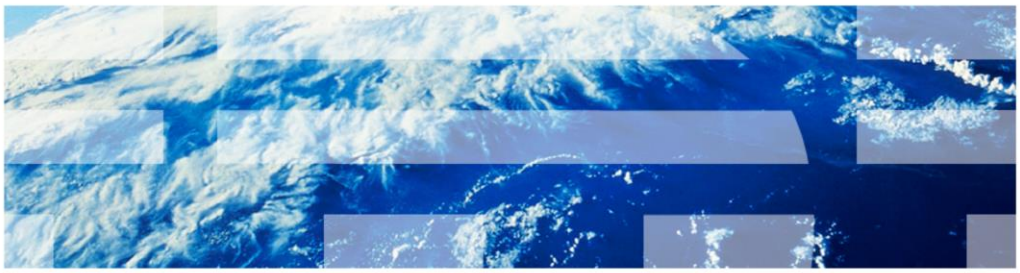
IBM

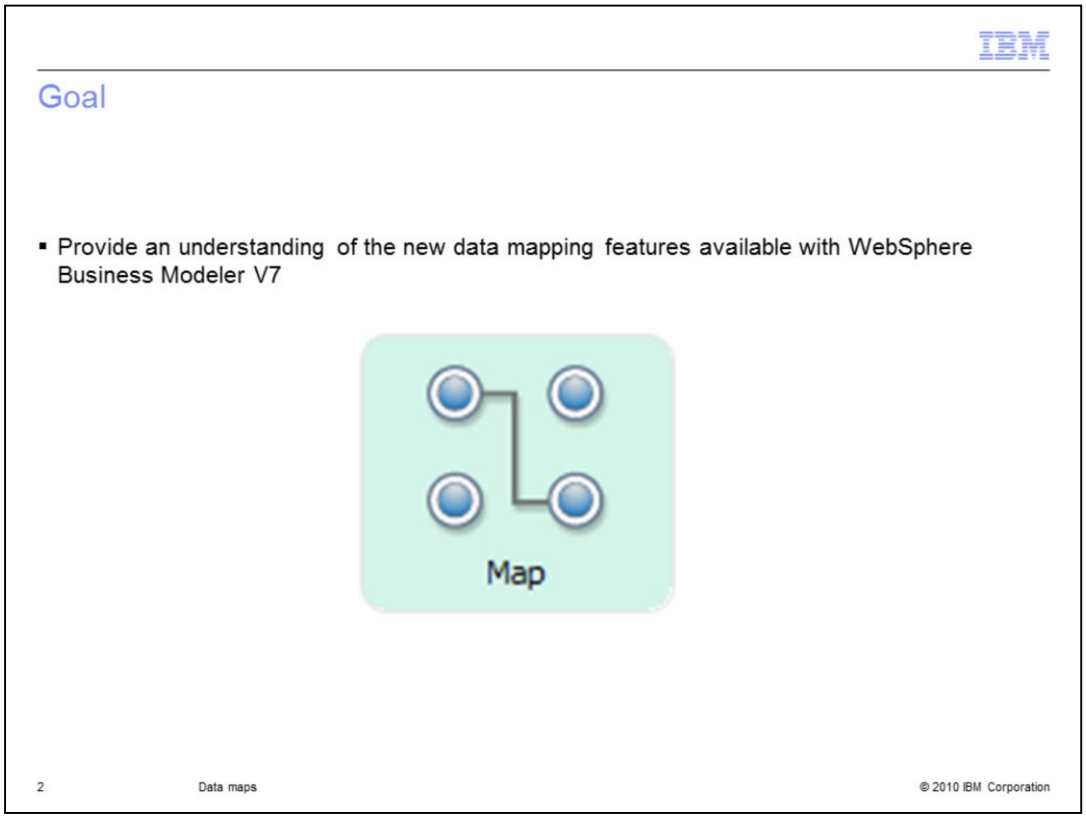
# WebSphere Business Modeler V7.0

## Data maps





This presentation provides an introduction to new data mapping features of WebSphere Business Modeler V7.0

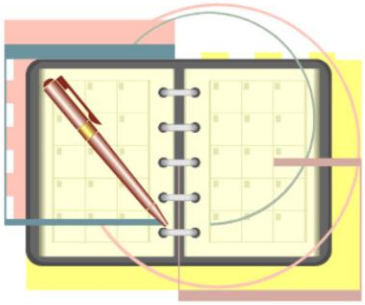



The goal of this presentation is to provide an understanding of he new data mapping features available with WebSphere Business Modeler V7.

Maps play a vital role in business process engineering, both in simulation-modeling and modeling-for-execution. With WebSphere Business Modeler V7, the foundation for data mapping has been improved dramatically with the introduction of an XSLT mapping engine. The XSLT mapping engine in WebSphere Business Modeler offers a subset of the mapping functions and transforms available with WebSphere Integration Developer V7.

## Agenda

- Introduction
- Mapping support in V7
- Global maps
- Container mapping
- Convert
- Exporting maps to WebSphere Integration Developer

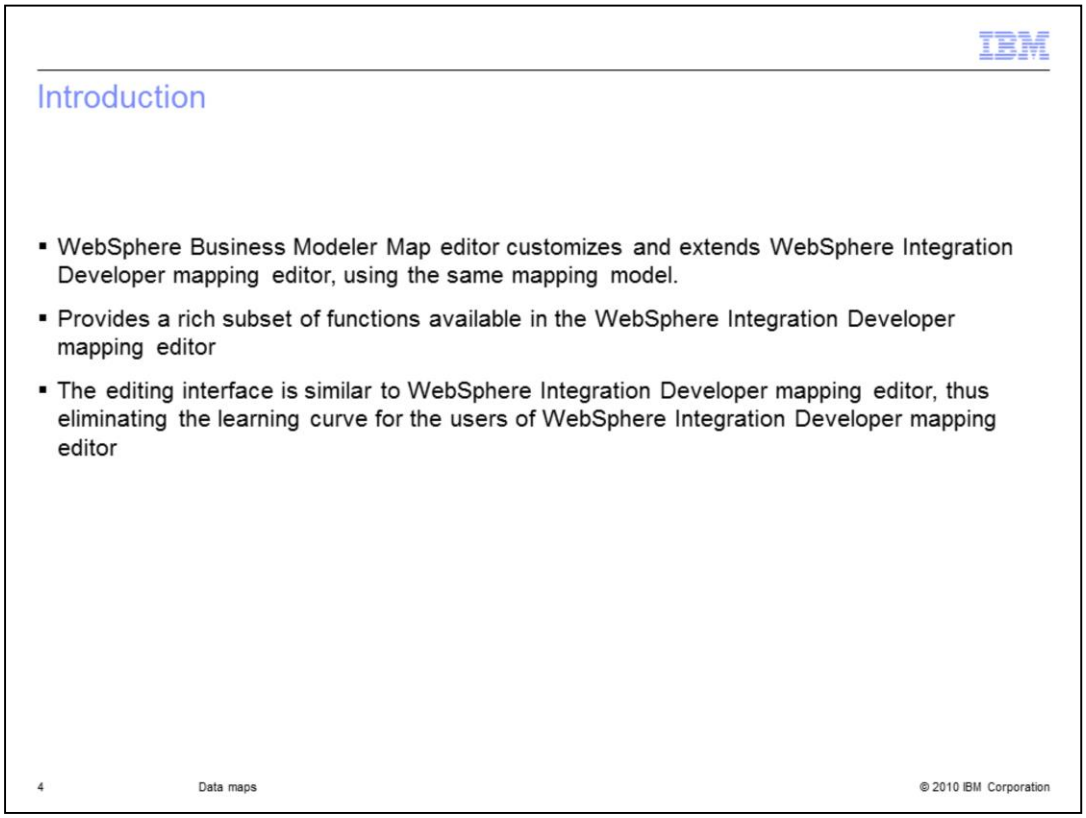


3 Data maps 

In this presentation you will learn about the new enhanced data mapping features available with WebSphere Business Modeler V7.

These new features include reusable global maps, container mapping transformations used for iterating over arrays, and data conversion transformations.

You will learn about all of these new features and how to use them.

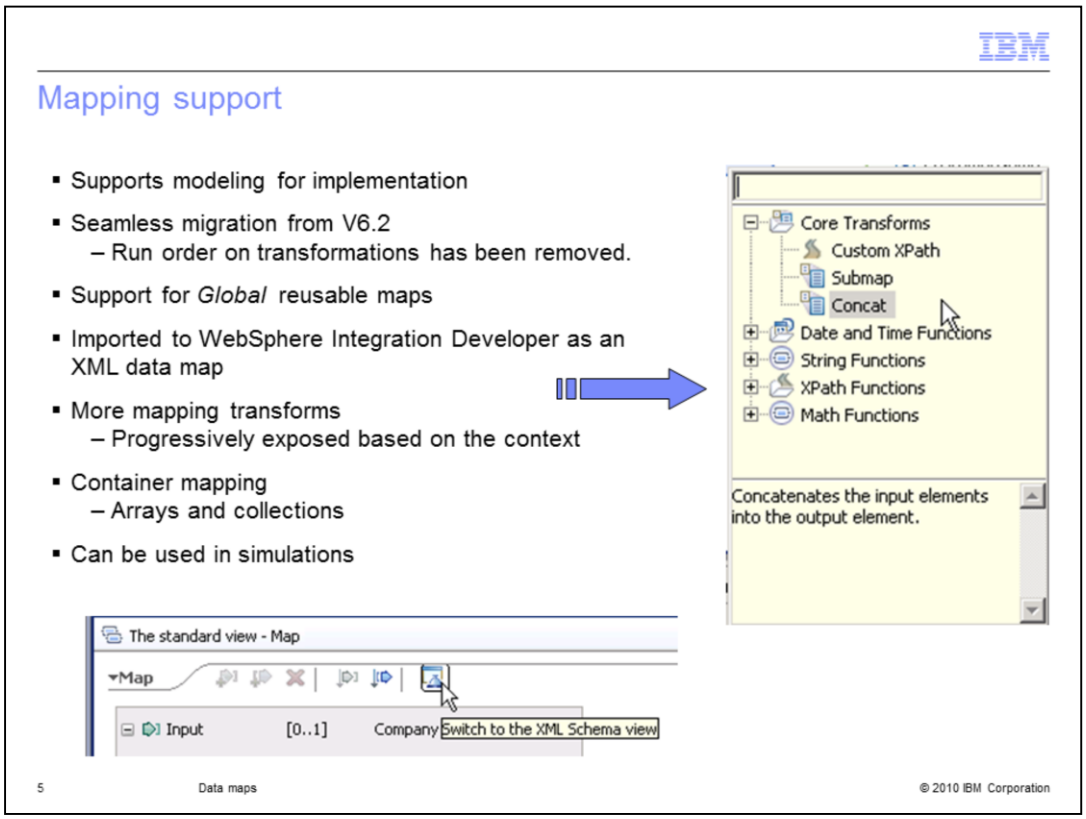



The WebSphere Business Modeler map editor customizes and extends the mapping editor used by WebSphere Integration Developer,

and uses the same mapping model.

The new map editor provides a rich subset of functions that are available with the WebSphere Integration Developer map editor.

By reusing the tools from WebSphere Integration Developer, the capabilities available to the business modeler are increased and the barrier to model driven development is reduced.




Mapping of data from one type to another is extremely important when you begin to develop your model for implementation.

In previous versions of WebSphere Business Modeler the mapping functions available were basic moves. You can take an element from one business item and assign it to an element in a different business item. When this was imported into WebSphere Integration Developer, the map element was transformed into a BPEL assign.

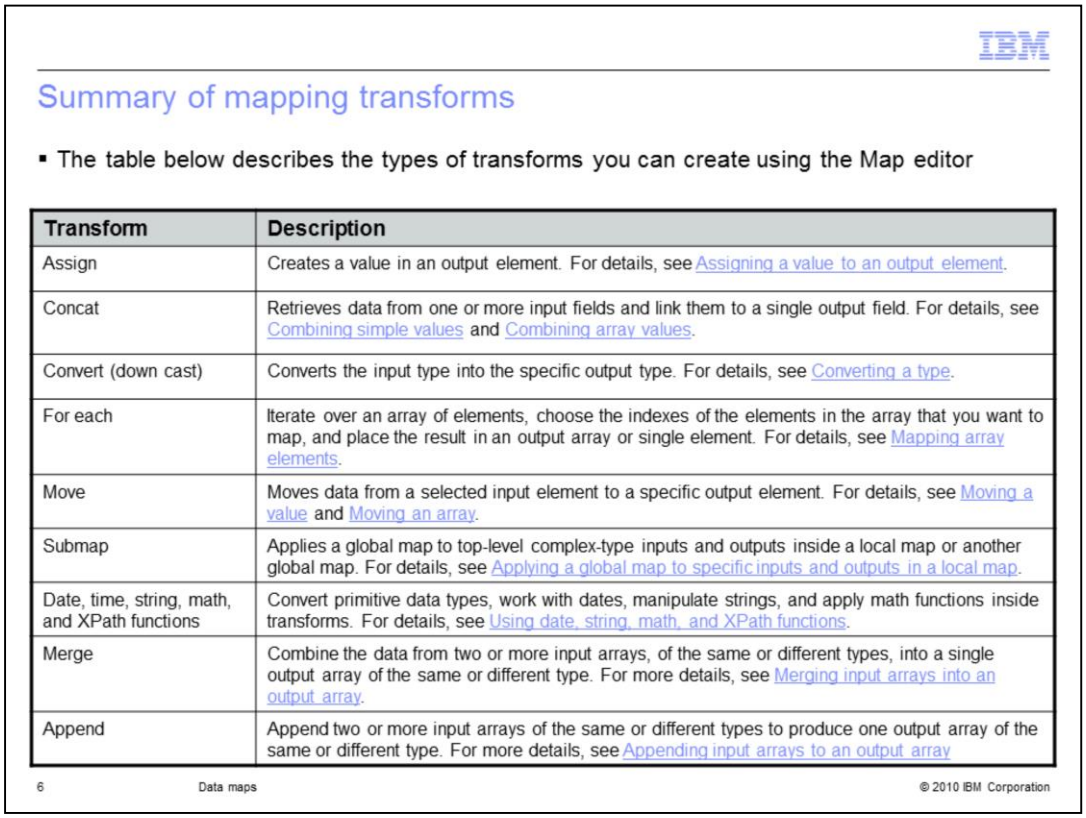
With version 7, the new mapping editor is a subset of the mapping tool used in WebSphere Integration Developer. This brings a consistent experience to the business modeler and it also provides many new data transformations. For the business modeler that has been using maps in V6.2, the migration is seamless. The one feature that you might miss is the 'run order'. This was an option to specify the order in which the transformations for a given map, are be run. It has been removed for version 7.

With the introduction of global maps you can now create a map and reuse it in many different business process flows. This will save you time and increase the accuracy of your models.

There are also many new transformations that can be done on the data as it is mapped from one business item to another. The transformations are only presented to you in a context where it makes sense to use them. That is to say, the concatenate transform won't be available to you unless you have two or more inputs. Some of the more advanced transformations won't be displayed until you switch to the XML Schema view.

Real business items are rarely simple flat structures. They are typically complex nested structures with arrays or collections embedded in them. The new mapping editor which is based on XSLT transformations, gives you the tools you need to manipulate these complex arrays or collections. This new feature is known as 'container' mapping.

Often when dealing with features that relate to runtime behavior it turns out that they can't be used with the simulation engine. This is not the case with the new enhanced mapping feature. The simulation engine can be used to verify the correctness of your mappings without the need to deploy your business process to a runtime.



IBM

## Summary of mapping transforms

- The table below describes the types of transforms you can create using the Map editor

| Transform | Description |
|---|---|
| Assign | Creates a value in an output element. For details, see Assigning a value to an output element. |
| Concat | Retrieves data from one or more input fields and link them to a single output field. For details, see Combining simple values and Combining array values. |
| Convert (down cast) | Converts the input type into the specific output type. For details, see Converting a type. |
| For each | Iterate over an array of elements, choose the indexes of the elements in the array that you want to map, and place the result in an output array or single element. For details, see Mapping array elements. |
| Move | Moves data from a selected input element to a specific output element. For details, see Moving a value and Moving an array. |
| Submap | Applies a global map to top-level complex-type inputs and outputs inside a local map or another global map. For details, see Applying a global map to specific inputs and outputs in a local map. |
| Date, time, string, math, and XPath functions | Convert primitive data types, work with dates, manipulate strings, and apply math functions inside transforms. For details, see Using date, string, math, and XPath functions. |
| Merge | Combine the data from two or more input arrays, of the same or different types, into a single output array of the same or different type. For more details, see Merging input arrays into an output array. |
| Append | Append two or more input arrays of the same or different types to produce one output array of the same or different type. For more details, see Appending input arrays to an output array |

6 Data maps © 2010 IBM Corporation

Here you see a summary of the transformations that are available with WebSphere Business Modeler V7.

The underlined parts in blue are live links that will take you to details for each of the transforms in this presentation.

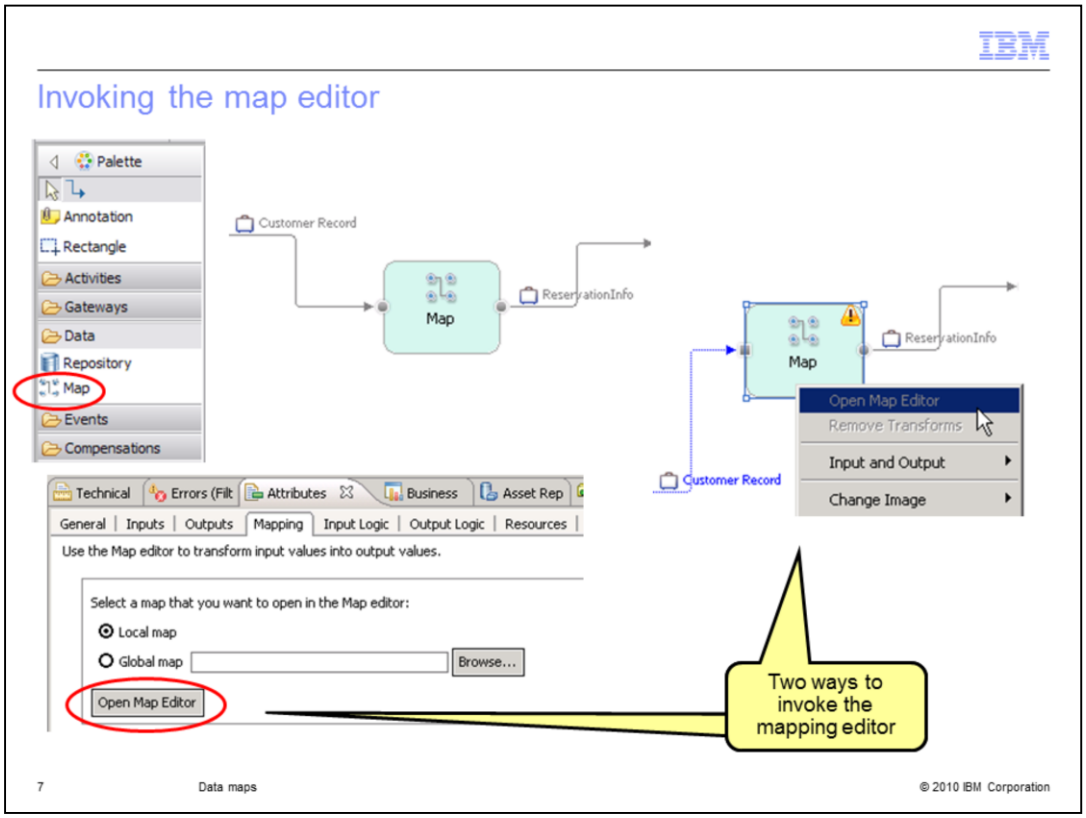
Most of the transformations are self explanatory, such as assign, concat, convert and move. Others are more complex and require further discussion. The ‘for each’ transform is for iterating over arrays. You can choose the starting index for both the input and the output arrays or you can specify a set of indexes to use.

Converting a type means converting the base type into the specific inherited type. This sometimes called down-casting. The convert transform can also be used to convert from one data type to another. If a conversion cannot be made the option is not available.

Submap is the way to use a global map. The mapping from a given input to an output is defined in a global map and then used inside another map as a submap.

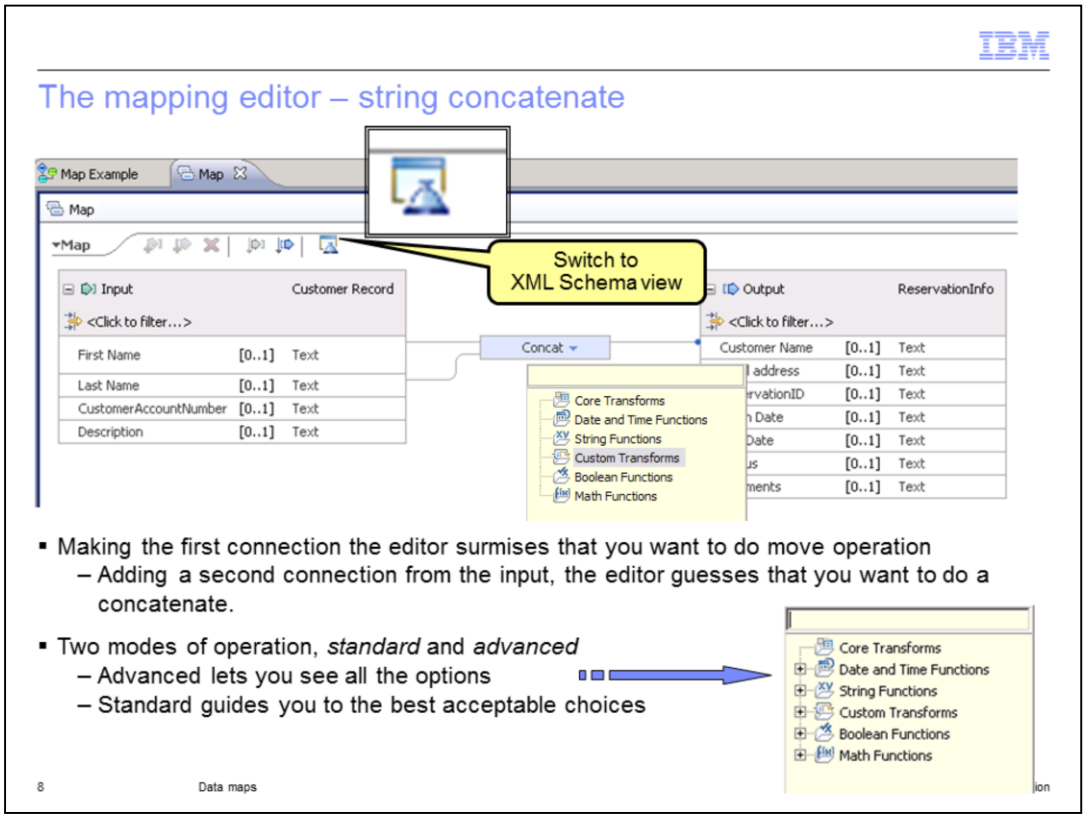
The XPath functions are the advanced functions available when you switch to the XML Schema view. You’ll need to refer to the online information center for the details on how to use them when you’re ready.

The last two transforms, Merge and Append are similar. They are both container transformations and they work as you’d expect from their names. The merge will combine data from two or more inputs into a single structure, merging in the elements. The append transform concatenates the array structures. Depending on the structures, the output array can be sparsely populated when using append. There is a example of both these important new transformations later in this presentation.




To create a local map, select it from the palette and drop it onto the canvas. To insert the map between two elements in an existing flow, just drop the map onto the link. You'll know you're on-target when the link changes to purple. Before you work with the map you have to define the inputs and the outputs and then save your work.

Once your work has been saved, there are two ways to invoke the mapping editor. One is to use the pop-up menu of the element, right-mouse-click, and the other is to use mapping tab in the attributes view. Using the attributes view also provides a way for you to use an existing global map for this top-level map. Global maps are created as a special kind-of business item. We'll see more about how to create them later in this presentation.



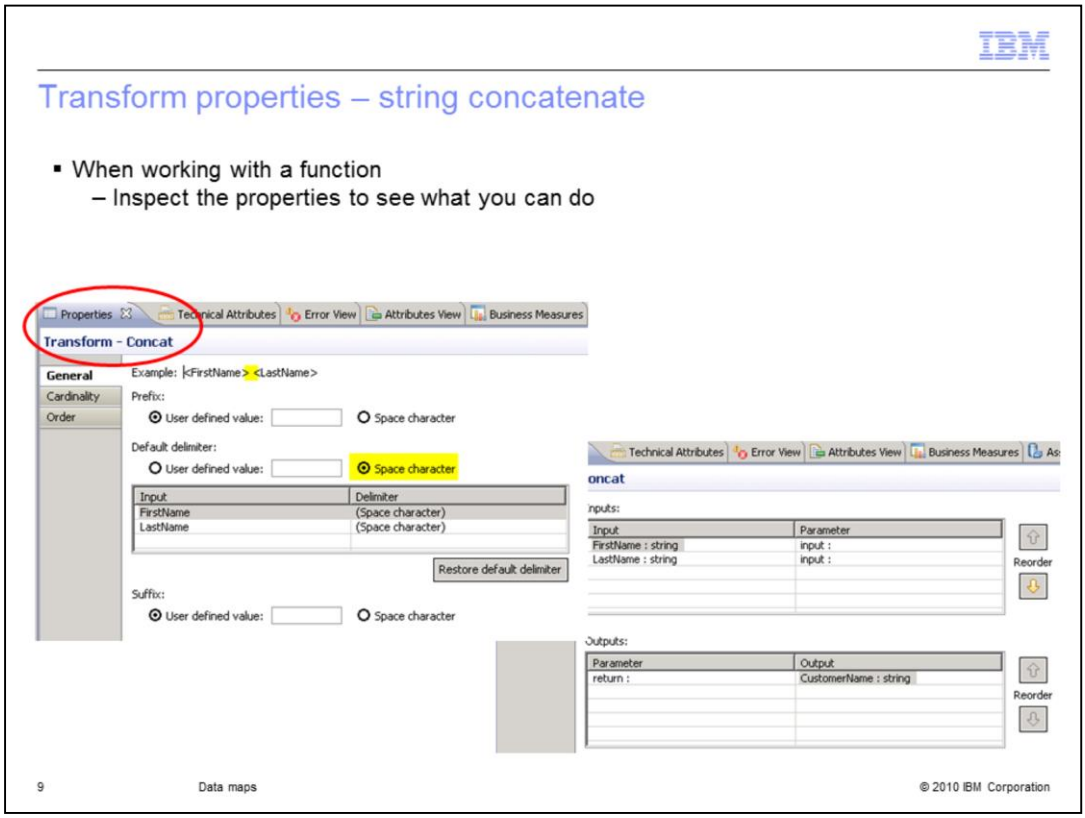

Mapping attributes from one business item to another is a simple matter of making a link from the input side to the output side, moving left to right.

You can move the entire business item with one connection by making a connection from the headers, that is, the gray areas at the top.

The mapping editor anticipates what you want to do based on the context. When you make the first connection, from the first-name on the input to the customer-name on the output, it surmises that you want to do a move. If you then make a second connection, from last name on the input to the input of the transform, the transform is changed to concatenate. To see if there are any other transforms that have two inputs and one output and operate on strings, you can select the drop down arrow on the transform. In this case there are no other transforms you can use. You can tell because there are no plus signs on the items in the list. You see that there might be some string functions but they are not available to you.

If you select the button at the far right of the tool bar, you can switch to the XML Schema view and gain access to the more advanced functions. As shown in the bottom right, the items in the list now have plus signs that can be expanded to reveal the functions available in the different categories.

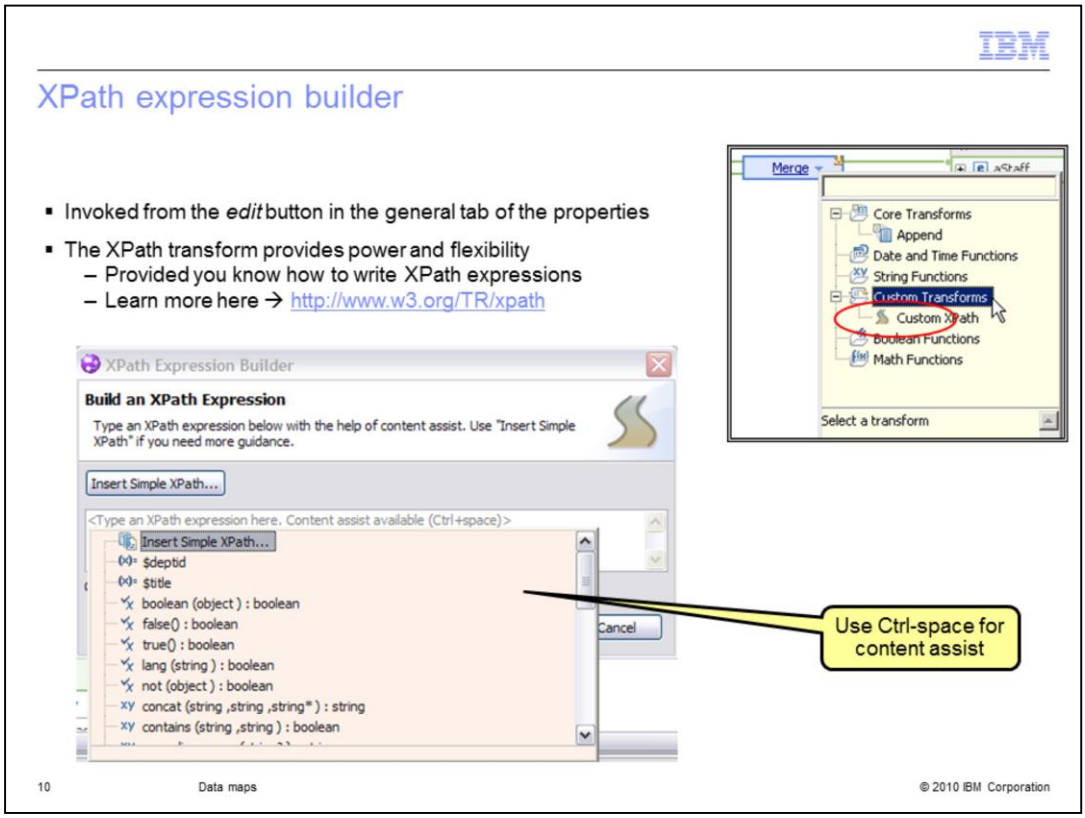
For now though concatenate is what you want to do, but you need to also do a little formatting. You want like to have the first-name first, with a space between the first and the last name. This is managed through the properties associated with the transform.




Here you see the properties associated with the concatenate transform.

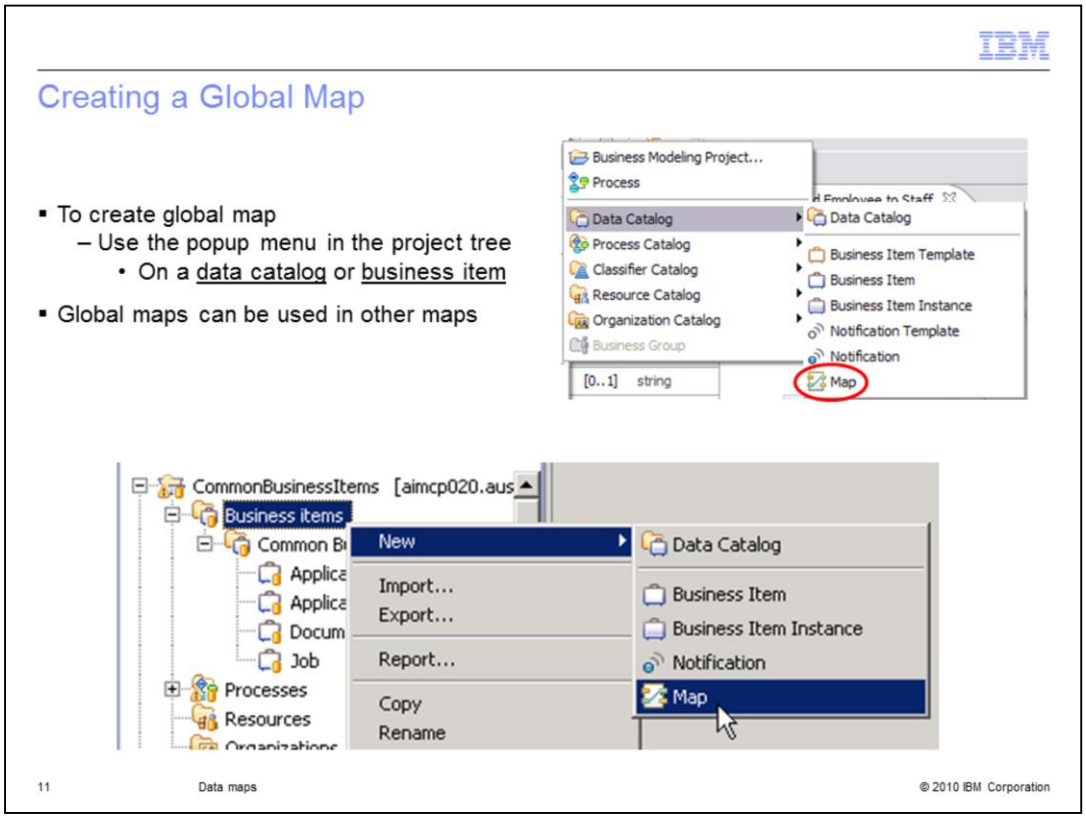
For the concatenate transform, you can see that there are a lot of options. You can add a prefix or a suffix, you can delimit the elements with a space or any character you want. You can also define the order of the elements.

The properties are specific to each transform. Be sure to inspect the properties when working with a transform for the first time.



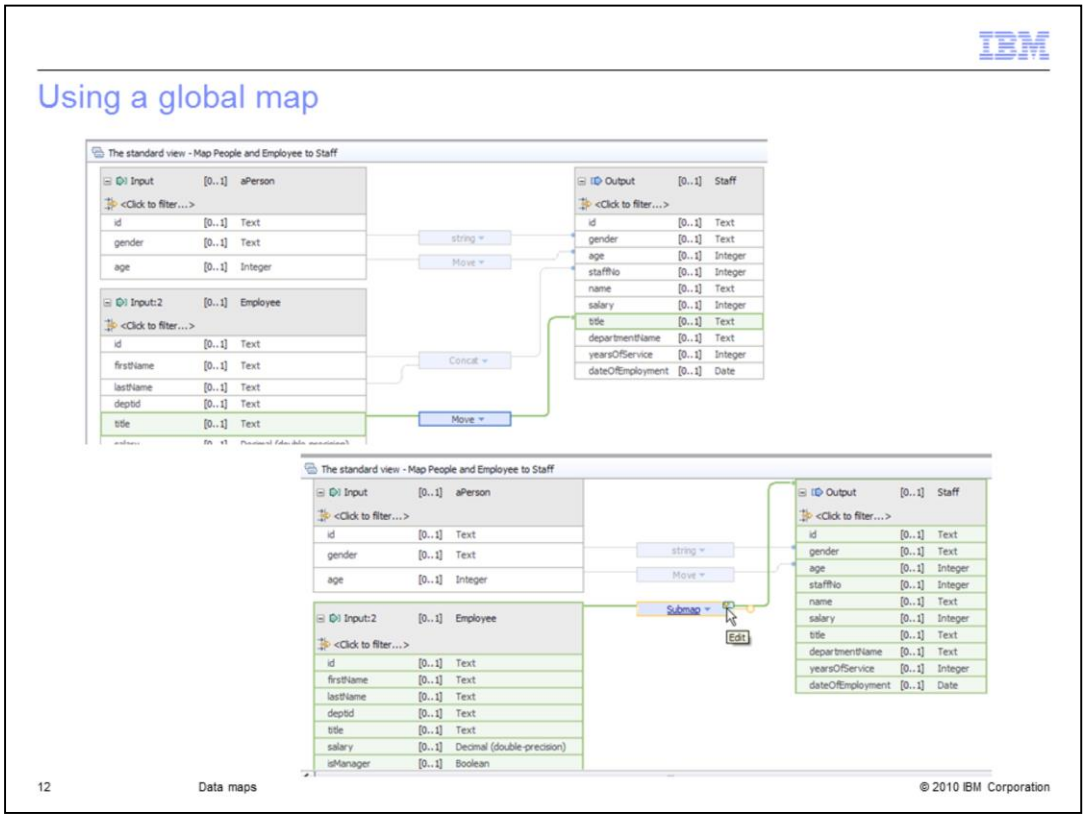

If the transformation you need is not available in the library of pre-defined functions, you can create your own custom transform using the XPath expression builder.

When you select this option, there is an edit button on the far right of the general tab in the properties. This edit button will invoke the XPath expression builder shown here. If you are not familiar with the XPath language you can learn more at the link provided. For creating very simple expressions you can use the button to insert simple XPath. It is very basic. If you are somewhat familiar with the XPath language and syntax and just need a little memory jog, the content assist is available, use control-space when in the edit field.




Creating a global is simple. The thing you need to remember is that it is an element of the data catalog. That is to say it is a kind-of business item.

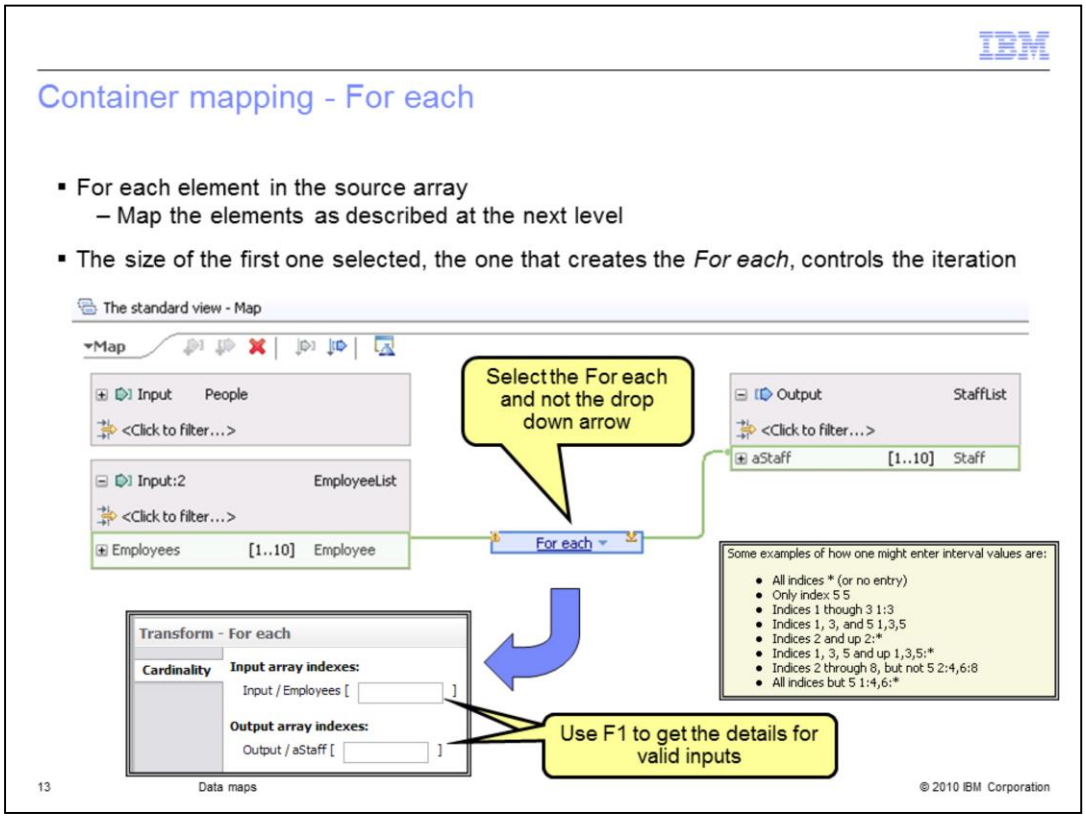
You will need to supply the inputs and outputs and wire it up the same as for a local map. What distinguishes the global map from the local map is that the global map is in the project tree as a business item. This means that you can reference it as a submap in other local or global maps.




In this simple example you can see how the original map has the mapping between employee and staff.

At some point it was noticed that this mapping is very common and done in other maps too.

A global map called EmployeeToStaff was made and then used as a submap, as shown in the bottom screen capture. When you come across a map that is using a submap, to see or edit the submap, just select the submap link or the drop down menu.




Container mapping is a term used to describe the process of iterating over an array of items and mapping the elements as needed, to the target array. There are two parts to this. At this level you define the arrays that you want to map. The properties associated with the 'for-each' transform let you pick the starting index in the source and the target arrays. If the fields are left blank, then the starting index is assumed to be one.

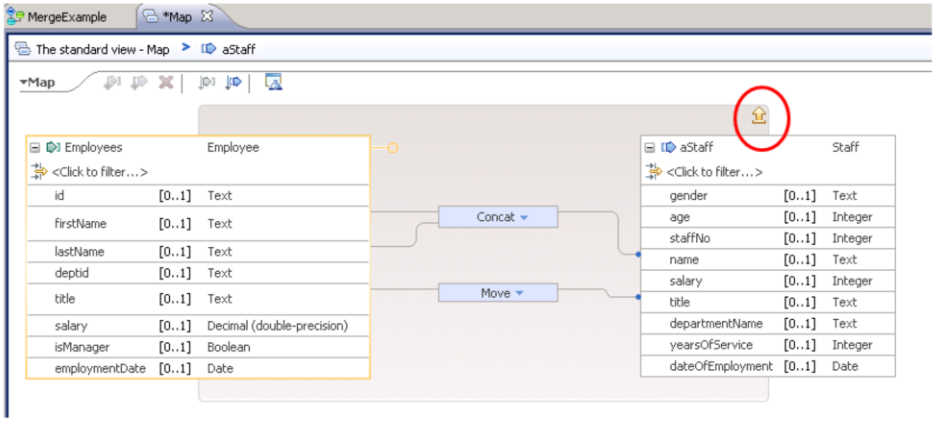
For the input array, you can leave it blank to start at one and do the whole array or you can specify discreet indexes using commas and range operators. To get the whole story, press F1 while the cursor is in the input field.

On the output, you can only specify a single starting index. If you leave it blank, it will start at one.

When you select the 'for-each' link in the transform, a mapping editor is opened on the two arrays.

## For each – internal mapping

- Selecting the '*For each*' will open a map editor with the arrays as the inputs and outputs
  - This is a private '*submap*'. This is to say it is scoped by the '*For each*' loop





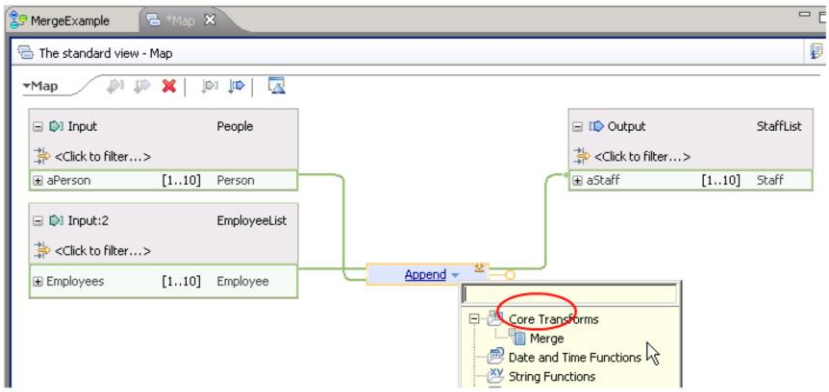


The editor for the nested map in the 'for-each' loop is the same as the regular map editor with the exception of the up arrow, which has been highlighted with the red circle for you. This arrow will take you up one level in the nested hierarchy. Remember, you can have sub-maps and sub-maps with more sub-maps. The arrow will take you to the next enclosing level up.

Using the map editor is the same as was described previously.

## Container mapping – Merge

- Merge, combines two input arrays
  - into a single output array of complex elements or a complex element
- Steps through the input arrays in unison
- Adding the second input will change the *for each* transform to *append*
  - Select the drop down to change it to *merge*
  - Then select the transform to edit and add the mappings




15 Data maps © 2010 IBM Corporation

Merge and append are similar in that they are both container mapping transforms that take two arrays as inputs. The results are quite different though. With the merge, the output can be an array or a single complex object.

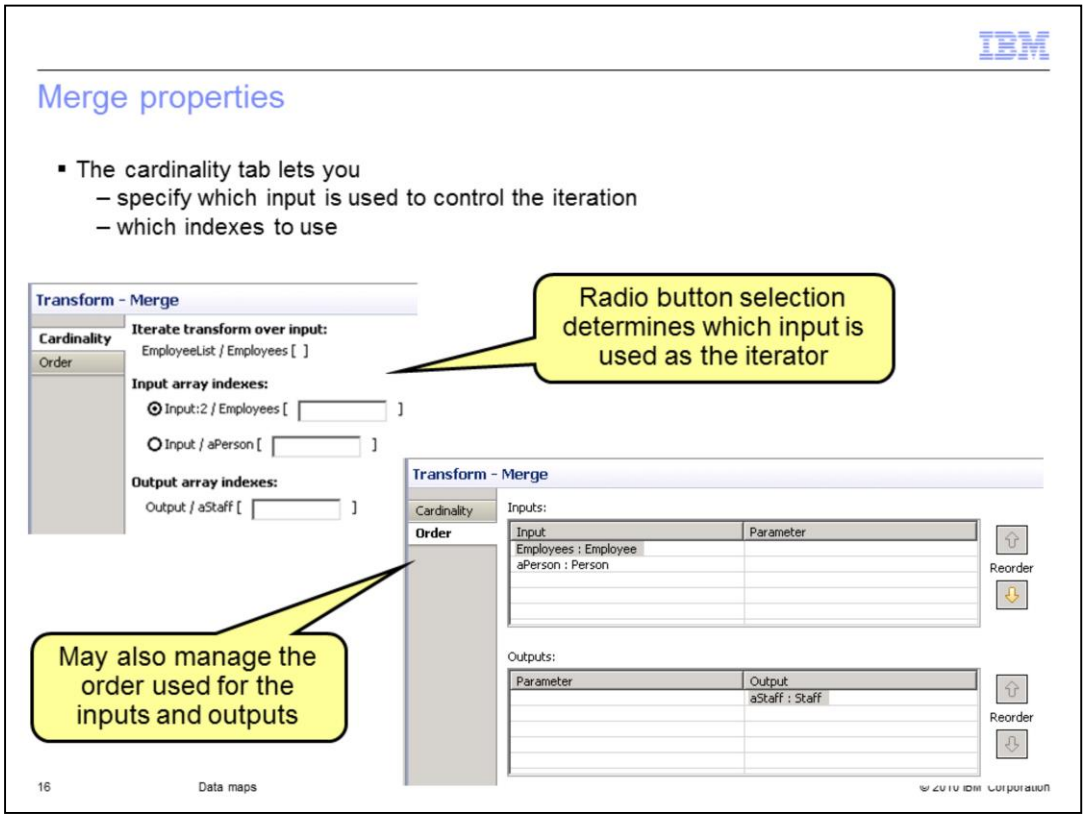
During the execution, the merge transform iterates over each array input in unison.

This means that the merge transform runs through all inputs and outputs in index 1, then in index 2, and so on.

The size of the output array is determined by the size of the input array that is being used as the iterator. The input iterator is defined in the properties of the merge transform. Of course if the output is a single complex object, the size will be a single object.

Given that you have the proper inputs and outputs, when you connect the second input to the ‘for-each’ transform, it will change to the append as shown here. Append is not what you want at this time. You want the merge. Opening the drop down menu on the transform, you’ll see that ‘merge’ is an available option.

Before opening the nested map, take a few moments to look at, and understand the properties associated with the merge transform.




The index is used by the merge transform to determine which input array elements to iterate over and which to skip.

During the execution, the merge transform iterates over each array input in unison.

This means that the merge transform runs through all inputs and outputs in index 1, then in index 2, and so on.

The input arrays can be of different sizes.

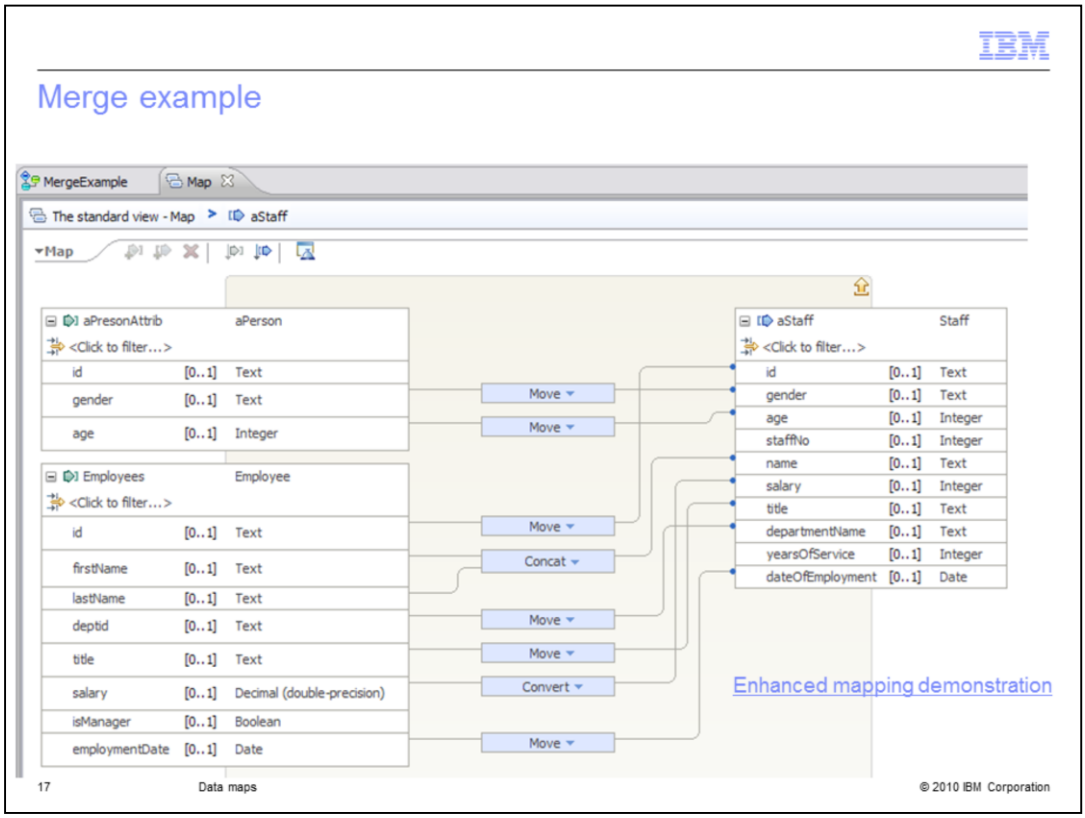
In this case, iteration runs as follows.

If an input array is larger than the size of the input array chosen for iteration, the iteration will stop when the end of the input array chosen for iteration is reached.

If an input array is smaller than the size of the input array chosen for iteration, when the end of the smaller array is reached, subsequent iterations will ignore transforms defined on the smaller array.

The net is that the size of the output array is determined by the size of the input iterator array.

If the order in which the arrays are accessed is important, you can control this too, using the ‘order’ tab in the properties.




Shown here is an example of the nested map used for the merge example.

Notice that the inputs and outputs are for the elemental array types. person, employee and staff.

The staff object takes the age and gender from the person object and the remaining attributes from the employee object.

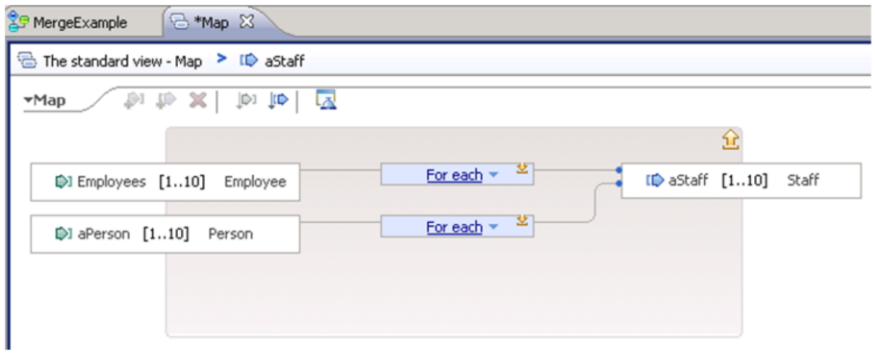
Assume that the People array is the iterator and has eight elements at run-time.

If the EmployeeList has only five elements then the resulting staff array will have eight elements but the last three elements are missing the information that comes from the employee object.

If it were the other way around, the People array has five elements and the EmployeeList has 8, then the resulting StaffList will have five elements and there will be no missing data.

## Container mapping - Append

- Inputs can be arrays or single complex elements
- Output must be an array of simple or complex type
- Creates a new array that is the sum of the input arrays
  - Mapping the elements as specified in the transforms
- Cardinality and ordering the same as with the merge




18 Data maps 

With the append the resulting output is like a concatenation of the inputs. The inputs can be single complex elements or arrays.

In the case shown here, the inputs are of different types, therefore the resulting StaffList is a sparse array.

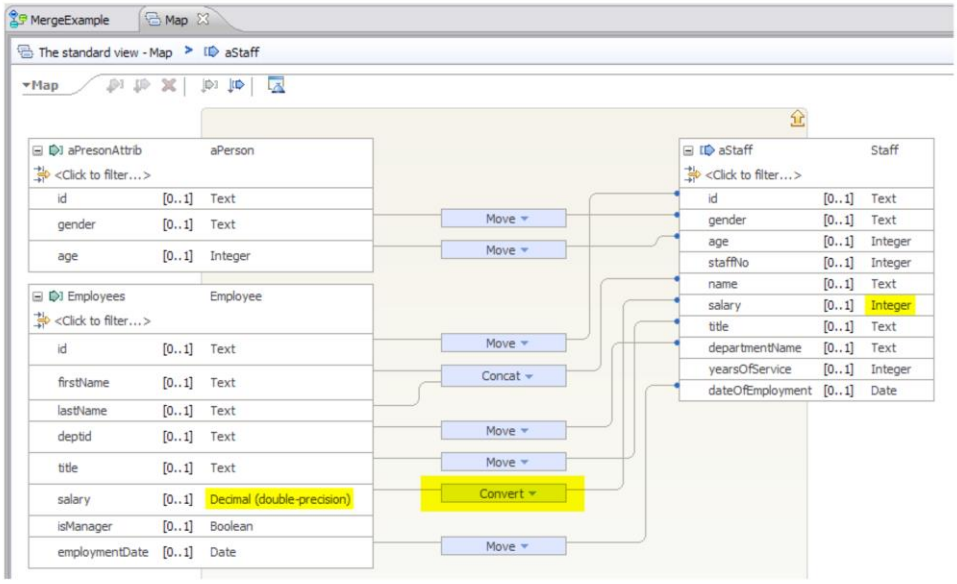
There will be Staff elements that have attributes from the EmployeeList but the attributes mapped from the People array will be empty.

There will also be Staff elements that have attributes from the People array but the attributes mapped from the EmployeeList will be empty.

The size of the StaffList will be the size of the two arrays combined.

This means that if both input arrays were at their max of 10, you will receive a run-time error when running the map.

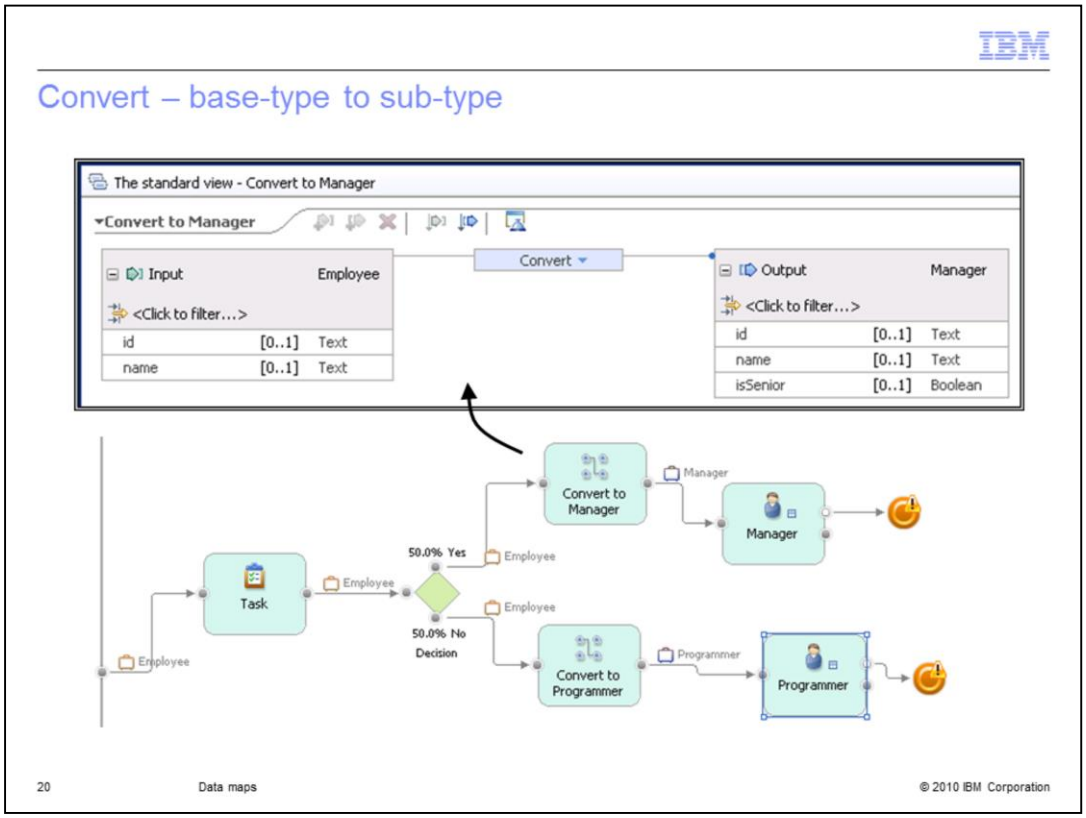
## Convert transform







The convert transform can be used to convert data types during the transformation. Here, a decimal is converted to an integer.




The convert transform is also used for converting a base-type to one of its possible sub-types. This is sometimes called down casting.

It is used in conjunction with a new decision predicate called, “is-instance-of”.

In this example there is a hierarchical data schema for the Employees. The base-type is the Employee and the two sub-types are Manager and Programmer.

The task at the beginning of the flow creates a specific kind of employee based on some business rules. The interface is kept general because there can potentially be many different kinds of employees and from the perspective of the task, it does not care what kind of employee is generated. Keeping the interface general, will make the task more flexible in the long run.

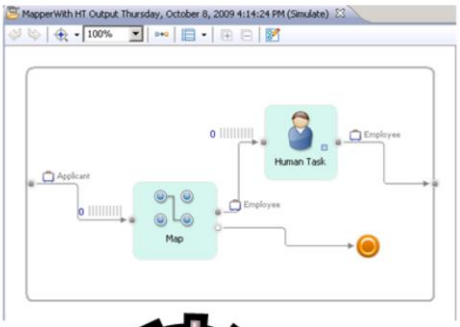
Somewhere downstream in the business process, the kind-of employee does become important. At this point the decision gateway can be used to interrogate the object to find out what it is an instance of, and route it accordingly. To use the object is must still be converted to the proper sub-type, and this is where the map element and the convert transform come into play.

The convert transform will convert the base Employee type to either the Manager or Programmer sub-type. Notice that the connection is between the gray header areas.

## Using simulation to verify transforms

- To make sure that the transform works correctly and to view effects of the transformation on the process
  - run a simulation on the process that contains the map.
- Use a human task and a form to view the results
  - Set preferences to…
    - Method of selecting output path: ***Based on an expression***
    - Enable Form simulation: ***Yes***
- Create one or more instances of the input data
  - Can be fed automatically to the simulation
- This is shown in the "Enhanced Mapping" demonstration

Enhanced mapping demonstration




21 Data maps 

The simulation tool available with WebSphere Business Modeler can also be used to verify the correctness of your mappings and transformations.

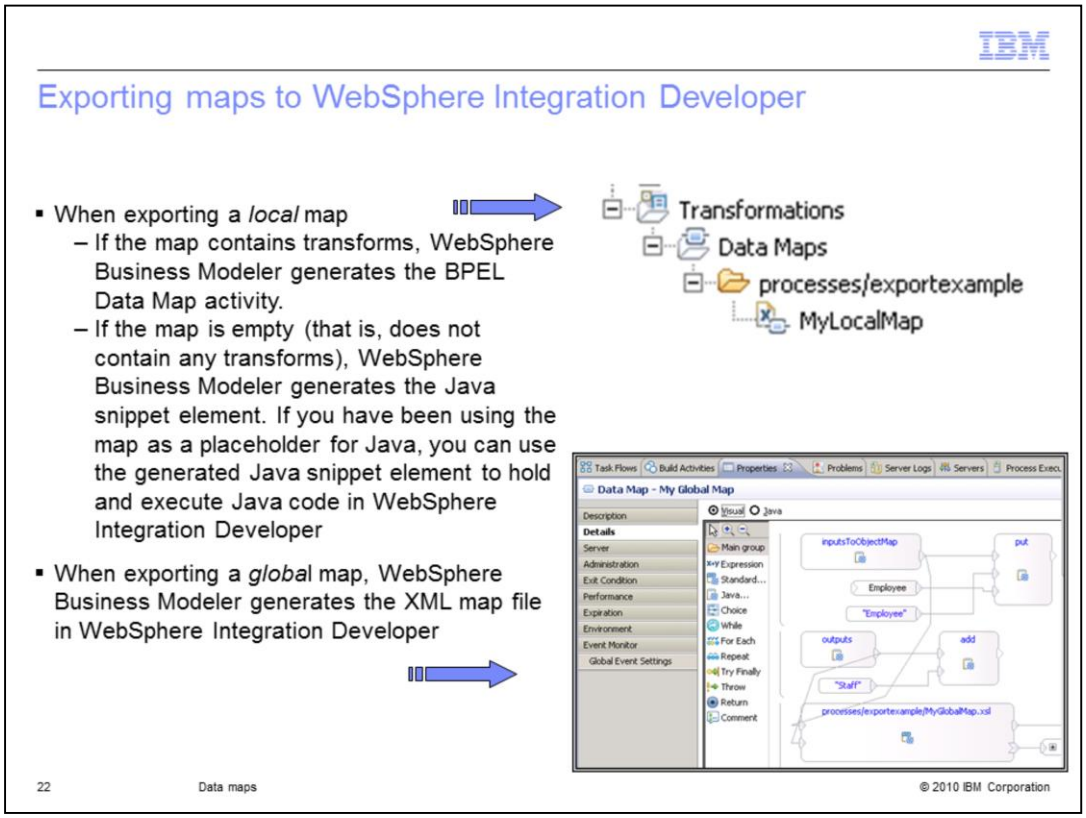
You can use a human task with a form to view and inspect the inputs and the outputs.

To setup the simulation, you will need to set the method of selecting the output path to 'based on an expression'. This tells the simulation engine that you want the use the data that are flowing through the business process to be used when making decisions that will effect the path of the business process flow.

Next you will need to enable 'form' simulation. This indicates that when a human task is reach in the business process flow, that you want to be presented with the input and output forms so you can enter data.

Finally create some instance data that you want to use as input.

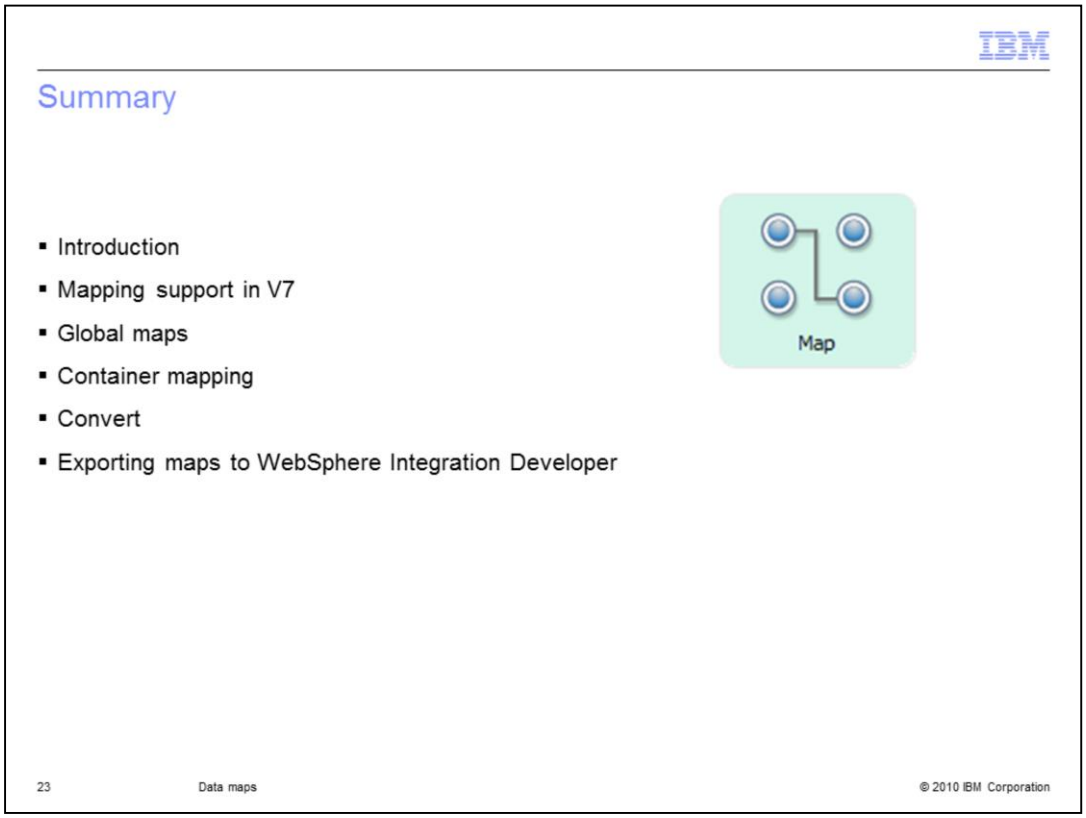
The demonstration that is available will take you through the steps of creating a merge transform and then verifying it using the simulator. The link presented here is to the same demonstration that was available earlier.




There are two kinds of maps available in WebSphere Business Modeler, local maps and global maps.

When the local maps are imported into WebSphere Integration Developer, the maps are represented as BPEL data maps. You can find them in the “Transformations” folder in the project tree. Opening the local map will take you to the mapping editor that is very similar to the one that is available in WebSphere Business Modeler.

The global maps, alternatively, are created as XML map files. These are essentially Java snippets that are accessed through the detailed properties of the map element in the business process flow. You can see in the bottom screen capture the visual representation of the snippet. Selecting the Java radio button will show you the underlying Java code that makes the XSL transform calls.




WebSphere Business Modeler V7 greatly improves the support for data mapping by introducing a subset of the XML mapping tool available with WebSphere Integration Developer.

In this presentation you learned about the new enhanced mapping and how it works. You learned about global maps, how to create them, when to use them and how they are represented in WebSphere Integration Developer. The discussion also covered the details of container mapping, using the ‘for-each’, ‘merge’ and ‘append’ transforms and how to verify the transformation using the simulation feature.

An additional example was presented that showed you how to use the ‘convert’ transform to cast a base-type to a sub-type.

# Trademarks, disclaimer, and copyright information