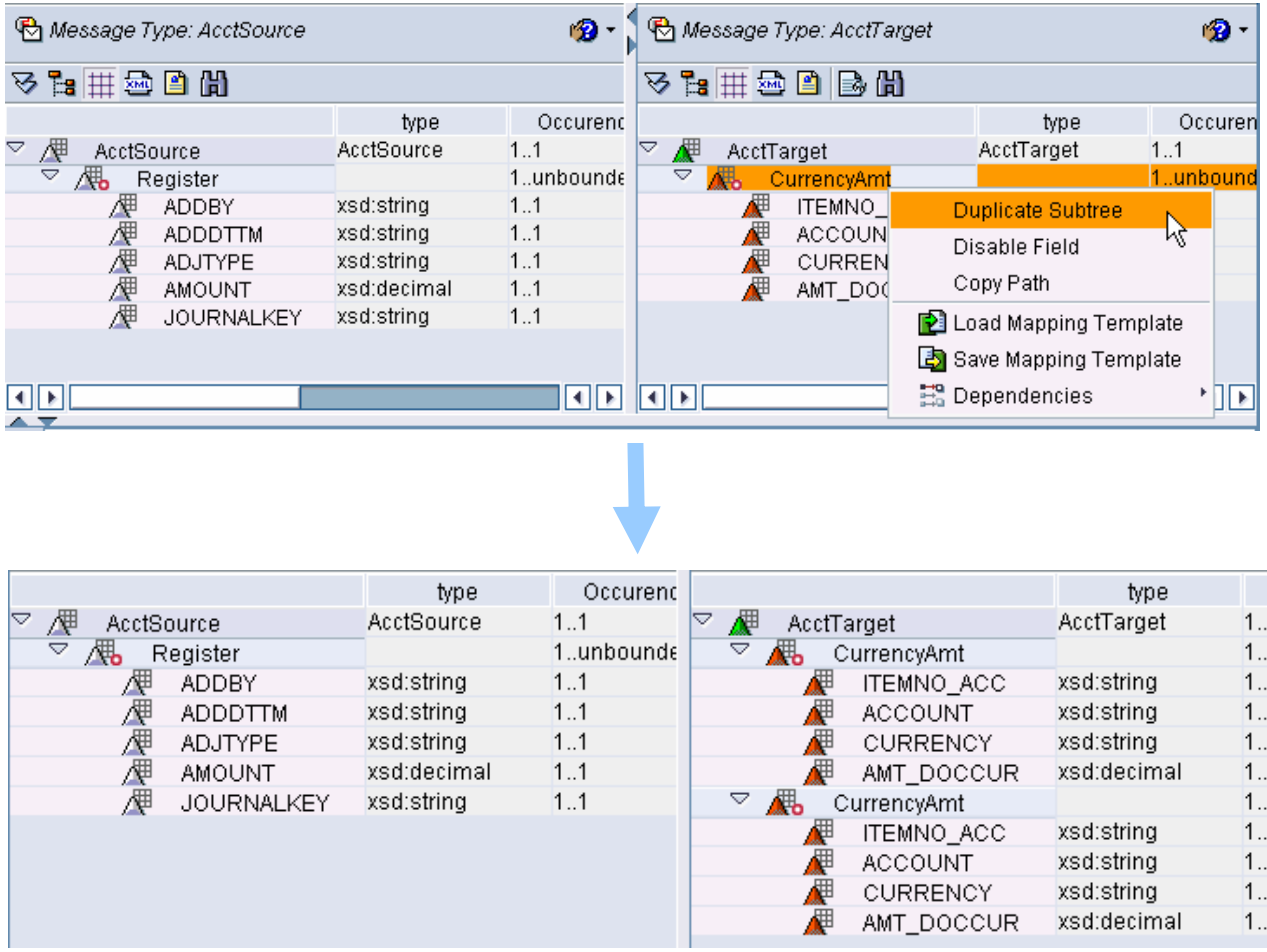
Structure	Category	Type	Occurrence
AcctSource	Complex Type		
Register	Element		1..unbounded
ADDBY	Element	xsd:string	1
ADDDTTM	Element	xsd:string	1
ADJTYPE	Element	xsd:string	1
AMOUNT	Element	xsd:decimal	1
JOURNALKEY	Element	xsd:string	1

Target: Two CurrencyAmt’s are to be created. One is to credit one account. The other is to debit another account. The amount will either be positive or negative depending on the ADJTYPE.

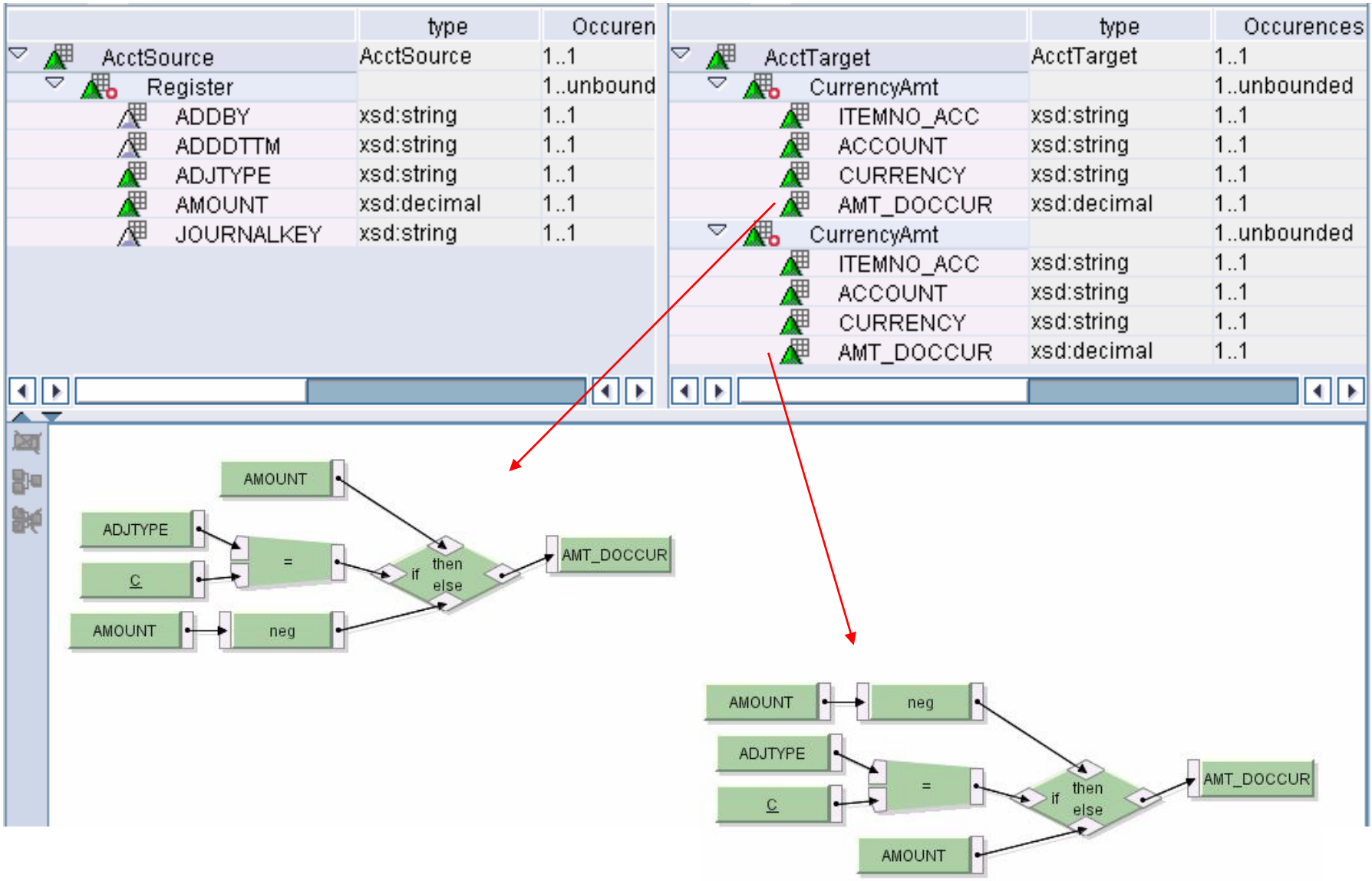
Structure	Category	Type	Occurrence
AcctTarget	Complex Type		
CurrencyAmt	Element		1..unbounded
ITEMNO_ACC	Element	xsd:string	1
ACCOUNT	Element	xsd:string	1
CURRENCY	Element	xsd:string	1
AMT_DOCCUR	Element	xsd:deci...	1

Mapping “Patterns” – Duplicating Subtrees

The subtree “CurrencyAmt” must be duplicated first. This will create 2 CurrencyAmt’s. One for credit, and one for debit.

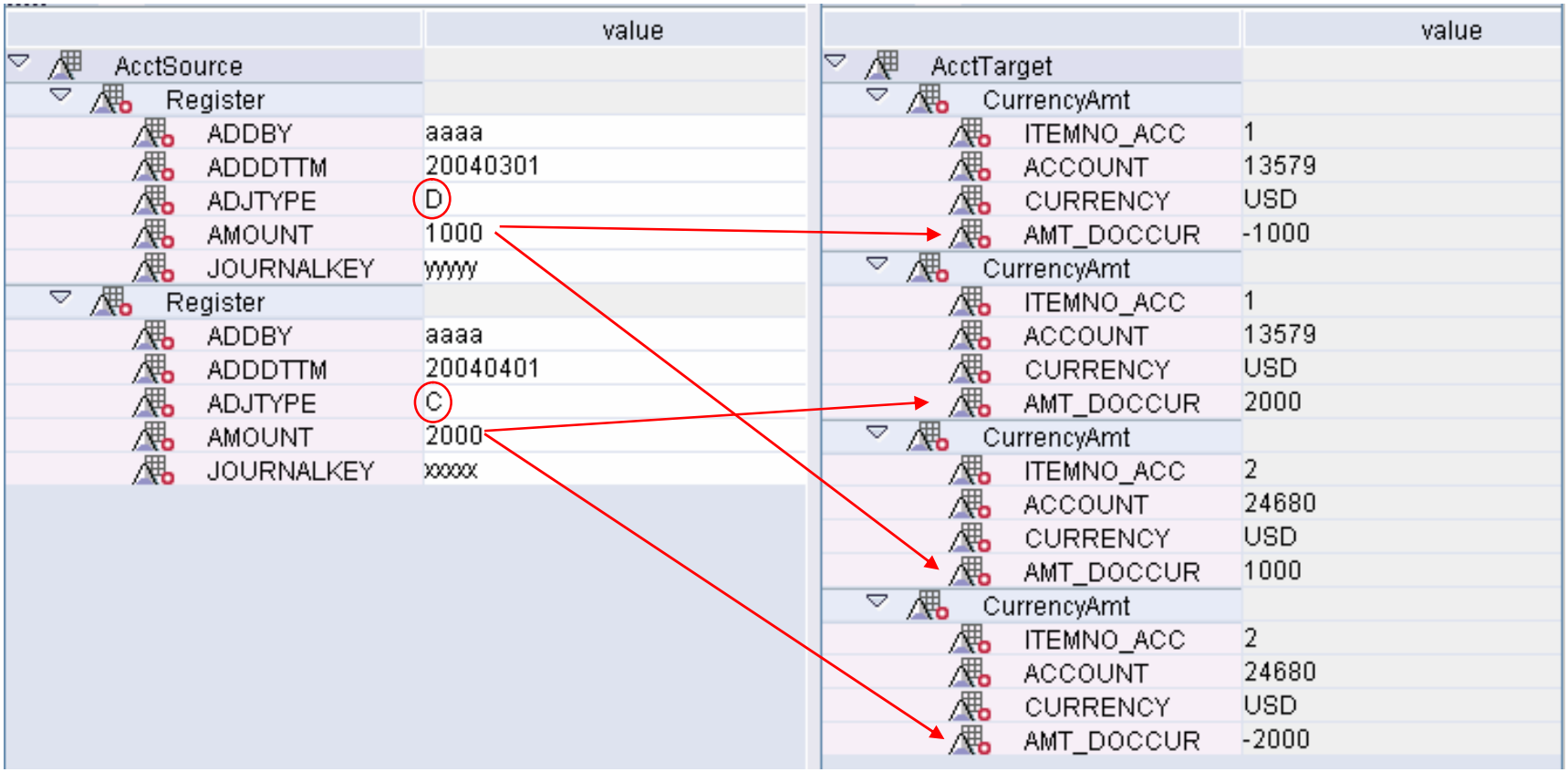


Mapping “Patterns” – Duplicating Subtrees



Mapping “Patterns” – Duplicating Subtrees

Mapping Results:



Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions
- Function Usage
- Simple Function Examples
- Node Function Examples

Introduction to Most–Common Java Usage

User-defined Functions
- Usage

Advanced User–Defined Functions
- Usage
- ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”
- Summarization
- Duplicating Subtrees
- Table/Value Lookup (within source, not external)
- Tree–Reversal

Multi–Mappings

Mapping Templates

Summary

Multi–Mappings

Multi–Mapping can only be used in ccBPM:

- **n:1 Transformation**

  Bundles multiple messages into one message, for example, individual purchase order items into one purchase order.

- **1:n Transformation**

  Splits a message into multiple messages, for example, a purchase order into the individual purchase order items.

- **n:m Transformation**

  Converts a message into another message, for example, a message that is defined by interface A is converted to message that is defined by interface B.

Multi–Mappings reference multiple message structures:

- **All source message structures are combined into 1 source structure.**
- **All target message structures are combined into 1 target structure.**
- **Therefore, there is only one source structure mapped to one target.**

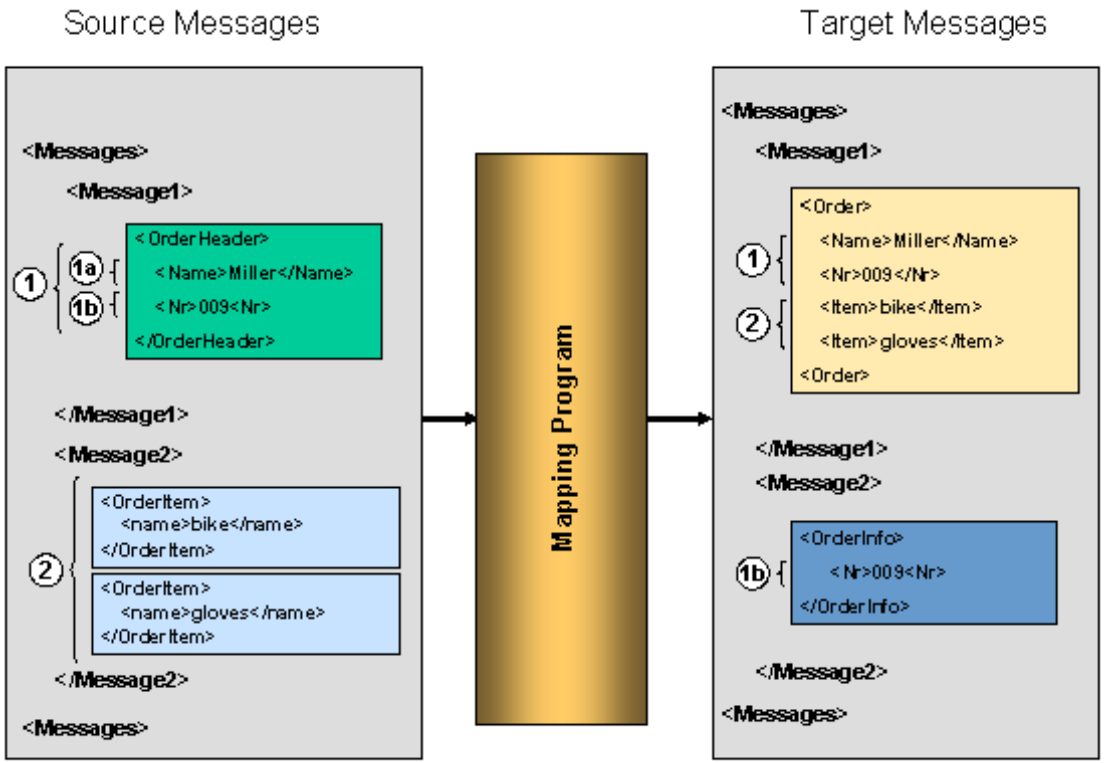
Mapping “Patterns” – Multi–Mappings

The Message Types are entered using the Messages tab in the mapping editor.

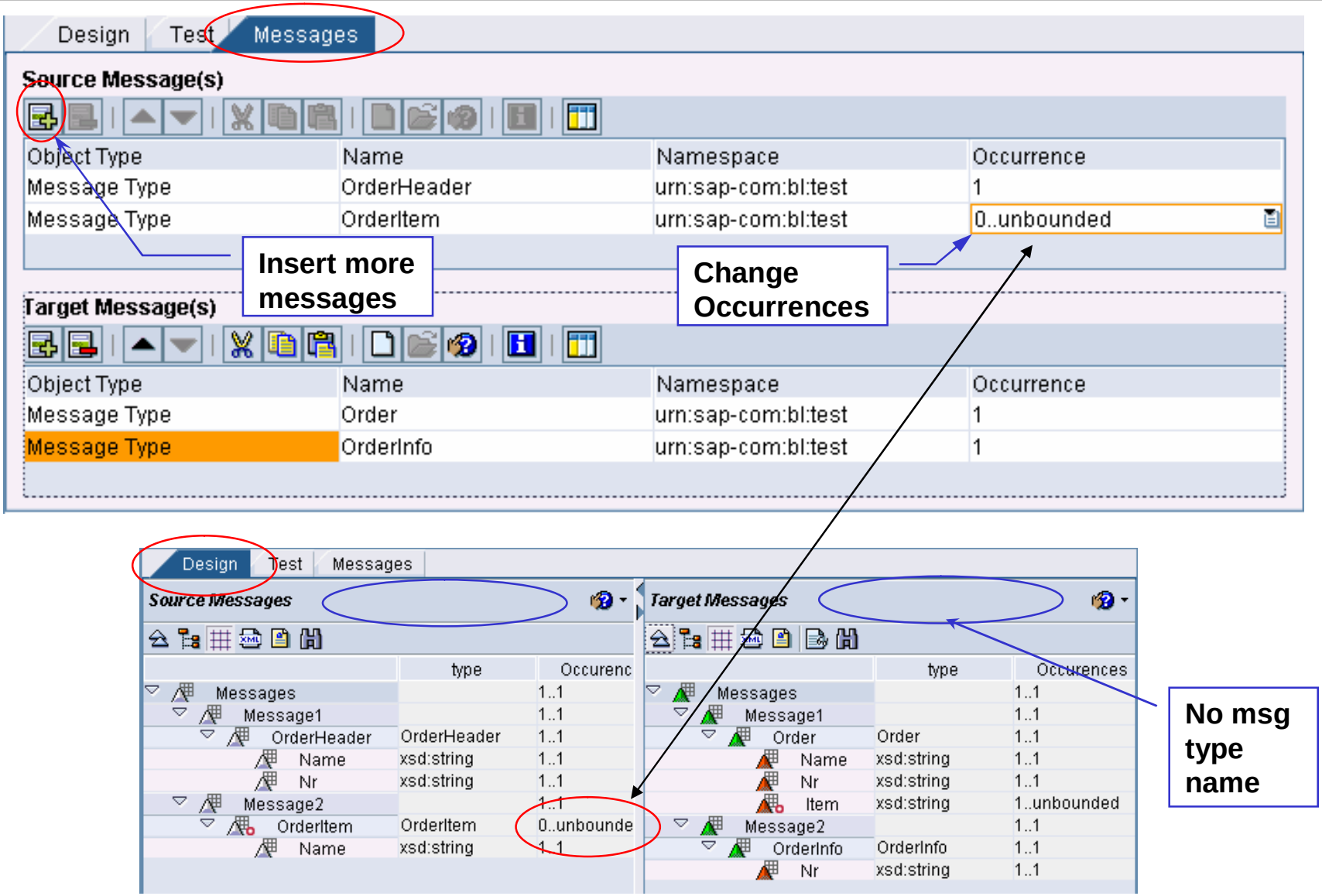
- The root element is always <Messages>.
- The Mapping Editor inserts an element <Message**N**> for each source or target message, where N is the position of the message.

Example:

Message Type (Source)	Message Type (Target)
OrderHeader**(1)**	Order**(1)**
OrderItem**(0…unbounded)**	OrderInfo**(1)**



Mapping “Patterns” – Multi–Mappings



Mapping rules are the same as for previously discussed Message Mappings.

Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions
- Function Usage
- Simple Function Examples
- Node Function Examples

Introduction to Most–Common Java Usage

User-defined Functions
- Usage

Advanced User–Defined Functions
- Usage
- ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”
- Summarization
- Duplicating Subtrees
- Table/Value Lookup (within source, not external)
- Tree–Reversal

Multi–Mappings

Mapping Templates

Summary

Mapping Templates

Message Mappings can be saved as *Mapping Templates*.

***Mapping Templates* can be reused (or loaded) in other Message Mappings or Mapping Templates.**

Features:

Mapping Templates can be defined for structure mappings of the following structures:

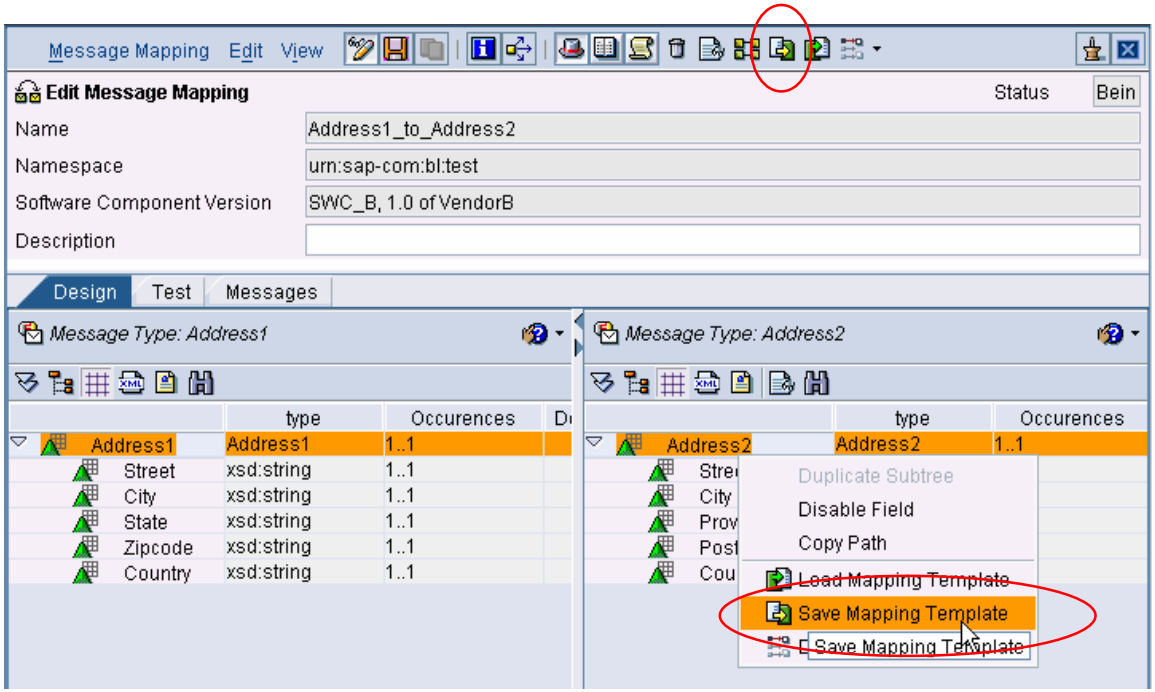
- **Data Types**
- **Complex types in IDocs and RFCs**
- **Complex types in External Definitions**
- **The referenced types used in mapping templates can be located in any software component versions.**

Mapping Templates

The Mapping Editor is used to create and load the structures used by the Mapping Templates.

Defining Mapping Templates

- **The Mapping Template can be saved by:**
  1. **Selecting the Source and Target source in the Mapping Editor**
  2. **Choose Save Mapping Template from either the target structure context menu or from the object toolbar**
  3. **Enter a unique name for the new Mapping Template.**

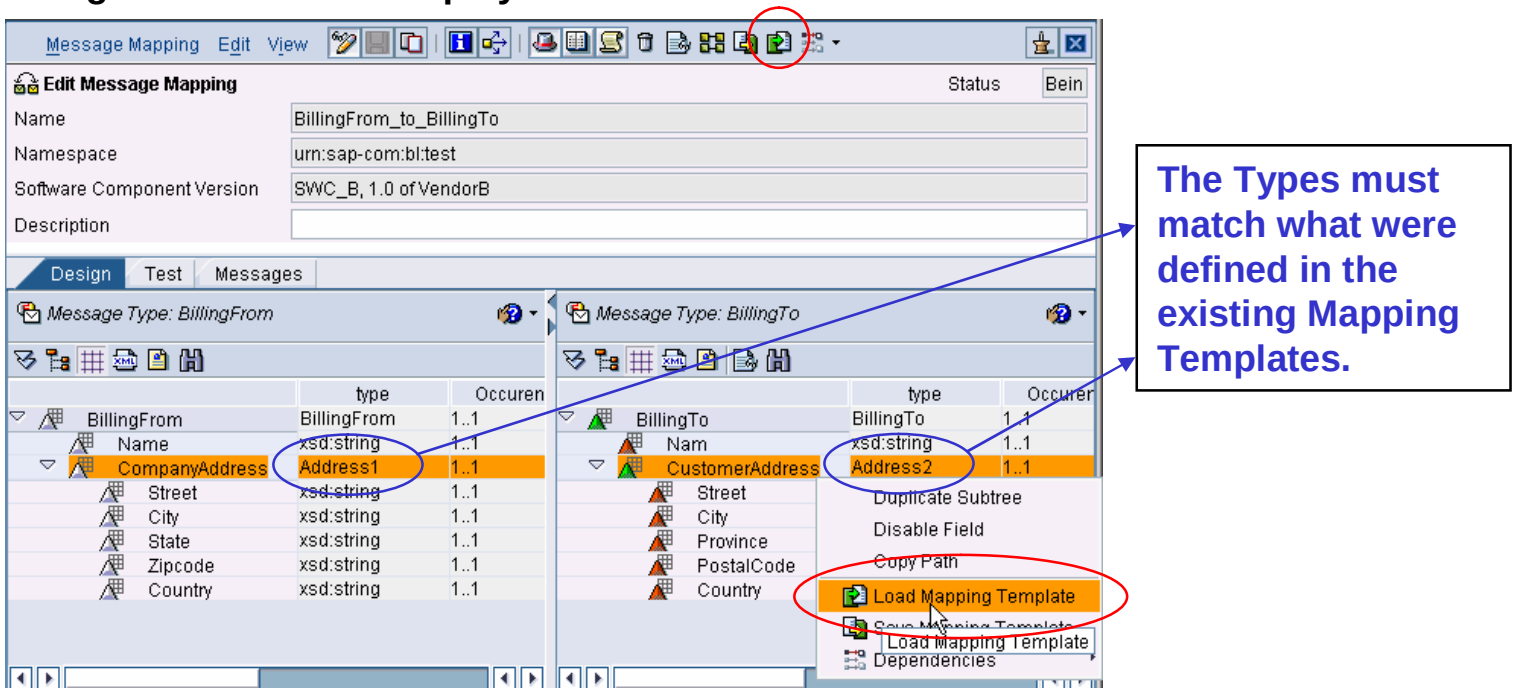


Mapping Templates

Using Mapping Templates:

Mapping Templates can be used in Message Mappings from any software component version:

1. **In the Mapping Editor, select a type element in the source and target structure.**
2. **Choose Load Mapping Templates from either the target structure context menu or from the object toolbar.**
3. **If mapping templates are available for the types in the source and target fields, a select dialog window will be displayed.**



Agenda

Concepts / Overview

Test/Debugging Environment

Standard Functions
- Function Usage
- Simple Function Examples
- Node Function Examples

Introduction to Most–Common Java Usage

User-defined Functions
- Usage

Advanced User–Defined Functions
- Usage
- ResultList, Container Object, GlobalContainer Object, MappingTrace Object

Message Mapping “Patterns”
- Summarization
- Duplicating Subtrees
- Table/Value Lookup (within source, not external)
- Tree–Reversal

Multi–Mappings

Mapping Templates

Summary

Further Information

è Public Web:

http://help.sap.com

http://sdn.sap.com

http://service.sap.com

SAP Customer Services Network: http://www.sap.com/services/

è Related XI 3.0 Workshop / Training Opportunities

NetWeaver04 Overview

XI 3.0 Implementation Workshop

Introduction to XML and Technical Standards

Advanced Integration Builder

Advanced BPM

Adapter Framework

B2B and Industry Standards

Proxy Development and Deployment

Questions?

Q&A

Copyright 2004 SAP AG. All Rights Reserved

Copyright 2004 SAP AG. Alle Rechte vorbehalten